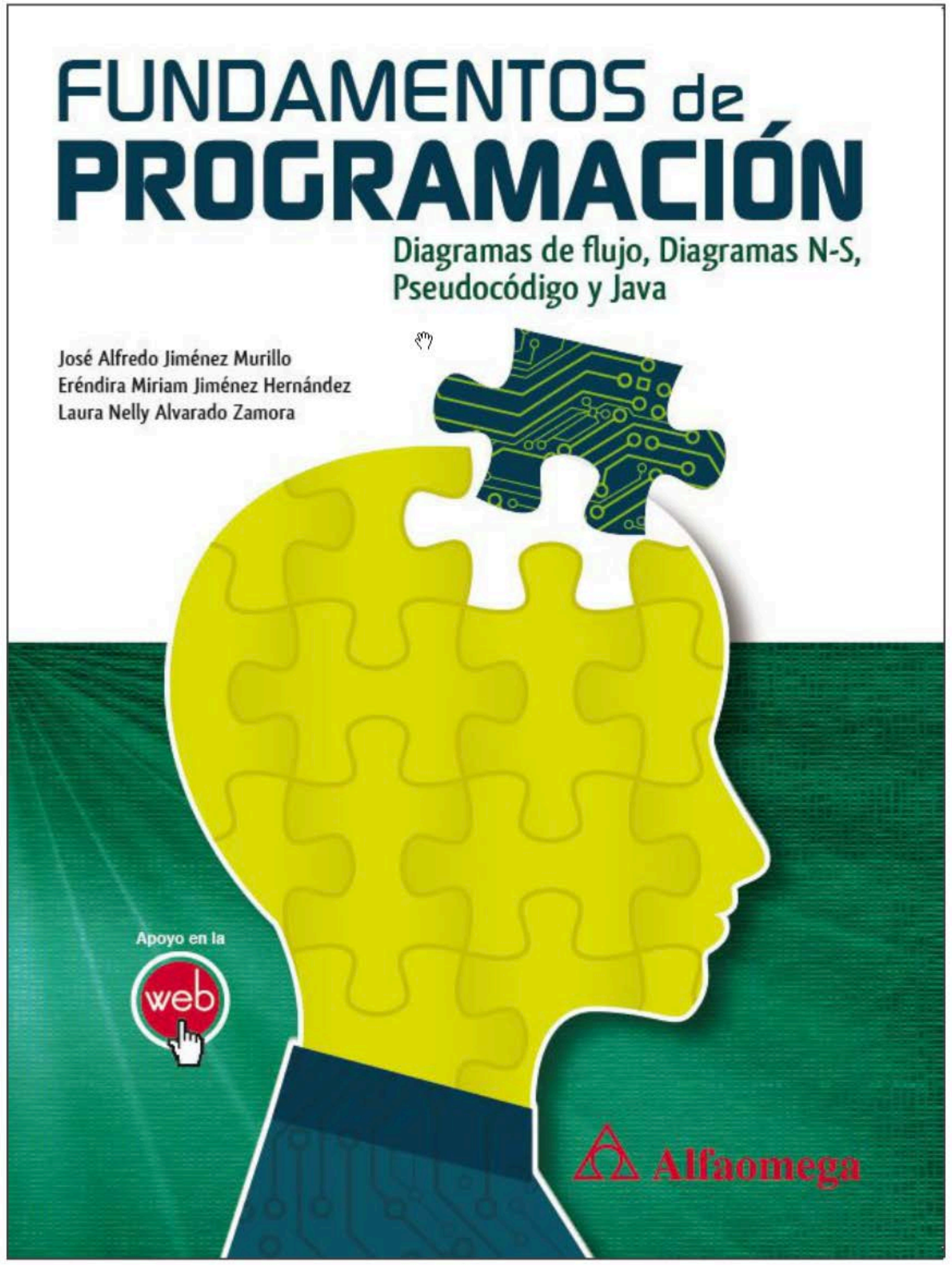
FUNDAMENTOS de PROGRAMACIÓN

Diagramas de flujo, Diagramas N-S, Pseudocódigo y Java

José Alfredo Jiménez Murillo
Eréndira Miriam Jiménez Hernández
Laura Nelly Alvarado Zamora



Fundamentos de Programación

Diagramas de flujo, Diagramas N-S, Pseudocódigo y Java

José Alfredo Jiménez Murillo
Eréndira Miriam Jiménez Hernández
Laura Nelly Alvarado Zamora

Alfaomega
Buenos Aires • Bogotá • México DF • Santiago de Chile

Gerente editorial:
Marcelo Grillo Giannetto
mgrillo@alfaomega.com.mx

Edición:
Gerardo González Núñez
ggonzalez@alfaomega.com.mx

Datos catalográficos

Jiménez, José Alfredo; Jiménez, Eréndira
y Alvarado, Laura

Fundamentos de programación
Diagramas de flujo, Diagramas N-S,
Pseudocódigo y Java

Alfaomega Grupo Editor, S.A. de C.V., México

ISBN: 978-607-622-202-7

Formato: 17 × 23 cm Páginas: 432

Fundamentos de programación. Diagramas de flujo, Diagramas N-S, Pseudocódigo y Java
José Alfredo Jiménez Murillo, Eréndira Miriam Jiménez Hernández y Laura Nelly Alvarado Zamora

Derechos reservados ©Alfaomega Grupo Editor, S.A. de C. V., México.

Primera edición: Alfaomega Grupo Editor, México, octubre 2014

© 2015 Alfaomega Grupo Editor, S.A. de C.V.
Pitágoras 1139, Col. Del Valle, 03100, México D.F.

Miembro de la Cámara Nacional de la Industria Editorial Mexicana
Registro No. 2317

Pág. Web: **http://www.alfaomega.com.mx**
E-mail: **atencionalcliente@alfaomega.com.mx**

ISBN: 978-607-622-202-7

Derechos reservados:
Esta obra es propiedad intelectual de sus autores y los derechos de publicación en lengua española han sido legalmente transferidos al editor. Prohibida su reproducción parcial o total por cualquier medio sin permiso por escrito del propietario de los derechos del copyright.

NOTA IMPORTANTE:
La información contenida en esta obra tiene un fin exclusivamente didáctico y, por lo tanto, no está previsto su aprovechamiento a nivel profesional o industrial. Las indicaciones técnicas y programas incluidos, han sido elaborados con gran cuidado por los autores y reproducidos bajo estrictas normas de control. ALFAOMEGA GRUPO EDITOR, S.A. de C.V. no será jurídicamente responsable por: errores u omisiones; daños y perjuicios que se pudieran atribuir al uso de la información comprendida en este libro, ni por la utilización indebida que pudiera dársele.

Impreso en México. Printed in Mexico.

Empresas del grupo:

México: Alfaomega Grupo Editor, S.A. de C.V. – Pitágoras 1139, Col. Del Valle, México, D.F. – C.P. 03100.
Tel.: (52-55) 5575-5022 – Fax: (52-55) 5575-2420 / 2490. Sin costo: 01-800-020-4396
E-mail: atencionalcliente@alfaomega.com.mx

Colombia: Alfaomega Colombiana S.A. – Calle 62 No. 20-46, Barrio San Luis, Bogotá, Colombia,
Tels.: (57-1) 746 0102 / 210 0415 – E-mail: cliente@alfaomega.com.co

Chile: Alfaomega Grupo Editor, S.A. – Av. Providencia 1443. Oficina 24, Santiago, Chile
Tel.: (56-2) 2235-4248 – Fax: (56-2) 2235-5786 – E-mail: agechile@alfaomega.cl

Argentina: Alfaomega Grupo Editor Argentino, S.A. – Paraguay 1307 P.B. Of. 11, C.P. 1057, Buenos Aires, Argentina,
– Tel./Fax: (54-11) 4811-0887 y 4811 7183 – E-mail: ventas@alfaomegaeditor.com.ar

Para mis hijos: Eren, Pao y Marcelo.
Por su apoyo y cariño.

José Alfredo

A mis papás por ayudarme y darme un gran ejemplo.
A mi mamá por motivarme para alcanzar mis metas.
A Pepe por su amor y apoyo.

Eréndira Miriam

Al null pointer assigment, mi primer motivador a discernir en el área de la programación.
A Andy y Axel, mis entrenadores de paciencia. Los amo.

Laura Nelly

Mensaje del editor

Una de las convicciones fundamentales de Alfaomega Grupo Editor es que los conocimientos son esenciales en el desempeño profesional, ya que sin ellos es imposible adquirir las habilidades, las destrezas y las actitudes necesarias para competir en el mercado laboral. El continuo avance en la ciencia y la tecnología demanda la actualización constante en estos conocimientos.

Alfaomega Grupo Editor tiene como misión brindar a los estudiantes y a los profesionales textos actualizados escritos por especialistas destacados en su campo y que se encuentran enmarcados con una estructura didáctica fundamentada en bases pedagógicas que favorecen el aprendizaje y facilitan su utilización y aplicación en la vida diaria.

Consciente de las exigencias del entorno actual, Alfaomega Grupo Editor ofrece complementos a este medio impreso a través de materiales adicionales en una página web que coadyuvarán, junto con el presente libro, a complementar y desarrollar las competencias necesarias que una profesión determinada requiera para responder a estas demandas.

Esta obra busca desarrollar la lógica de programación del estudiante usando diferentes maneras de representar los algoritmos y presentando interesantes y variados problemas resueltos y para resolver, que ilustran los conceptos revisados.

Acerca de los autores

José Alfredo Jiménez Murillo

Obtuvo el título de Ingeniero Industrial Mecánico en Diseño de Manufactura en el Instituto Tecnológico de Morelia.

Después Obtuvo el grado de Maestría en Ciencias en Cómputo Aplicado en el Centro de Estadística y Cálculo (CEC) del Colegio de Postgraduados (CP), en Montecillo Estado de México, en donde recibió un reconocimiento por su trayectoria académica en este centro de investigación.

Posteriormente obtuvo una segunda Maestría en Ciencias, en la Enseñanza de las Ciencias en el área de Matemáticas en el Centro Interdisciplinario de Investigación y Docencia en Educación Técnica (CIIDET) de la Ciudad de Querétaro Qro. Méx., en donde le fue entregado un reconocimiento por el mejor promedio de la generación. Es autor del libro "Matemáticas para la computación" de la Editorial Alfaomega.

Eréndira Miriam Jiménez Hernández

Nació en Tlaxcala México, en 1987. A los 16 años de edad ganó medalla de bronce en la 9ª Olimpiada Mexicana de Informática. Como parte de su formación académica obtuvo el título de Ingeniera en Sistemas Computacionales con mención honorífica en el Instituto Tecnológico de Morelia y el grado de Maestra en Ciencias de la Computación en el Centro de Investigación en Computación del Instituto Politécnico Nacional (IPN). Actualmente realiza sus estudios de Doctorado en Ciencia e Ingeniería de la Computación en la Universidad Nacional Autónoma de México (UNAM).

Laura Nelly Alvarado Zamora

De origen Michoacano, nacida en la ciudad de Morelia. Realizó sus estudios de Licenciada en Informática y se graduó con mención honorífica en el Instituto Tecnológico de Morelia.

Obtuvo el grado de Maestra en Administración con Orientación a los Sistemas por la Universidad Michoacana de San Nicolás de Hidalgo (UMSNH)

Inició como desarrolladora independiente y desde 1997 se dedica a la docencia en el área de la programación.

Contenido

Antes de comenzar a leer xi
Introducción xiii

Capítulo 1. Conceptos básicos

1.1 Introducción 2
1.2 Computadora 3
1.3 Clasificación del *software* 4
1.4 Algoritmos. 6
1.5 Lenguaje de programación. 7
1.6 Programa. 8
1.7 Programación 8
1.8 Paradigma de programación 9
1.9 Editores de texto 10
1.10 Compiladores e intérpretes 12
1.11 Ejecutables. 13
1.12 Consola de línea de comandos. 14
1.13 Resumen 15
1.14 Problemas 16

Capítulo 2. Algoritmos

2.1 Introducción 24
2.2 Resolución de problemas de programación 24
2.2.1 Análisis de problema 24
2.3 Representación del algoritmo. 26
2.3.1 Descripción narrada 26
2.3.2 Diagrama de flujo 26
2.3.3 Pseudocódigo 46
2.3.4 Diagramas Nassi-Shneiderman (N-S) 51
2.4 Diseño de algoritmos de funciones 58
2.4.1 Funciones en computación 59
2.4.2 Funciones recursivas 63
2.4.3 Procedimientos 67
2.5 Resumen 73
2.6 Problemas 74

Capítulo 3. Introducción a la programación

3.1 Introducción 86
3.2 Lenguaje de programación Java 86
3.2.1 Características del Lenguaje Java 86
3.3 Entornos de desarrollo para Java 87
3.3.1 NetBeans IDE. 87
3.4 Traducción de un programa. 88
3.5 Ejecución de un programa. 88
3.6 Estructura básica de un programa . . . 93
3.7 Elementos del lenguaje Java. 94
3.7.1 Comentarios 94
3.7.2 Tipos primitivos de variables 94
3.7.3 Cadenas de caracteres 96
3.7.4 Definición de variables 96
3.7.5 Identificadores constantes 98
3.7.6 Operadores 98
3.8 Salida de datos. 101
3.9 Lectura de datos 107
3.10 Conversión de tipo de datos 108
3.11 Clase Math 111
3.12 Errores en tiempo de ejecución 117
3.13 Resumen 122
3.14 Problemas 124

Capítulo 4. Control de flujo

4.1 Introducción 130
4.2 Estructuras selectivas: simple, doble y múltiple 130
4.2.1 Instrucción selectiva simple (if) 130
4.2.2 Instrucción selectiva doble (if - else). . 131
4.2.3 Instrucción condicional múltiple (switch) 136

4.3 Estructuras iterativas: mientras, hacer-mientras, desde 139
4.3.1 Mientras (`while`) 139
4.3.2 Hacer-mientras (`do-while`) 143
4.3.3 Desde (`for`) 146
4.4 Ciclos anidados 149
4.5 Manipulación de cadenas. 154
4.6 Diseño e implementación de funciones y métodos 158
4.6.1 Funciones 159
4.6.2 Métodos 163
4.6.3 Recursividad 176
4.7 Resumen 181
4.8 Problemas 182

Capítulo 5. Arreglos

5.1 Introducción 196
5.2 Arreglos unidimensionales: conceptos básicos, operaciones y aplicaciones. 196
5.3 Arreglos bidimensionales: conceptos básicos, operaciones y aplicaciones. 208
5.3.1 Arreglos irregulares 216
5.3.2 Arreglos como argumentos y parámetros 219
5.4 Arreglos multidimensionales: conceptos básicos, operaciones y aplicaciones. 232
5.5 Resumen 234
5.6 Problemas 235

Capítulo 6. Introducción a la programación orientada a Objetos

6.1 Introducción 246
6.2 Orígenes de la programación orientada a objetos. 247
6.3 Beneficios de la programación orientada a objetos 248
6.4 Clases 248
6.5 Métodos 252
6.6 Objetos 255
6.7 Sobrecarga de métodos 266
6.8 Herencia 267
6.9 Otros conceptos del paradigma de la programación orientada a objetos . . . 270
6.10 Resumen 270
6.11 Problemas 270

Capítulo 7. Introducción a las Interfaces gráficas de usuario

7.1 Introducción 286
7.2 Definición y evolución de las Interfaces gráficas. 286
7.3 Proceso de desarrollo de *software* con Interfaces gráficas. 287
7.4 Paquetes gráficos 290
7.5 Cuadros de diálogo 290
7.6 Etiquetas 298
7.7 Cuadros de texto 300
7.8 Botones 305
7.9 Un vistazo a JTextArea y JScrollPane 311
7.10 Resumen 314
7.11 Problemas 317

Respuestas a problemas nones

Capítulo
1 Conceptos básicos 327
2 Algoritmos. 332
3 Introducción a la programación. 347
4 Control de flujo. 350
5 Arreglos. 367
6 Introducción a la programación orientada a objetos. 381
7 Introducción a las interfaces gráficas de usuario 397

Apéndices
A Instalación de NetBeans IDE 409
B Sintaxis para importar una clase de otro paquete 414
C Instrucciones para agregar la biblioteca Absolute Layout 416

Antes de comenzar a leer

En este libro se utiliza la tipografía Courier en los casos en que se hace referencia a código o acciones que se han de realizar en la computadora, ya sea en un ejemplo o cuando se refiere a alguna función mencionada en el texto. También se usa para indicar menús de programas, teclas, URLs, grupos de noticias o direcciones de correos electrónicos.

Material de apoyo en la web

Para tener acceso al material de la página web de apoyo del libro:

1. Ir a la página http://libroweb.alfaomega.com.mx
2. Ir a la sección de catálogo y seleccionar la imagen de la portada del libro, al dar doble clic sobre ella, tendrá acceso al material descargable.

Estimado profesor: Si desea acceder a los contenidos exclusivos para docentes por favor contacte al representante de la editorial que lo suele visitar o envíenos un correo electrónico a webmaster@alfaomega.com.mx

Introducción

El índice de reprobación en las asignaturas que permiten a los alumnos aprender a programar en un lenguaje determinado es muy alto, pero no porque sea difícil aprender un lenguaje de programación o porque sea complicado codificar un algoritmo, sino porque las personas tienen problemas en lógica, les cuesta mucho trabajo crear el algoritmo que permita resolver el problema. Quieren imprimir el resultado antes de calcularlo, o quieren calcularlo antes de tener todos los datos para realizar dicho cálculo. Esperan que la computadora les adivine el pensamiento y se desesperan porque sus programas no realizan lo que se espera de ellos. La verdad es que el principal problema es que no plasman en sus programas las instrucciones necesarias para resolver el problema, bien porque no conocen la infinidad de maneras en que se puede usar una instrucción para que la computadora realice lo que se quiera, o bien porque el orden de las instrucciones no es el adecuado. En ambos casos el problema es de lógica porque no es suficiente saber que existen instrucciones para leer e imprimir información, para ejecutar un bloque de instrucciones varias veces, para seleccionar el conjunto de sentencias se ejecutarán en determinado momento si se cumple una cierta condición. Si las instrucciones no se usan adecuadamente en el momento indicado de nada sirven, o si no se combinan con un poco de imaginación por parte de la persona que las está utilizando para adaptarlas a las diferentes situaciones que se presenten, entonces no tiene ninguna ventaja conocerlas si no se saben usar.

Esta obra está pensada para ayudar a las personas que comienzan una carrera en el área de la computación. Busca desarrollar la lógica de programación del estudiante usando diferentes maneras de representar los algoritmos y resolviendo problemas interesantes para ilustrar y explicar cada uno de los conceptos o instrucciones de los lenguajes de programación, tiene problemas que obligarán a la persona a pensar y ser creativa para resolverlos.

El libro está integrado por siete unidades. En la primera unidad "Concepto básicos" se abordan los conceptos y definiciones de los términos más usados en la programación, entre los cuales se pueden mencionar: computadora, usuario, programa, programación, compilador y su diferencia con intérprete, lenguaje de programación, paradigma de programación, algoritmo, entre otros. El objetivo de esta unidad es familiarizar al lector con las palabras y conceptos propios del área.

La segunda unidad "Algoritmos" tiene como finalidad el análisis de problemas y la representación de su solución por medio de lenguajes algorítmicos. Se representan algoritmos por medio de descripción narrada, diagramas de flujo, diagramas N-S, y pseudocódigo con sus ventajas y desventajas de cada una de estas formas de representación. También en esta segunda unidad se ve la representación de instrucciones matemáticas usando para ello operadores aritméticos básicos y para finalizar se resuelven problemas con ayuda de funciones y procedimientos. Todas las actividades de la unidad se realizan sin apegarse a ningún lenguaje de programación, pero tratando de que los algoritmos aquí representados se puedan codificar en la mayoría de los lenguajes de programación conocidos.

En la unidad tres "Introducción a la programación", se explica la forma de instalar el compilador Java, puesto que se utilizará este lenguaje de programación para el diseño de programas en este libro de texto. Se analiza la estructura básica de un programa en Java. La manera de leer datos de teclado y la forma de imprimir la información en modo consola. Se codifican en Java ecuaciones matemáticas, se estudian los diferentes tipos de datos y se resuelven problemas secuenciales en Java.

En la unidad cuatro "Control de flujo" se estudian y analizan las estructuras de control que permiten a la computadora tomar decisiones y con ello elegir la ruta que se debe seguir en el momento de la ejecución de un programa, en función de las características del problema, los datos conocidos y la presentación del resultado. Se estudian también las instrucciones que permiten ejecutar un bloque de sentencias varias veces. Además en esta unidad se trata a profundidad el uso de métodos, como una forma de segmentar el programa en pequeños subprogramas que realizan una actividad determinada, pero que en conjunto y de forma organizada, son parte de un sistema computacional más complejo.

La unidad cinco "Arreglos" presenta las propiedades y restricciones de estructuras que permiten guardar y procesar gran cantidad información, aprovechando la memoria de la computadora, para manipular los datos contenidos en vectores, matrices y arreglos multidimensionales.

La unidad seis "Introducción a la programación orientada a objetos" tiene como finalidad el introducir al alumno en el paradigma de programación más utilizado actualmente como lo es la programación orientada a objetos. Se explican los diferentes elementos de este paradigma como son: objeto, clase, método, herencia, sobrecarga de métodos, entre otros.

En la unidad siete "Introducción a las interfaces gráficas de usuario" se aborda de forma breve la programación gráfica con la finalidad de que el alumno desarrolle programas más amigables usando diferentes elementos gráficos como son: cuadros de diálogo, etiquetas, campos de texto y botones.

El objetivo de este libro es que las personas que incursionan en el área de la computación aprendan a programar, se utiliza el lenguaje Java como una herramienta para observar los resultados obtenidos al codificar algoritmos que resuelven un problema determinado, pero la finalidad no es aprender Java sino aprender a programar independientemente del lenguaje, pero si además de ello, se aprende un poco del lenguaje más utilizado actualmente, los resultados serán doblemente satisfactorios.

CONCEPTOS BÁSICOS 1

El *hardware* es lo que hace a una máquina rápida; el *software* es lo que hace que una máquina rápida se vuelva lenta.

Craig Bruce

Competencia de la unidad

Comprender, dominar, diferenciar y relacionar los conceptos básicos, de la programación.

Competencias específicas

- Reconocer y diferenciar los conceptos básicos más importantes, entre ellos: computadora, usuario, *software*, *hardware*, sistema operativo, algoritmo, lenguaje, programa, programación, paradigmas de programación, ejecutables, compiladores e intérpretes.
- Conocer el entorno de un lenguaje de programación.
- Ilustrar la relación que existe entre los diferentes conceptos básicos de la programación por medio de mapas conceptuales, mapas mentales y cuadros sinópticos.

Conceptos básicos

Contenido

1.1. Introducción.
1.2. Computadora.
1.3. Clasificación del *software*.
1.4. Algoritmo.
1.5. Lenguaje de programación.
1.6. Programa.
1.7. Programación.
1.8. Paradigma de programación.
1.9. Editores de texto.
1.10. Compiladores e intérpretes.
1.11. Ejecutables.
1.12. Consola de línea de comandos.
1.13. Resumen.
1.14. Problemas.

1.1 Introducción

A principios de la década de los setenta las computadoras eran grandes, costosas y solamente las podían adquirir grandes empresas o universidades. Los usuarios escribían sus programas en equipos fuera de línea como perforadoras de tarjetas o de cintas de papel. Algunas veces las computadoras estaban instaladas en el campus principal de la universidad y los departamentos académicos o instituciones más pequeñas a las cuales apoyaba la universidad, solamente contaban con la perforadora de tarjetas en donde se perforaban las instrucciones de sus programas, las indicaciones iniciales y los datos. Posteriormente, una persona llevaba pequeños paquetes correspondientes a cada uno de los programas al lugar en donde estaban las computadoras para leer la información de las tarjetas; dicha operación se realizaba por medio de un lector de tarjetas que mandaba la información del programa a la memoria de la computadora, posteriormente se compilaba el programa y si no tenía errores se ejecutaba. Por lo general, todos los programas tenían algún tipo de error que no permitía correr el programa, de tal manera que la persona encargada de llevar los programas, regresaba con los mismos en hojas de papel, indicando el tipo de error marcado en la compilación. Posteriormente se perforaban nuevamente otro conjunto de tarjetas para corregir el programa que ya estaba almacenado en el disco de la computadora, las enviaban nuevamente y se hacían las correcciones. Estas nuevas tarjetas por lo regular permitían quitarle algunos errores, pero no todos los que tenía el programa, de tal manera que la actividad de perforación, transportación y lectura se realizaba varias veces antes de ver los resultados de la ejecución del programa. Era muy común que transcurriera mucho tiempo antes de ver un programa funcionando. Los lenguajes más utilizados en esa década eran Basic y Fortran IV.

A principios de la década de los ochentas salieron las primeras computadoras personales; marcas típicas de ese tiempo fueron "Radio Shack" y "Apple". Estas computadoras eran del tamaño de las computadoras de escritorio que ya se conocen actualmente, relativamente baratas y los dispositivos de entrada más comunes eran el teclado, la unidad de disco magnético de 5 ¼ pulgadas de diámetro y el casete (cinta magnética para leer música en esa década), posteriormente el disco de cinco pulgadas sería sustituido por discos de tres y media pulgadas más pequeños y con mayor capacidad de almacenamiento. Las computadoras no tenían mucha capacidad en memoria principal como las de ahora, pero permitían correr los programas.

Los lenguajes más comunes eran Basic y Pascal; aunque todavía se seguían utilizando las grandes máquinas en las empresas que tenían mayores recursos económicos, como las computadoras IBM 4341 e IBM 370 que permitían llevar la administración principalmente en el lenguaje Cobol y PL1. Realmente se tenía que trabajar mucho en ese tiempo para obtener los resultados que ahora se consiguen de una manera muy sencilla; por ejemplo, para hacer una gráfica, dicha figura se construía a base de asteriscos y otro tipo de caracteres. Las computadoras personales permitieron que a fines de esta década ya se contara con los primeros editores de texto, graficadores y hojas de cálculo. En esta década y buena parte de la década de los noventa el sistema operativo más usado fue el MS-DOS.

A mediados de los noventas y los primeros años de este nuevo milenio las computadoras tuvieron un gran desarrollo en la capacidad de almacenaje, velocidad de procesamiento y fueron cada vez más compactas hasta llegar a las que hoy se conocen como *laptops*. Surgieron nuevos dispositivos de almacenamiento secundario como la USB. Internet se convierte en la forma de comunicación más rápida, efectiva y económica. La diversidad de *software* para llevar a cabo todo tipo de tareas en distintas áreas de la ciencia, la industria, la educación y el comercio está disponible. Los sistemas operativos multitarea permiten que las computadoras puedan realizar varias actividades al mismo tiempo y los sistemas multiusuario permiten aprovechar los recursos de la computadora por varios usuarios. Los lenguajes más utilizados fueron: C, C++ y Java. Los sistemas operativos más usados fueron Windows en sus diferentes versiones, Mac Os, Linux y Unix.

En estos primeros años de la segunda década del siglo veintiuno, la comunicación juega un papel importante en las actividades y forma de vida de la sociedad, surgen los teléfonos móviles, el ipod, las cámaras fotográficas, cámaras de video, robots que utilizan la computación y el *software* para realizar infinidad de actividades. El texto, graficas, fotografías y vídeos son elementos fácilmente transportables por medio del correo electrónico y las redes sociales de tal manera que la computación y la computadora están inmersas en todo tipo de actividades a desarrollar. Los sistemas operativos más usados son multitarea y multiusuario, el lenguaje de programación más utilizado para desarrollar *software* web es Java juntamente con el lenguaje de programación C++.

En la computación como en otras áreas de la ciencia es conveniente definir primeramente los conceptos más importantes, la diferencia que existe entre ellos, las similitudes y la relación que existe entre dichos conceptos. Esto permitirá comprender mejor la información y evitará dar un significado erróneo al material que se aborda en esta obra. Por tal motivo en la primera unidad se definen algunos de los conceptos más importantes necesarios en la computación.

1.2 Computadora

El ordenador, la máquina de procesamiento de datos o más comúnmente llamada computadora, es una máquina electrónica que recibe datos de un medio de entrada, procesa dichos datos de una manera rápida y exacta, para posteriormente enviar la información resultante a un medio de salida. La estructura general de una computadora se ilustra por medio del siguiente diagrama:

Medios de entrada. Son las herramientas utilizadas para ingresar todo tipo de datos a la computadora. Los más comunes son: el teclado, ratón, pantalla táctil, CD, DVD, USB, *TouchPad* (sistema usado por las computadoras portátiles), escáner, *joystick*, micrófonos, entre otros.

Medios de salida. Los dispositivos de salida son el equipo con el cual se puede tener la información de la computadora que resulta de un procesamiento de datos o bien de buscar información que está almacenada en la computadora y obtener un reporte de ella organizada en cierto orden. Ejemplo de dispositivos de salida son: monitor, impresora, USB, CD, DVD, plóter, proyector, bocinas, entre otros.

Hay dispositivos que se utilizan como medios de entrada y también como medios de salida; es decir son mixtos, entre ellos se tienen: CD, DVD, USB, módem, pantalla táctil, *router*, entre otros.

Para la entrada, procesamiento y salida de información en una computadora es necesario *hardware* (equipo) y programas (*software*) que permitan el reconocimiento, manipulación y la emisión de la información en forma lógica y ordenada que pueda ser entendida por el usuario.

Software. Son los programas que contienen las instrucciones que permiten el funcionamiento del *hardware*. Son indicaciones intangibles de una computadora. Es el conjunto de programas y procedimientos necesarios para que la computadora lleve a cabo alguna tarea específica.

Hardware. Son las partes tangibles de un sistema computacional (computadora, robot, cámara fotográfica, teléfono móvil, etcétera.) como: teclado, pantalla, *mouse*, impresora, cables, conectores, dispositivos de almacenamiento. Todo aquello que se puede tocar y que sea parte de un sistema computacional es considerado como *hardware*.

Usuario. Es una persona o sujeto que utiliza una computadora, sistema operativo o cualquier sistema computacional. También puede ser una máquina que hace uso de los recursos del sistema para mandar, recibir o procesar información.

1.3 Clasificación del *software*

Actualmente existe *software* que se aplica en la medicina, administración, ingeniería, publicidad, diseño, educación, entre otras áreas. Por lo tanto, se puede encontrar *software* específico de un área de la ciencia, e incluso dentro de esa área se puede encontrar una subclasificación ya que se tiene *software* para cardiología, oncología o ginecología. La mayoría del *software* es desarrollado por compañías que lo venden y obtienen recursos económicos, pero también existe *software* libre (*freeware*) en donde no se paga ninguna cantidad por su uso, o bien *software* en donde solamente se paga una pequeña parte (*shareware*), con lo cual es posible clasificar el *software* de acuerdo a su costo monetario. Sin embargo, todos ellos se pueden agrupar en dos grandes grupos: *Software de sistemas* y *Software de aplicación*.

a) ***Software de sistemas***. Son aquellos programas que permiten administrar los recursos de la computadora. Facilitan la comunicación entre la computadora y el usuario. Es el sistema operativo que permite comunicarse al usuario con la computadora. Se pueden clasificar como:

- *Monousuario*. Son aquellos sistemas operativos que permiten la comunicación entre el usuario y una sola computadora. Se clasifican en:

  ➔ *Monousuario monotarea*. Sistema diseñado para que el usuario lleve a cabo una sola tarea o proceso a la vez (MS-DS, sistema operativo (SO) de la computadora portátil Palm).

  ➔ *Monousuarios multitarea*. En este caso el usuario puede llevar a cabo varias tareas a la vez. Es el sistema operativo de mayor uso actualmente, Windows y MacOS son ejemplos típicos del sistema monousuario multitarea ya que en ambos casos el usuario podría estar escribiendo un documento en un editor de texto al mismo tiempo que baja un video de Internet o imprimiendo una gráfica en un plóter.

- *Multiusuarios*. Sistemas operativos que permiten la comunicación, administración y manejo de los recursos del usuario con varias computadoras. En este sistema operativo dos o más usuarios pueden estar utilizando un programa al mismo tiempo, de tal manera que son multiusuarios y multitareas. Linux, Unix, Windows NT y el sistema operativo del servidor son ejemplos de estos sistemas operativos.

b) ***Software de aplicación***. Son programas para llevar a cabo tareas específicas como: edición de textos, graficación, cálculos, diseños, simulación, entre otras tareas. Se puede clasificar en:

- *Software de uso general*. Es aquel que se puede utilizar para diversas aplicaciones personales, científicas, empresariales, administrativas y de diseño. Los editores de texto, hojas de cálculo, diseño asistido por computadora (CAD) y manejadores de bases de datos, son ejemplo de este *software* de uso general. Se ajustan a las necesidades de cualquier usuario y este *software* se puede conseguir en el mercado.

- *Software de uso específico*. Está diseñado específicamente para tareas particulares de una persona, empresa, organización o institución, como el control de inventarios, procesamiento de nómina, simulación de un proceso y resolución de problemas particulares. Se diseña y

desarrolla de acuerdo a las necesidades particulares de un usuario específico. Generalmente se encarga el diseño y desarrollo de este tipo de *software* a personas o compañías desarrolladoras de *software*.

El Sistema operativo (SO). Es el programa cuya finalidad es organizar el trabajo de la computadora. El sistema operativo permite llevar a cabo tareas básicas para el buen funcionamiento de la computadora como:

- Reconocer las señales de entrada de algún dispositivo (teclado, *mouse*, tacto, disco, USB, etcétera).
- Enviar señales de salida a diferentes elementos del sistema (monitor, disco, impresora, teléfono móvil, plóter, etcétera.).
- Administrar la información guardada en carpetas y archivos de la computadora (copiar archivos, borrar archivos, crear nuevas carpetas, cambios de nombre de archivos, entre otras).
- Controlar y mantener la comunicación con los equipos periféricos (teclado, monitor, impresora, disco duro, escáner, ratón, USB etcétera.).
- Administrar la memoria de la computadora. Esto permite que un *software* funcione en dos computadoras que tienen diferentes memorias y capacidad de almacenamiento.
- Diagnosticar el funcionamiento del *software*. Indica el estado de los programas así como detectar mal funcionamiento de los mismos.
- Controlar la secuencia en que se deben llevar a cabo los procesos y tareas a realizar.

Los sistemas operativos han evolucionado mucho. Existen diferentes características ventajas y desventajas de cada uno de ellos. La recomendación de un sistema operativo depende de los recursos con que cuenta la computadora, las aplicaciones a usarse y el número de usuarios. A continuación se presenta una lista de los sistemas operativos más destacados del área de la computación.

MS-DOS. Salió a principios de la década de los años 80 del siglo pasado, fue desarrollado por Microsoft y se hizo popular debido a la gran cantidad de *software* que se podía ejecutar con él, al procesador Intel y a su compatibilidad con IBM.

Windows. Microsoft crea Windows 3.0 en 1990 con la finalidad de tener una interfaz gráfica más amigable con íconos que representan las diferentes operaciones que se pueden ejecutar con un solo click o doble click apoyándose como apuntador ejecutor al *mouse*. Esto permitió un manejo y administración de los archivos amigable y fácil, porque no era necesario que el usuario tuviera que recordar las instrucciones y la sintaxis de las mismas para ejecutarlas. Posteriormente Microsoft crea otras versiones de este SO como son Windows 3.1, Windows 95, Windows Vista, Windows XP , Windows 7 y Windows 8 que tienen la misma filosofía pero con una mayor diversidad de operaciones.

OS/2. IBM desarrolló el OS/2 con una interfaz muy buena y fácil de manejar, pero que no tuvo mucha suerte debido al poco *software* y aplicaciones que se pueden ejecutar con este sistema operativo, ya que la mayoría del *software* del mercado ha sido monopolizado por Microsoft.

Mac OS. Este sistema operativo es tan amigable para el usuario, que cualquier persona puede aprender a utilizarlo en poco tiempo aún sin tener antecedentes de otros sistemas operativos. Fue desarrollado por Apple Computer, Inc., y es la base de la popularidad de las computadoras Macintosh.

Windows NT. Microsoft desarrolla también el SO Windows NT que permite trabajar con un *software* al mismo tiempo en una red de usuarios aprovechando de esa manera los beneficios de Windows en redes de computadoras. Especialmente recomendado para estaciones de trabajo y servidores en los

cuales se necesita un máximo de rendimiento. Sistema seguro mediante un sistema de acceso a la red que utiliza una base de datos llamada SAM (Security Account Manager) para guardar la información de seguridad. La primera versión de NT fue lanzada en 1993, pero en la familia de los sistemas operativos Windows NT también están Windows Vista, Windows XP, Windows Server 2003, Windows Server 2008 y Windows 7.

Unix. Es un sistema operativo multiusuario y multitarea que corre en diferentes computadoras: personales, minicomputadoras, supercomputadoras y redes de computadoras. Fue desarrollado por los laboratorios Bell de AT&T en 1969 y es un SO que se usa con frecuencia en la red Internet, es portable ya que permite la instalación del mismo en diferentes computadoras.

Linux. Fue lanzado al mercado a principios de la década de los noventas tomando en cuenta las sugerencias de programadores. Linux está escrito en el lenguaje de programación C, además de pequeñas secciones de código en lenguaje ensamblador. Linux es portable y funciona en sistemas muy diversos como: computadoras personales, redes, iPAD, teléfonos móviles, entre otros. Es rápido, seguro, eficaz y fiable sin que una aplicación pueda causar problemas a las otras. Es multitarea ya que se pueden estar ejecutando varias tareas a la vez y no se tendrá que esperar a que termine una cosa para hacer otra. Es multiusuario ya que varias personas pueden usar la misma computadora a la vez, compartiendo el microprocesador (imprimiendo, escribiendo, jugando, navegando en Internet). Es libre, es decir; no tiene ningún costo.

1.4 Algoritmo

El programador de computadoras es una persona que resuelve problemas de una manera rigurosa y sistemática, usando como herramienta principal a la computadora. El programador establece la secuencia de pasos que debe seguir la computadora para resolver un problema, esta secuencia de pasos recibe el nombre de *algoritmo*. Existen varias definiciones de algoritmo, a continuación se presentan algunas de ellas:

a) Conjunto finito de operaciones a realizar para resolver un determinado problema.
b) Conjunto de operaciones y procedimientos que deben seguirse para resolver un problema.
c) Conjunto finito de pasos, precisos y ordenados para resolver un problema. La palabra *finito* es importante ya que el algoritmo debe estar integrado por un número razonable de pasos; solamente los necesarios. *Precisos*, porque cada instrucción debe ser clara y concisa. *Ordenados* porque los pasos se organizan de una manera lógica y ordenada.

Los algoritmos se pueden representar por medio de frases que describen cada uno de los pasos que integran el algoritmo. Ejemplo.

Algoritmo para ir a la escuela

	Operación (Paso)		
	Inicio	5	Me visto
1	Me despierto	6	Me desayuno
2	Me levanto de la cama	7	Me lavo los dientes
3	Me desvisto	8	Tomo el transporte
4	Me baño	9	Al llegar a la escuela me bajo del transporte
			Fin

Algoritmo para comprar zapatos

	Operación (Paso)
	Inicio
1	No he comprado zapatos
2	Mientras no he comprado zapatos hacer Inicio mientras
3	Selecciono los zapatos
4	Me los pruebo
5	Si me quedan y me gustan entonces Los compro
6	Fin mientras
7	Me entregan los zapatos
	Fin

Todo algoritmo comienza con un "*Inicio*" y termina con un "*Fin*". En el **algoritmo para ir a la escuela** cada una de las instrucciones se ejecuta una sola vez. Sin embargo en el **algoritmo para comprar zapatos** se observa que las operaciones de las líneas 3, 4 y 5 se ejecutan varias veces, hasta que se compren los zapatos y deben encerrarse entre las palabras "*Inicio mientras*" y "*Fin mientras*" se dice que este algoritmo contiene un ciclo marcado claramente por la instrucción *"Mientras"*. Siempre que se requiera de la ejecución de varias operaciones estas estarán dentro de un ciclo perfectamente delimitado. La información en la línea: "Si me quedan y me gustan entonces "Los compro" es una instrucción condicional que también es característica de los algoritmos computacionales para tomar decisiones.

Otra forma de representar algoritmos es por medio de pseudocódigo en donde las instrucciones están conformadas por palabras en algún idioma para indicar la acción a realizar y alguna proposición o expresión matemática. La instrucción para mandar los mensajes correspondientes de "Aprobado" o "Reprobado" en caso de que se cumpla o no que calificación sea mayor o igual a 70 es:

```
si (calificación>=70)
    imprimir " Aprobado"
  sino
    imprimir "Reprobado"
```

Las palabras en negritas (**`si`**, **`imprimir`**, **`sino`**) son parte del pseudocódigo que indican la acción a realizar, la palabra "calificación" es un identificador o variable cuyo valor determina si se cumple o no la condición. Los mensajes "Aprobado" y "Reprobado" encerrados entre comillas son textos que la computadora despliega en caso de que se cumpla o no la proposición. En fin; el pseudocódigo es un lenguaje algorítmico que utiliza palabras de algún idioma (español, inglés, francés, según sea el caso) juntamente con expresiones matemáticas para representar las operaciones de un algoritmo.

También se pueden representar los algoritmos por medio de diagramas de flujo, por medio de Diagramas N-S y otras formas de lenguaje algorítmico. Se abundará más al respecto en la unidad 2 de este libro.

1.5 Lenguaje de programación

Un lenguaje es un conjunto de símbolos (o palabras) y métodos para estructurar y combinar dichos símbolos. Un lenguaje también recibe el nombre de idioma y como tal consta de todos los símbolos válidos por dicho lenguaje y los métodos para estructurar correctamente cada una de las palabras, frases y ora-

ciones. Esta clase de lenguaje recibe el nombre de lenguaje natural. Ejemplos de este tipo de lenguajes o idiomas son el: Español, Inglés, Francés, etcétera.

Lenguajes formales. Existen lenguajes de menor capacidad para simular y modelar lenguajes naturales, tal es el caso de los lenguajes Java, C, Pascal, entre otros; que se utilizan para la comunicación con las computadoras. A este tipo de lenguajes se les llama lenguajes formales. Pero también son lenguajes formales aquellos que utilizan símbolos gráficos (cuadrados, rombos, rectángulos, círculos, flechas) como sucede en los organigramas y diagramas de flujo, así mismo lo son aquellos que usan perforaciones de puntos y rayas para la comunicación como el sistema braille, o bien aquellos que solamente hacen uso de algunas de las palabras de un idioma (Leer, Imprimir, Hacer, Ejecutar, Si, entonces, Mientras, etc.) que permiten indicar los pasos de un algoritmo que será transformado posteriormente en un programa que entienda la computadora.

Lenguaje de programación. Conjunto de reglas sintácticas y semánticas que permiten la comunicación con una computadora. Los lenguajes formales: Binario, Java, C, Pascal, Basic, etc. son lenguajes de programación que obedecen a un grupo de reglas sintácticas y semánticas que permiten determinar si una instrucción es parte de un lenguaje y la actividad que deberá realizar la computadora con determinada instrucción.

1.6 Programa

Programa. Es un conjunto de instrucciones basadas en un lenguaje de programación que una computadora ejecuta para resolver un problema o realizar una función específica.

Generalmente los programadores elaboran un algoritmo que contiene los pasos para la solución de un problema usando alguna forma de representación algorítmica, pero las operaciones de un algoritmo se deben traducir a instrucciones que entienda la computadora, para ello el programador utiliza el editor de texto de un lenguaje determinado. Los programas se deben guardar en algún dispositivo de almacenamiento secundario (Disco, USB) con la finalidad de que se conserve la información de dicho programa cuando se apaga la computadora o se cambia de aplicación.

Al programa escrito por el programador se le llama "*programa fuente*", este programa ya lo entiende la computadora pero no lo puede ejecutar hasta que se lleve a cabo la compilación, para determinar que esté correctamente escrito. Si al hacer la compilación se detectan errores se deberán corregir, y una vez que no tiene errores ese programa fuente la computadora lo convierte en un "*programa objeto*" escrito en lenguaje máquina que puede ejecutar perfectamente la computadora. En resumen, para resolver un problema primeramente se representa como algoritmo, después se traduce a un lenguaje de programación y posteriormente se compila dicho programa usando el compilador del lenguaje seleccionado.

1.7 Programación

Programación. Es el lenguaje que los programadores usan para comunicar instrucciones a una computadora y poder lograr de esa manera que la computadora lleve a cabo alguna actividad. Sistema de escritura para la descripción precisa de algoritmos o programas informáticos. Es el proceso de codificar, depurar y mantener el código fuente de programas computacionales.

Un programador debe elaborar programas de calidad. En la programación para que un programa se considere de calidad deberá tener los siguientes factores:

- *Conciso*. Los programas deben tener solamente las instrucciones necesarias para resolver un problema específico de una manera óptima. Esto significa que deben ser lo más pequeños y concretos posibles.

- *Claro*. Deben ser fáciles de revisar, entender, corregir y actualizar con la finalidad de enriquecerlos y darles mantenimiento.
- *Eficaz*. Los programas deben cumplir el objetivo para el que fueron creados, en el menor tiempo posible y utilizando los menores recursos de tiempo y memoria.
- *Portable*. Deben poderse ejecutar en diferentes plataformas (*hardware* o *software*) distinto a aquel en que se desarrolló dicho programa.

Para programar se requiere del dominio del lenguaje de programación a utilizar, bases matemáticas y sentido común.

1.8 Paradigma de programación

Es una propuesta tecnológica adoptada por una comunidad de programadores cuyo núcleo central es incuestionable en cuanto a que unívocamente trata de resolver uno o varios problemas claramente delimitados. Los tipos de paradigmas de programación más comunes son:

- ***Imperativo o Procedimental***. Es el más común de los paradigmas. En este paradigma se le debe indicar a la computadora cada paso a realizar para resolver un problema determinado. Lenguajes clásicos para este paradigma son: C, Pascal, Basic, Fortran, etcétera.
- ***Lógico***. Gira en torno al concepto de proposición y predicado. Consiste en expresar formalmente problemas complejos y resolverlos mediante la aplicación de hipótesis, reglas de inferencia, tautologías y teoremas. Es por esta razón que la programación lógica es atractiva donde la programación tradicional ha fracasado. Un lenguaje típico para este paradigma es Prolog.
- ***Estructurada***. Este paradigma sostiene que todo programa puede ser desarrollado utilizando únicamente tres instrucciones: Secuencia, Instrucción condicional e Iteración (ciclo). Si bien los lenguajes de programación tienen mayor cantidad de instrucciones, éstas se puede construir a partir de las tres instrucciones citadas, haciendo de esta manera innecesarias instrucciones como: goto, return, exit sub, entre otras. Lenguajes de programación que usaban con regularidad estas instrucciones eran: Basic y Fortran.
- ***Modular***. La programación modular fue creada para resolver problemas más grandes que la programación estructurada no puede atacar. Está basada en el lema "divide y vencerás" que consiste en dividir un programa grande en módulos o subprogramas más pequeños, con el fin de hacerlo más claro y manejable. Los procedimientos y funciones que integran un programa grande se podrían considerar como módulos aunque no son los únicos. La combinación del paradigma estructurado y modular es una forma de programar que todavía usan los desarrolladores de *software* con lenguajes de antaño con muy buenos resultados apoyándose en lenguajes como: Basic, Pascal, C, Cobol y PL1.
- ***Programación Orientada a Objetos (POO)***. Tiene su origen en el lenguaje para realizar simulaciones Simul67. La POO se convirtió en la forma de programación más usada a finales de los ochenta y principios de los noventa en gran parte al lenguaje C++. Las características de POO fueron agregadas a varios lenguajes de programación existentes de esta época como: Ada, Basic, Lisp y Pascal, lo cual llevó a problemas de compatibilidad y en la capacidad de mantenimiento del *software*. Para evitar este problema se crearon otros lenguajes como

Eiffel el cual a su vez fue reemplazado por Java y PHP, en gran parte debido a la aparición de Internet. En la POO se manejan nuevos conceptos como: clase, herencia, método, evento, mensaje, abstracción, encapsulamiento y polimorfismo, que la distinguen de otros paradigmas de programación. La POO sigue siendo hasta ahora uno de los paradigmas más usados por los desarrolladores de *software*.

- ***En la nube***. Este paradigma fue propuesto por las grandes compañías desarrolladoras de *software* entre ellas: Microsoft, Google y Amazon AWS. Es un paradigma que proporciona servicios de computación a través de Internet. El usuario accede a un catálogo de servicios estandarizado que se adapta de manera flexible a los requerimientos particulares de cada uno de ellos. Los usuarios pagan solamente aquellos servicios que hayan recibido. Entre sus ventajas principales están: la prestación de servicios a nivel mundial, actualizaciones automáticas del *software*, no es necesario la instalación de nuevo *hardware*. Sus principales desventajas son: centralización de las aplicaciones y dependencia de los proveedores de servicio, la disponibilidad de las aplicaciones está ligada al servicio de Internet, vulnerabilidad y robo de la información.
- ***Funcional***. Está basado en la utilización de funciones matemáticas a las cuales se les manda parámetros y se obtiene de dichas funciones un resultado. Se puede decir que la mayoría de los lenguajes conocidos utilizan de una manera u otra el paradigma funcional porque utilizan funciones matemáticas, pero hay *software* que se caracteriza por apegarse más a este modelo como son las hojas de cálculo, en donde el tratamiento de la información se realiza prácticamente en un 100% usando para ello funciones.

Existen otros paradigmas de programación como: *orientada a aspectos, declarativa, orientada a componentes*, entre otras, pero las citadas anteriormente son las de mayor uso por los desarrolladores de *software* actualmente.

Algunos lenguajes de programación pueden soportar y trabajar perfectamente con varios paradigmas de programación, entre ellos se tienen a: Java, JavaScript, C++, C#, PHP, Scala, etc. De tal manera que al citarlos anteriormente no implica que son exclusivos para trabajar con esos paradigmas.

1.9 Editores de texto

Antes de que existieran los editores de texto, los datos e instrucciones de un programa se proporcionaban a la computadora por medio de tarjetas o cintas de papel perforadas. Posteriormente salieron los teletipos que estaban provistos de un rollo de papel y un teclado, esto permitía escribir una instrucción o comando usando el teclado y registrando dicho mensaje en papel. Cada línea que se mandaba era analizada y revisada por la computadora y en función de ella llevaba a cabo la operación solicitada o bien determinaba si dicha línea era una palabra válida del lenguaje.

Posteriormente con la aparición del monitor fue posible la edición de varias líneas de texto a la vez. Los editores de texto fueron usados en sistemas operativos para enviar los comandos e instrucciones de manejo y administración del equipo de cómputo, pero sobretodo porque permiten la edición de complejos programas. La mayoría de los compiladores e intérpretes tienen un editor de texto que permite escribir sus programas, guardar la información en dispositivos secundarios, compilar y ejecutar dichos programas. Algunas de las funciones básicas que permiten llevar a cabo los editores de texto son:

- **Marcaje de párrafos**. La información marcada posteriormente será sometida a una nueva operación como copiar, borrar, pegar, colorear, etcétera.

- **Copiar, borrar, cortar, pegar**. Son operaciones que se llevan a cabo con frecuencia cuando se está editando un programa, independientemente del lenguaje en que se esté codificando dicho programa.
- **Colorear**. Es posible cambiar la presentación de texto usando colores, subrayado, negritas, etcétera.
- **Rehacer y Deshacer**. Por lo general el editor de texto lleva un registro de las últimas operaciones que se han llevado a cabo, de tal manera que si el usuario se arrepiente el editor le ayude a rehacer o deshacer la operación.
- **Importar información**. Por medio de esta opción es posible insertar el contenido o parte de información de un segundo archivo en el texto que se está editando actualmente.
- **Filtrado de información**. La mayoría de los editores de texto permiten llevar a cabo operaciones a párrafos de textos, por ejemplo ordenar las palabras de dicho párrafo en forma alfabética.
- **Búsqueda y reemplazo**. Es posible buscar una palabra y sustituirla por otra.

Es posible observar que todas estas operaciones y otras muchas más, se pueden llevar a cabo con los conocidísimos procesadores de texto pero, ¿cuál es la diferencia entre un editor de texto y un procesador de texto? La verdad es que la línea que los divide es muy fina e incluso se podría decir que son iguales. Véase la definición de ambos.

Editor de texto. Es un programa que permite crear y modificar archivos digitales compuestos únicamente por texto plano sin formato. Los editores de texto más comunes son los que tienen los lenguajes de programación para la edición de sus programas, aunque las operaciones que se pueden llevar a cabo con ellos rebasan por mucho a las antes mencionadas.

Procesador de texto. Tiene una amplia gama de posibilidades para el manejo y la presentación del texto, entre ellas: tipo y tamaño de letra, formateo de párrafos, efectos artísticos, corrector de ortografía, diccionario en varios lenguajes, intercalado de imágenes, entre otras. Ha habido varios procesadores de texto desde que salieron al mercado a principios de la década de los ochenta, pero el procesador de texto más utilizado fue el "*Word Perfect*" de la Suite de Corel, que funcionaba bajo la plataforma MS DOS. Este procesador de texto fue posteriormente sustituido por "*Microsoft Word*" de Office bajo la plataforma Windows.

Lenguajes de alto nivel. Es aquel que permite al programador escribir las instrucciones de un programa utilizando palabras de un idioma. Algunas de las palabras del idioma inglés que son usadas para la estructuración de instrucciones en diferentes lenguajes son: begin, case, write, if, input, then, else, while, for, repeat, integer, real, etcétera. Por ejemplo la instrucción en Java:

```
if (calificación >= 70)
System.out.print ("aprobado");
```

Le indican a la computadora que si la *calificación* tiene un valor mayor o igual a 70 debe imprimir el mensaje "*aprobado*". En esta línea de código **`if`** y **`System.out.print`** son palabras reservadas del lenguaje de programación, pero también son palabras del idioma inglés que el programador conoce. Algunos de los leguajes de alto nivel más utilizados son: Ada, Basic, C++, C#, Cobol, Fortran, Java, MATLAB, Pascal, Perl, Python, PHP, entre otros.

Lenguajes de bajo nivel. Son lenguajes que la computadora puede entender en el momento de ejecutar un programa y cuyas instrucciones son difícilmente entendidas por el ser humano. Dentro de los lenguajes de bajo nivel se tienen dos:

- ***Lenguaje máquina***. Las instrucciones de este lenguaje están integradas por 0s y 1s. Tiene la ventaja de ser un lenguaje más rápido que los lenguajes de alto nivel, pero tiene la desventaja de ser difícil de entender, usar y manejar, porque los códigos son cadenas de ceros y unos muy grandes, de tal manera que si se presenta un error es complicado de detectar y corregir.
- ***Lenguaje ensamblador***. Es derivado del lenguaje máquina y está formado por abreviaturas de palabras y números llamados mnemotécnicos. Como ventaja con respecto al lenguaje máquina es que los códigos fuentes son más cortos ya que en lugar de ceros y unos usa palabras. Por ejemplo, si se quiere sumar los números 67 con 25 y asignarlos a una localidad de memoria, se tiene el segmento de programa en lenguaje ensamblador.

```
mov AX, 67 (Mueve el 67 al acumulador AX (AX=67))
mov BX, 25 (Mueve 25 al registro base BX (BX=25))
add AX, BX (Suma al acumulador AX el registro Base BX (AX=AX+BX))
```

Donde `mov` y `add` son mnemotécnicos de las palabras Mover y Adición, `AX` es un acumulador y `BX` es un registro base, ambos de uso general.

La desventaja del lenguaje ensamblador sigue siendo la misma que tiene el lenguaje máquina ya que es complicado de entender, sus instrucciones son limitadas, se tiene que escribir mucho para tener pocos logros.

Evolución de los leguajes de programación.

Los lenguajes fueron cambiando haciéndose cada vez más sencillos de entender, aplicar, mayor abstracción y más poder en sus sentencias. La evolución de los lenguajes de programación se puede dividir en 5 etapas o generaciones.

- ***Primera generación***: Lenguaje máquina.
- ***Segunda generación***: Primeros lenguajes ensambladores.
- ***Tercera generación***: Lenguajes de alto nivel. Ejemplos de estos lenguajes son: C, C++, C#, Pascal, Cobol, PL1, Ada, Delphi, Java, etcétera.
- ***Cuarta generación***. Son los lenguajes capaces de generar código por sí solos. Aquí se ubican los lenguajes de mayor abstracción, que reutilizan partes del código de otros programas para la creación y desarrollo de nuevos programas. Ejemplo: Clipper, FoxPro, Visual, DataFlex, FOCUS, NATUAL, SQL, MATLAB, SAS, etcétera.
- ***Quinta generación***: Aquí se encuentran los lenguajes orientados a la inteligencia artificial. Estos lenguajes todavía están poco desarrollados. Ejemplo de ellos son: LISP, Prolog, OPS5 y Mercury.

1.10 Compiladores e intérpretes.

Son programas utilizados para analizar, determinar e interpretar el contenido y significado de un programa fuente. Tienen cierto parecido pero no se pueden considerar iguales.

- **Compilador**. Analiza el programa fuente y lo traduce a otro equivalente llamado lenguaje objeto, escrito por lo general en lenguaje máquina, que puede entender y ejecutar la computadora. Es como aquella persona que toma un documento escrito en un idioma lo traduce y lo escribe en otro idioma (entendible para la persona que desea leer su contenido). La principal ventaja de los compiladores es que es posible crear programas más rápidos y eficientes ya que la revisión del programa se hace en forma global una sola vez, de tal manera que cuando el programa está compilado y se generó el programa objeto, la ejecución del mismo es rápida debido a que se encuentra en lenguaje máquina. Ejemplo de lenguajes que se consideran como compiladores son: Fortran, Pascal, C, C++, C#, Clipper, ADA,

Programa fuente. Archivo escrito generalmente en un lenguaje de alto nivel.

Intérprete. Analiza cada línea del programa fuente que se va escribiendo y lo traduce directamente sin generar ningún código equivalente. Es como aquella persona que traduce cada una de las frases que pronuncia un conferencista y que a la par que el conferencista habla, el intérprete dice la frase equivalente en el idioma que puede ser comprendido por la audiencia que está en la conferencia. Entre las principales ventajas de los intérpretes se pueden mencionar: que los errores de los programas se detectan inmediatamente después de haber escrito la instrucción, facilitando de esa manera la corrección de los mismos y que el programa puede modificarse sobre la marcha sin volver a comenzar la ejecución. Las desventajas son que debido a que el programa se revisa y ejecuta línea por línea es más tardada la revisión y ejecución del mismo. Ejemplo de algunos lenguajes que se pueden considerar intérpretes son: PHP, Perl, JavaScript, Logo, Basic, Snobol, MATLAB, entre otros.

Algunos lenguajes se pueden considerar como lenguajes intermedios debido a que el programa fuente al ser compilado se transforma en un archivo no tan sencillo de entender por el usuario, pero que tampoco puede ser ejecutado por la computadora, dado que no se encuentra en su lenguaje, sino que requiere de un intérprete para trasformar la información en lenguaje máquina al momento de la ejecución. Ejemplo de este tipo de lenguajes intermedios son: Java, Lisp y Python.

Extensión de un archivo. Por lo general el nombre de un archivo está integrado por dos cadenas de caracteres separadas por un punto.

ventas.doc

La primera de las cadenas (en este caso ventas) es el nombre propiamente dicho del archivo y la segunda (doc) es la extensión cuya función principal es diferenciar el contenido del archivo, de tal manera que el sistema operativo pueda determinar el procedimiento necesario para ejecutarlo o interpretarlo, algunas de las extensiones más comunes son: sys, exe, class, jar, xls, entre otras.

1.11 Ejecutables

Ejecutable es un programa o archivo que contiene instrucciones en código máquina y que puede ejecutarse en una plataforma determinada. Pero también un ejecutable puede ser un archivo con instrucciones en *bytecode* que requiere de un intérprete para ser ejecutado.

Los programas ejecutables en código máquina funcionan bajo una plataforma específica dado que en su ejecución hacen llamadas a funciones específicas de un sistema operativo. Ejemplo de este tipo de archivos son los que tienen la terminación "exe, com, dll, y bat" en Windows y los que tienen la terminación "com" en MS-DS. Por lo general este tipo de archivos ejecutables son un medio de transmisión de virus, dado que los virus atacan principalmente a funciones en código máquina, alterando directamente las cadenas de ceros y unos del código.

La característica de un archivo ejecutable es que no es necesario abrir el compilador para ejecutar el programa, sino que se puede correr aunque la computadora no tenga instalado el compilador en que fue creado ese programa o archivo ejecutable.

A los programas ejecutables que están integrados por instrucciones que son interpretadas por un programa, se conocen como "Scripts". La instrucciones que conforman a los Scripts son portables (se pueden ejecutar en varias plataformas, Windows, Linux, Mac OS, FreeBSD, Unix, etc.). Por ejemplo un ejecutable en Java es portable porque sus instrucciones son bytecode no asociadas a un procesador en concreto y que por lo tanto pueden ser ejecutadas en diferentes plataformas, se dice que son programas "multiplataforma".

Un programa con instrucciones bytecode puede ser ejecutado usando como intérprete un programa que comúnmente se conoce como "máquina virtual". Un programa ejecutable en java utiliza la JVM (Java Virtual Machine) para interpretar las instrucciones del mismo.

1.12 Consola de línea de comandos

También conocida como Interfaz de Línea de Comando (CLI, *Command Line Interface*). Es un área de un *software* (una ventana) en donde se teclea una serie de comandos u órdenes en modo texto para editar archivos o ejecutar programas. En un CLI es posible modificar, editar o ejecutar programas por medio de texto plano.

Cada instrucción se escribe en una línea de texto y se ejecuta al presionar ENTER, aunque también es común escribir varias instrucciones de un programa como líneas de texto y al final ejecutar ese programa. Los programadores por lo general hacen uso de un CLI para escribir sus programas, los guardan en un archivo de texto y posteriormente lo compilan y ejecutan para lograr un fin específico.

La consola de línea de comandos existe desde inicios de la computación y se usa en diferentes sistemas computacionales como lenguajes de programación y sistemas operativos. Se tienen CLIs para diferente *hardware* y con diversos fines. Posiblemente la generación de usuarios de Windows no lo recuerde, pero antes de Windows, el sistema operativo más usado era MS DS el cual era un CLI con mucho éxito que todavía es posible acceder a él desde Windows con el comando *Todos los programas/Accesorios/Símbolo del sistema*. Algunos ejemplos del MS DS se muestran a continuación:

Tabla 1.1.

Comando	Operación
dir	Lista los archivos y directorios (carpetas) contenidos en la carpeta actual.
del nomina.xls	Borra el archivo nomina.xls de la carpeta actual.
rd azul	Suprime la carpeta azul.

(*continúa*)

Tabla 1.1. (*continuación*)	
Comando	**Operación**
cd c:\manzana\pera	Cambia o se posiciona en la carpeta "pera" que está contenida en la carpeta "manzana" de la unidad "C".

Actualmente estas actividades se llevan a cabo de una manera muy fácil, por ejemplo el comando "dir" lo ejecuta automáticamente la computadora al posicionarse en una carpeta, la instrucción "del nomina. xls" se ejecuta si se señala el archivo nomina.xls y después se le da la orden de suprimir dicho archivo. Sin embargo "cd c:\manzana\pera" requiere de posicionarnos en la carpeta o subdirectorio "pera" avanzando poco a poco hasta llegar a ella, actividad que con MS DS se lleva a cabo al ejecutar una sola línea de texto. Algunas veces se requiere elaborar un programa integrado por varios comandos de este tipo y entonces Window tiene algunos problemas.

Los CLIs son muy útiles; aunque es necesario recordar la sintaxis de las instrucciones o comandos para lograr una comunicación con la computadora. Para los que están acostumbrados a usar interfaces gráficas, un CLI le resultará complicado o ineficiente, pero por medio de ellos se pueden tener programas más potentes, rápidos y efectivos cuando las interfaces gráficas no tienen todas las opciones o símbolos requeridos para llevar a cabo alguna operación a nivel bajo o intermedio.

1.13 Resumen

A principios del siglo XX el barco, el automóvil y el ferrocarril fueron los medios de transportación más usados, posteriormente se les unió el avión. El correo, el telégrafo y después el teléfono fueron los medios que permitieron la comunicación entre las personas. En todo el siglo anterior los cambios no fueron muy significativos si se comparan con los que han ocurrido en esta primera década del siglo XXI. El Internet vino a revolucionar la forma en que se comunican las personas, esto ha hecho que desaparezcan los telégrafos porque a nadie le interesa enviar telegramas en donde la información es limitada y exclusivamente a base de texto, pudiendo usar el correo electrónico para mandar texto, imágenes y videos en forma rápida y económica, información a la cual se puede acceder desde cualquier parte del mundo sin necesidad de estar en el domicilio para enterarse de la correspondencia.

Ahora los automóviles, trenes y aviones son rápidos y pueden ser manejados en forma automática. La forma de comunicación telefónica también ha cambiado sustancialmente dejando atrás aquellos teléfonos que solamente funcionaban conectados a un cable y en un domicilio particular, para dar paso a esa telefonía móvil que puede tomar fotografías, videos, escuchar música, es posible acceder a Internet, con todo lo que eso significa; y al final también hacer llamadas telefónicas. Las pláticas ya no solamente son por medio del teléfono sino que se les unió a él, el chat y las redes sociales para una comunicación en tiempo real. Se puede tener una conversación escuchando no solamente su voz sino también viendo su imagen y movimientos.

Anteriormente lo que ocurría en Europa se conocía tiempo después por medio de periódicos, revistas o en el mejor de los casos por la televisión una vez que se trasportaban las imágenes y videos a las televisoras y periódicos. Ahora lo que sucede en los lugares más lejanos de la tierra se puede ver en el mismo momento que está sucediendo. Los avances en la comunicación y transportación han globalizado al mundo, las empresas son internacionales y lo que se produce puede comercializarse de una manera relativamente fácil.

En todos estos avances la electrónica y la computación han tenido un papel fundamental, ya que el equipo está implementado principalmente a base de circuitos electrónicos y funcionan por medio de programas de computación.

Sin embargo todavía hay mucho que hacer ya que cada día se requieren computadoras más accesibles, pequeñas, agradables, fáciles de manejar, con una mayor capacidad de almacenamiento y procesamiento de información, para poder integrarse en todos esos equipos aplicables a todas las áreas de la ciencia.

Pero también es necesario el *software* que permita obtener los mejores resultados de ese nuevo *hardware*; de tal manera que seguirá desarrollándose *software* de base y *software* de aplicación porque será necesario para el equipo existente y el de nueva creación.

Anteriormente el *software* se realizaba sin ninguna planificación y a la medida para cada una de sus aplicaciones. El *software* era desarrollado y utilizado por la misma persona o empresa que lo creaba. Anteriormente el *hardware* era muy caro en relación al *software* que muchas de las veces era gratis, ya que se entregaba en la compra del equipo. Sin embargo el equipo es cada vez más barato mientras que el costo del *software* es cada vez mayor. Aunque los grandes desarrolladores de *software* están tendiendo a crear monopolios, es tan grande la demanda de *software* que es imposible satisfacerla, porque todas las empresas y personas requieren aprovechar la computadora para realizar nuevas tareas aún no consideradas por los grandes grupos desarrolladores de *software*.

De tal forma que sistemas operativos, procesadores de texto, hojas electrónicas y *software* de aplicación en general, seguirán utilizándose. Se crearán nuevos lenguajes que permitirán desarrollar *software* que permita el manejo de nuevo equipo, de tal manera que existe un campo de aplicación importante para la computación y los programadores de computadoras

1.14 Problemas

1. Relacionar los conceptos con su definición colocando en el paréntesis el número que corresponda.

Definición	Concepto	
1. Equipo con el cual se puede obtener la información de la computadora que resulta de un procesamiento de datos o bien información que está almacenada en la computadora.	Sistema operativo	( )
2. Es una persona o sujeto que utiliza una computadora, sistema operativo o cualquier sistema computacional	Medios de Entrada	( )
3. Programas que permiten administrar los recursos de la computadora	*Hardware*	( )
4. Máquina electrónica que recibe datos de un medio de entrada, procesa dichos datos de una manera rápida y exacta, para posteriormente enviar la información resultante a un medio de salida	*Software* de sistemas.	( )
5. Conjunto finito de pasos, precisos y ordenados para resolver un problema.	Lenguajes formales	( )
6. Conjunto de programas y procedimientos necesarios para que la computadora lleve a cabo alguna tarea específica	Windows NT.	( )
7. Conjunto de instrucciones basadas en un lenguaje de programación que una computadora ejecuta para resolver un problema o realizar una función específica.	Medios de salida	( )

8.	Lenguajes de menor capacidad para simular y modelar lenguajes naturales.	Paradigma de programación	( )
9.	Tiene una amplia gama de posibilidades para el manejo y la presentación del texto, entre ellas: tipo y tamaño de letra, formateo de párrafos, efectos artísticos, corrector de ortografía, diccionario en varios lenguajes, intercalado de imágenes, entre otras.	Programación.	( )
10.	Programas para llevar a cabo tareas específicas como: edición de textos, graficación, cálculos, diseños, simulación, entre otras tareas.	Lenguaje máquina.	( )
11.	Analiza el programa fuente y lo traduce a otro equivalente llamado lenguaje objeto.	Computadora	( )
12.	Programa o archivo que contiene instrucciones en código máquina y que puede ejecutarse en una plataforma determinada.	Lenguaje de programación.	( )
13.	Lenguajes orientados a la inteligencia artificial. Estos lenguajes todavía están poco desarrollados. Ejemplo de ellos son: LISP, Prolog, OPS5 y Mercury.	Windows.	( )
14.	Herramientas utilizadas para ingresar todo tipo de datos a la computadora.	Programa.	( )
15.	Sistema operativo de código libre, portable, multiusuarios, escrito en lenguaje C, que fue lanzado al mercado a principios de la década de los noventas tomando en cuenta las sugerencias de programadores.	*Software*	( )
16.	Permite trabajar con un *software* al mismo tiempo en una red de usuarios aprovechando de esa manera los beneficios de Windows en redes de computadoras.	Editor de texto.	( )
17.	Programa cuya finalidad es organizar el trabajo de la computadora.	Algoritmo	( )
18.	Sistema de escritura para la descripción precisa de algoritmos o programas informáticos.	*Software* de aplicación.	( )
19.	Conjunto de reglas sintácticas y semánticas que permiten la comunicación con una computadora.	Compilador.	( )
20.	Tangibles de un sistema computacional	Usuario	( )
21.	Permite al programador escribir las instrucciones de un programa utilizando palabras de un idioma	Intérprete.	( )
22.	Lenguaje que la computadora puede entender en el momento de ejecutar un programa.	Linux.	( )
23.	Salió a principios de la década de los 80s, fue desarrollado por Microsoft y se hizo popular debido a la gran cantidad de *software* que se podía ejecutar con él.	Ejecutable	( )

24.	Sistema operativo multiusuario y multitarea desarrollado por los laboratorios Bell de AT&T que corre en diferentes computadoras: personales, minicomputadoras, supercomputadoras y redes de computadoras.	Lenguajes de quinta generación	( )
25.	Derivado del lenguaje máquina y está formado por abreviaturas de palabras y números llamados mnemotécnicos.	Procesador de texto.	( )
26.	Sistema operativo desarrollado con la finalidad de tener una interfaz gráfica más amigable con íconos que representan las diferentes operaciones que se pueden ejecutar.	Lenguaje de bajo nivel.	( )
27.	Programa que permite crear y modificar archivos digitales compuestos únicamente por texto plano sin formato.	Unix	( )
28.	Las instrucciones de este lenguaje están integradas por 0´s y 1's	Lenguajes de alto nivel.	( )
29.	Analiza cada línea del programa fuente que se va escribiendo y lo traduce directamente sin generar ningún código equivalente.	MS-DOS.	( )
30.	Propuesta tecnológica que es adoptada por una comunidad de programadores cuyo núcleo central es incuestionable en cuanto a que unívocamente trata de resolver uno o varios problemas claramente delimitados	Lenguaje ensamblador.	( )

2. En cada uno de los incisos siguientes colocar una "V" si es verdadero o bien una "F" si es falso que el sistema operativo lleve a cabo dicha tarea.

Actividad

1. Reconocer las señales de entrada de algún dispositivo (teclado, *mouse*, tacto, disco, USB, etc). ( )
2. Convertir un programa fuente en un programa objeto (de lenguaje C a lenguaje máquina por ejemplo). ( )
3. Administrar la información guardada en carpetas y archivos de la computadora. (copiar archivos, borrar archivos, crear nuevas carpetas, cambios de nombre de archivos, entre otras) ( )
4. Enviar señales de salida a diferentes elementos del sistema (monitor, disco, impresora, teléfono móvil, plóter, etc.). ( )
5. Controlar y mantener la comunicación con los equipos periféricos (teclado, monitor, impresora, disco duro, escáner, ratón, USB etc.). ( )
6. Graficar funciones matemáticas (seno, coseno, absoluto, raíz cuadrada, etc.) ( )
7. Controla la secuencia en que se deben llevar a cabo los procesos y tareas a realizar. ( )
8. Permite editar programas en un lenguaje de alto nivel ( )
9. Administra la memoria de la computadora. Esto permite que un *software* funcione en dos computadoras que tienen diferentes memorias y capacidad de almacenamiento. ( )
10. Manipula bases de datos (realiza búsquedas, filtra, une, agrega y elimina información en una base de datos). ( )

3. Para cada *software* de la siguiente tabla colocar un asterisco si tiene la característica de ser un: sistema operativo, lenguaje, intérprete, hoja de cálculo, procesador de texto, multitarea, monousuario, multiusuario, etc. Puede cumplir con una o más características.

Software	Sistema operativo	Lenguaje	Intérprete	Compilador	Hoja de cálculo	Procesador de texto	Monousuario	Multiusuario	Monotarea	Multitarea	*Software* libre	De alto nivel	De bajo nivel	Lenguaje máquina
MS-DS														
Pascal														
Basic														
Unix														
Excel														
JSP														
Linux														
Java														
Windows NT														
Ada														
Word														
PHP														
C++														
MacOS														
Binario														
Windows														
Ensamblador														
Word Perfect														

4. Contestar lo que se pide

1. Elaborar un cuadro sinóptico de la clasificación del *software* indicando por lo menos dos ejemplos de cada uno de ellos.

2. Escriba el algoritmo usando solamente texto para cada uno de los incisos siguientes.

   a) Algoritmo para leer dos números A y B sumarlos e imprimir su resultado.

   b) Algoritmo para leer dos números A y B e imprimir el mayor de ellos, en caso de que sean iguales imprimir cualquiera de ellos.

   c) Algoritmo para ingresar como alumno de nuevo ingreso al Tecnológico de Morelia

3. Tomando como elementos y/o conjuntos, elaborar un diagrama de Venn con ellos.

   - Lenguajes naturales.
   - Inglés.
   - Lenguajes.
   - Java.
   - Francés.
   - Lenguajes de programación.
   - Lenguajes formales.
   - Español.
   - C++

4. Lenguaje braille. Elaborar un mapa conceptual llamado "La computadora" en donde se muestre la relación que existe entre:

   a) CPU

   b) Dispositivos de entrada.

   c) Dispositivos de salida.

   d) *Software* de sistemas

   e) *Software* de aplicación

5. Indicar por medio de una figura geométrica cada uno de los incisos anteriores colocar el *hardware* o *software* citado a continuación, según corresponda dentro de esas figuras geométricas.

Impresora	Teclado	Ratón
Programas antivirus	CD	Manejadores de bases de datos
Pantalla táctil	Windows	USB
MacOS	TrackBall	Cañón.
Plóter	Bocinas	Hojas electrónicas
Escáner	Editores de texto	Unix
Linux	CAD	

5. Relacionar los tipos de paradigmas de programación con la característica que los distingue colocando en el paréntesis la letra que corresponda.

	Paradigma	Característica	
a)	Imperativo o Procedimental	Establece que todo problema puede ser resuelto únicamente con tres instrucciones (Secuencia, instrucción condicional y ciclo)	( )
b)	Lógico	Proporciona servicios de computación a través de Internet.	( )
c)	Estructurada	Está basado en el lema "Divide y vencerás".	( )
d)	Modular	Maneja conceptos como: clase, herencia, método, evento, mensaje, abstracción, encapsulamiento y polimorfismo.	( )
e)	Programación Orientada a Objetos (POO)	Indica a la computadora cada paso a realizar para resolver un problema determinado	( )
f)	En la nube	Está basado en la utilización de funciones matemáticas.	( )
g)	Funcional	Expresa formalmente problemas complejos y los resuelve mediante hipótesis, tautologías y reglas de inferencia.	( )

6. Colocar la década (70, 80, 90, 00) que corresponda a las características del *hardware* y/o *software* de dicho período de tiempo.

Década	Características del *hardware* y/o *software*
	Las computadoras usaban cintas y discos magnéticos de 5 ¼ pulgadas para guardar y leer información.
	Se usan con frecuencia para programar los lenguajes C++ y Java
	Las computadoras eran de grandes dimensiones y utilizaban tarjetas y/o cintas perforadas para leer información.
	Surgen las computadoras portátiles "*laptop*" y las USB como dispositivos de almacenamiento secundario.
	Los lenguajes más utilizados en esta década fueron Basic y Fortran IV
	Se empezaron a utilizar los sistemas operativos Windows, Linux y Unix.
	El ipod, cámaras digitales y robots son muy comunes en esta década.
	Sistemas operativos multitarea y multiusuario son comunes.
	Los lenguajes más utilizados eran: Basic, Pascal y Cobol.
	MS-DS fue el sistema más utilizado en esta década.

ALGORITMOS

2

Las computadoras son buenas siguiendo instrucciones, no leyendo tu mente.

Donald Knuth

Competencia de la unidad

Analizar problemas y representar su solución mediante algoritmos

- Analizar y resolver problemas cotidianos usando representaciones algorítmicas.
- Representar algoritmos por medio de: Diagramas de flujo, N−S (Nassi−Shneiderman), Pseudocódigo y Descripción narrada.
- Representar expresiones matemáticas usando los operadores aritméticos de suma, resta, multiplicación, división y módulo.
- Resolver problemas con algoritmos usando funciones.

Algoritmo

Contenido

2.1. Introducción.
2.2. Resolución de problemas de programación.
2.2.1. Análisis del problema.
2.3. Representación de algoritmos:
2.3.1. Descripción narrada.
2.3.2. Diagrama de flujo.
2.3.3. Pseudocódigo.
2.3.4. Diagramas Nassi−Shneiderman (N−S).
2.4. Diseño de algoritmos de funciones.
2.4.1. Funciones en computación.
2.4.2 Funciones recursivas.
2.4.3. Procedimientos.
2.5. Resumen.
2.6. Problemas.

2.1 Introducción

Un algoritmo es el conjunto de pasos ordenados en forma lógica que se ejecutan para llevar a cabo una actividad o resolver un problema. Todos los días se usan algoritmos para realizar tareas cotidianas en donde los pasos del algoritmo se ejecutan sin reparar en ellas, porque son algoritmos aprendidos, integrados por actividades que se realizan a diario. Es posible tener algoritmos para resolver problemas complejos o problemas a los que nunca nos hemos enfrentado, en cuyo caso es necesario analizar el problema, determinar los elementos que se tienen y los resultados a los que se quiere llegar, pasando por los posibles estados no previstos en donde se deben tomar decisiones de acuerdo a la situación que se presente.

Es posible representar los algoritmos usando símbolos, figuras de animales o cosas, gráficas, lenguaje natural y lenguajes de programación. En esta unidad se explicará la forma en que se representan los algoritmos por medio de diagramas de flujo, diagramas N−S, descripción narrada y pseudocódigo, dejando para las unidades posteriores la representación en lenguaje de programación.

También se usarán los operadores aritméticos básicos de suma, resta, multiplicación, división y módulo para la representación y evaluación de expresiones matemáticas. Se resuelven problemas y la solución es representada con lenguajes algorítmicos, con la finalidad de desarrollar en el alumno la lógica de programación, olvidándose de la sintaxis de un lenguaje formal y concentrándose exclusivamente en los pasos que integran el algoritmo.

2.2 Resolución de problemas de programación

Con la computadora es posible resolver problemas grandes y complejos, en donde es necesario apoyarse en expertos del área, para entender perfectamente los aspectos técnicos y la manera en que se calculan los diferentes elementos del problema a resolver. Existen también problemas sencillos en donde es relativamente fácil comprender el problema y resolverlo.

Independientemente si el problema es complicado o sencillo, si se tiene la participación de expertos en el área para auxiliar al programador en la comprensión del problema o no, los pasos a realizar en la resolución de problemas de programación son:

a) Análisis del problema.
b) Diseño del algoritmo.
c) Codificación del algoritmo.
d) Compilación y ejecución.
e) Verificación y depuración.
f) Mantenimiento.
g) Documentación.

2.2.1 Análisis del problema

Partiendo de que el enunciado del problema es correcto y detallado, el análisis del problema consiste en determinar claramente: los datos de entrada, el proceso de cálculo de la información y los resultados de salida.

- ***Datos de entrada***. Es la información requerida para resolver el problema. En este caso se debe considerar el medio por el cuál se obtendrán los datos (teclado, disco, pantalla, etc.), las variables necesarias para leer la información, el tipo de datos (numérico, texto, caracter, booleano), el orden y formato de lectura de la información (si se manda un mensaje antes de leer el dato o no).

- ***Proceso***. Aquí se debe considerar principalmente la mejor manera en que se puede calcular la información de salida, de dónde se obtiene la información que se necesita para llevar a cabo las operaciones, si se tienen condiciones a considerar en el proceso del cálculo, si existen fórmulas matemáticas en las cuales apoyarse, el orden lógico del cálculo (qué se calcula primero y qué información se encuentra después).
- ***Resultados de salida***. Es la información que se obtendrá al resolver el problema. Aquí se debe considerar: la manera en que se presentará dicho resultado (texto, datos numéricos enteros o con una parte fraccionaria, ordenados alfabéticamente, de mayor a menor, etc.). Se deberá establecer claramente lo que se quiere y la forma en que se presentará.

Para analizar el problema, determinar la información de entrada, establecer el proceso de cálculo y obtener el resultado del problema, es conveniente imaginar y representar esa imagen por medio de una Interfaz.

Representación de la Interfaz. Permite entender con mayor claridad el problema a resolver, ya que muestra los mensajes que se verán en la pantalla, los datos que serán leídos y la información esperada de la computadora al ejecutar el programa. La interfaz puede ser una figura que representa la pantalla de la computadora con los mensajes de lectura de información, los datos que se le proporcionan y los resultados esperados de la computadora así como la forma en que deberá proporcionarse. En programación se tienen dos retos importantes cuando se resuelve un problema, uno de ellos es entender el problema mismo, tener claramente cuáles son los datos de entrada, el proceso de cálculo y cuál es la información requerida, así como la forma en que se presentará dicha información. El otro reto es el diseño del algoritmo para la obtención de los resultados esperados. Para cada uno de los problemas a resolver de este libro, se presenta una interfaz de salida con la finalidad de facilitar la comprensión del problema, es decir; la información que se verá en la pantalla al momento de la ejecución del algoritmo o programa, se encuentra en un rectángulo con las esquinas redondeadas.

Ejemplo 2.1. Escribir un programa para leer la **base** y **altura** de un rectángulo y calcular su **área**. Indicar y explicar cuáles son:

a) Los datos de entrada
b) El proceso.
c) Los resultados de salida.
d) Interfaz de salida.

Datos de entrada. Claramente se observa que los datos de entrada son *base* y *altura,* se trata de datos numéricos, son números reales porque tienen parte entera y fraccionaria, los datos los proporciona el usuario al momento de ejecutar el programa, el orden de lectura de los mismos es primero la *base* y después la *altura* (aunque en este caso no afecta si primero se lee la altura y después la base).

Proceso. Los datos de entrada se leen por medio del teclado al momento de correr el programa ya que no se indica una forma especial de lectura, se sabe que: *área* = *base* × *altura*.

Resultados de salida. En este caso particular es el resultado de multiplicar la base por la altura y se presentará con un dato numérico que tiene parte entera y decimal.

Interfaz de salida. Muestra la información que se verá en la pantalla al momento de ejecutar el programa para resolver un problema típico. La manera en que se representará a lo largo de todo el libro es como se muestra a continuación:

```
Base:   8.4
Altura: 5.23
Área =  43.932
```

En la interfaz se observa que los mensajes de entrada son: "Base:" y "Altura:", que los datos de entrada son: 8.4 y 5.23. La información de salida está conformada por el mensaje de salida "Área=" y el resultado 43.932. El proceso en este caso consiste en multiplicar la base por la altura del rectángulo para encontrar de esa manera el área. Observar cómo los mensajes para leer información terminan con dos puntos (:).

Es importante mencionar que todos los programas desarrollados en esta obra deberán funcionar para cualquier valor leído, en otras palabras; deberán ser una solución general, aunque en la interfaz muestre la solución solamente para algunos datos particulares.

La interfaz puede tener diferentes tipos de letra, colores, figuras geométricas, diferentes mensajes, etc. Pero en este caso se trata de una interfaz muy sencilla que permite imaginar lo que se verá en la pantalla en el momento de ejecutar el programa.

2.3 Representación del algoritmo

Como se mencionó en el capítulo anterior, un algoritmo es un "*Conjunto finito de pasos, precisos y ordenados para resolver un problema*". Existen varios métodos para la representación de algoritmos entre los cuales se pueden mencionar:

a) Descripción narrada.
b) Diagramas de flujo.
c) Pseudocódigo.
d) Diagramas N−S.

2.3.1 Descripción narrada

Este método de representación de algoritmos utiliza el lenguaje natural hablado o escrito para describir los pasos que se deben seguir al resolver el problema. Cada una de las instrucciones puede ser una frase u oración narrada en forma concreta de lo que se debe hacer. Por ejemplo, el algoritmo en descripción narrada para el *Ejemplo 2.1* es como se muestra a continuación:

```
/*Algoritmo para calcular el área de un rectángulo*/
   inicio
   leer la base
   leer la altura
   calcular el área multiplicando la base por la altura
   imprimir el área.
   fin
```

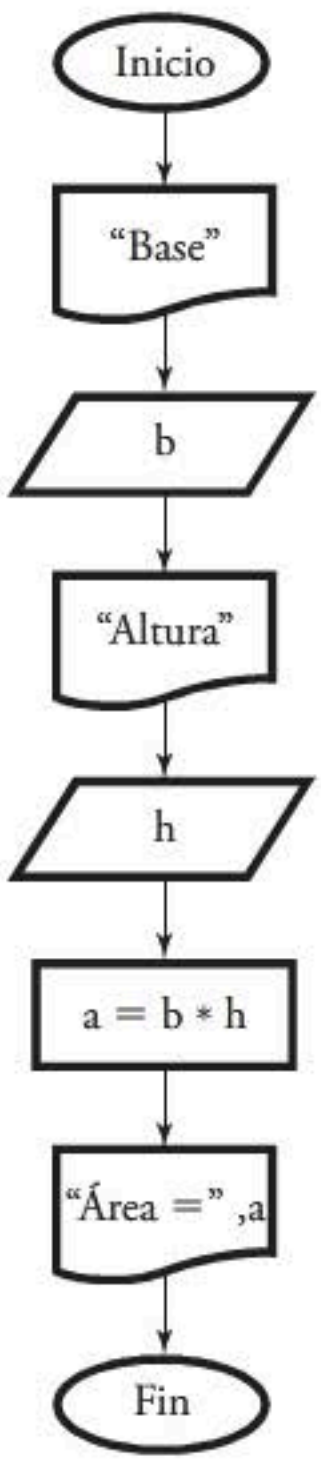
Los programas y también los algoritmos pueden tener comentarios acerca del algoritmo o de algunas de sus instrucciones que lo integran. Estos comentarios pueden ocupar una sola línea, una parte de línea (puede ir al lado de una instrucción) o varias líneas. En el caso anterior la información: /*Algoritmo para calcular el área de un rectángulo*/, es un comentario y se distingue porque está encerrado entre los símbolos /* y */.

2.3.2 Diagrama de flujo

Es una manera gráfica de representar los algoritmos, usando símbolos geométricos para cada uno de los pasos que integran el algoritmo. Por medio del diagrama de flujo se ilustra de manera gráfica la secuencia de operaciones que se realizan para resolver el problema. Un diagrama de flujo solamente tiene un principio y un final. Los símbolos más utilizados se muestran a continuación.

Símbolos básicos de los diagramas de flujo			
Símbolo	**Significado**	**Símbolo**	**Significado**
	Inicio o Fin de diagrama		Dirección de flujo
	Actividad		Entrada de datos
	Decisión		Conector para unir el flujo a otra parte del diagrama.
			Impresión o desplegado de información

La representación del algoritmo del *Ejemplo 2.1* usando diagrama de flujo se muestra a continuación:



Una salida típica si se corriera este programa es como se muestra en la interfaz siguiente:

```
Base:   8.4
Altura: 5.23
Área =  43.932
```

Observar que los mensajes se deben encerrar entre comillas y que es posible imprimir mensajes y el valor de variables en una misma línea, separándolos por coma.

Primeramente despliega el mensaje "Base:" en la pantalla y al llegar a la entrada de datos se detiene para que por medio del teclado el usuario proporcione el valor a la variable b. A continuación despliega el mensaje "Altura:" y espera el valor de la variable h, posteriormente multiplica el valor de b por el de h y se lo asigna a la variable a (a=b*h). Finalmente imprime el mensaje "Área=" seguido del valor de la variable a (la coma que separa al mensaje de la variable no se imprime ya que solamente es un separador, tampoco imprime las comillas que encierran a los mensajes).

Observar que es posible realizar operaciones aritméticas. A continuación se tiene una tabla con los signos aritméticos más comunes en los lenguajes de programación.

Tabla 2.1

Operador	Operación	Ejemplo	Significado de la operación
+	Suma	$c = a + b$	Suma el valor de *a* con *b* y asigna el resultado a *c*.
−	Resta	$c = a - b$	Al valor de *a* le resta el valor de *b* y el resultado de la operación se lo asigna a *c*
*	Multiplicación	$c = a * b$	Multiplica *a* por *b* y asigna el resultado a *c*.
/	División	$c = a/b$	Divide el valor de la variable *a* entre el valor de *b* y asigna el resultado a la variable *c*.
^	Potenciación	$c = a \wedge b$	Eleva a la potencia *b* el valor de *a* (a^b) y se asigna el resultado a *c*. Ejemplo si *a*=5 y *b*=3 entonces $c = 5 \wedge 3 = 5^3 = 125$.
mod	Módulo o residuo	$c = a \; mod \; b$	El valor de *c* es el residuo de dividir *a* entre *b*. Ejemplos: $c = 5 \; mod \; 3 = 2$; $c = 13 \; mod \; 8 = 5$

En los distintos lenguajes de programación se deben definir las variables como enteros, reales, caracter, cadenas, booleanas, etc. El resultado de una operación dependerá de los tipos de datos de las variables. Por ejemplo si se definen a las variables: *a*, *b* y *c* como enteras, donde $a = 7$ y $b = 3$, entonces $c = a/b = 2$, pero si *c* es real $c = a/b = 2.5$. También el símbolo para indicar las operaciones matemáticas puede cambiar de un lenguaje a otro (por ejemplo, la palabra "*mod*" o el símbolo para indicar la potenciación no se representan igual en C++, que en Java). En esta unidad utilizaremos los símbolos aritméticos indicados en la tabla anterior, pero una vez que se aborde el lenguaje de programación se harán las indicaciones pertinentes.

Representación de expresiones matemáticas usando lenguaje algorítmico

La computadora no puede leer una ecuación matemática como se representa normalmente en el área de matemáticas, sino que debe sufrir una transformación a un lenguaje algorítmico cercano al lenguaje de programación a usarse en la codificación del programa. La representación de expresiones matemáticas se lleva a cabo usando los operadores aritméticos indicados en la tabla anterior y con el auxilio de paréntesis para agrupar la información en forma lineal.

Ejemplo 2.2. En la siguiente tabla se representan con notación algorítmica algunas expresiones matemáticas, usando los símbolos aritméticos y paréntesis para agrupar la información cuando es requerido. Considerar que cada letra es una variable diferente.

Expresión matemática	Representación algorítmica
$y = ab - \frac{c}{d+e}$	$y = a * b - c / (d + e);$
$y = 5x^3 + \frac{\frac{ax-b}{8+x}}{x^2-1}$	$y = 5 * x\hat{}3 + ((a * x - b) / (8 + x)) / (x\hat{}2 - 1);$
$y = \frac{(bc-d)^3}{7+4a} + d - \frac{1}{3+b}$	$y = (b * c - d)\hat{}3 / (7 + 4 * a) + d - 1 / (3 + b);$
$y = (a+b) \bmod 3 + \frac{a^2}{7}$	$y = (a + b)\ mod\ 3 + a\hat{}2 / 7;$

Observar que se utilizan paréntesis para agrupar información cuando es necesario. Por ejemplo; si la primera expresión se hubiera representado sin paréntesis sería diferente a la ecuación deseada.

$$y = a * b - c / d + e; \quad \text{Equivale a} \quad y = ab - \frac{c}{d} + e$$

Se deben usar paréntesis solamente cuando sea necesario y no abusar de ellos, ya que eso puede causar la perdida de claridad o errores de la expresión que se está representando. Las siguientes tres expresiones son equivalentes:

$$y = a * b - c / (d + e); \qquad y = (a * (b)) - (c / (c / (d + e)); \qquad y = ((a * b) - (c / (d + e)));$$

Pero es obvio que es más clara y más simple la primera de ellas, aunque la computadora acepta a las tres como válidas y se obtiene el mismo resultado al sustituir el valor de las variables.

Jerarquía de operación

Las expresiones matemáticas no necesariamente se evalúan de corrido de izquierda a derecha, sino que los operadores aritméticos tienen diferente prioridad de evaluación y dependiendo del operador aritmético, la colocación de los mismos y la forma en que esté agrupada la información, marcará el orden de operación y por lo tanto el resultado obtenido. La tabla siguiente muestra la jerarquía de operación de los operadores matemáticos vistos hasta ahora.

Tabla 2.2	
Operador	**Jerarquía de operación**
( )	1a.
^	2a.
*, /, *mod*	3a.
+, −	4a.

Lo que implica que primeramente se evalúa la información que está encerrada entre paréntesis, posteriormente la potenciación, después tienen la misma jerarquía de operación la multiplicación, división y módulo para finalmente evaluar la suma y la resta.

Cuando existen operadores que tienen la misma jerarquía de operación como son: *, / y *mod* la operación se realiza de izquierda a derecha.

Ejemplo 2.3. La evaluación de la expresión: y = a + b/c*d − 3*(c + b*d). Para a = 2, b = 12, c = 3, d = 5. Realizando una operación a la vez, es como se muestra a continuación:

$y = a + b/c*d - 3*(c + b*d);$	
$y = 2 + 12/3*5 - 3*(3 + 12*5);$	Sustituyendo valores
$y = 2 + 12/3*5 - 3*(3 + 60);$	Evaluando la multiplicación del paréntesis
$y = 2 + 12/3*5 - 3*63;$	Evaluando la suma del paréntesis
$y = 2 + 4*5-3*63;$	Haciendo la división
$y = 2 + 20 - 3*63;$	Realizando la multiplicación de la izquierda
$y = 2 + 20 - 189;$	Haciendo la multiplicación
$y = 22 - 189;$	Realizando la suma
$y = -167;$	Finalmente haciendo la resta

Es importante mencionar que cuando hay varios paréntesis, primero se evalúa el que se encuentra más a la izquierda, si existe un paréntesis dentro de otro se evalúa primero el que se encuentra más adentro respetando siempre la jerarquía de operación.

Ejemplo 2.4. Para *a = 10, b = 5, c = 3, d = 2*, evaluar la expresión matemática representada en notación algorítmica de cada uno de los incisos:

a) $y = a/d - (7 - a/b*2)*c - 8 + (a + 3*(b - c\wedge 2))*d;$
b) $y = (a + b/(c - a*d + (4 - c)/b))*2 + a*c;$

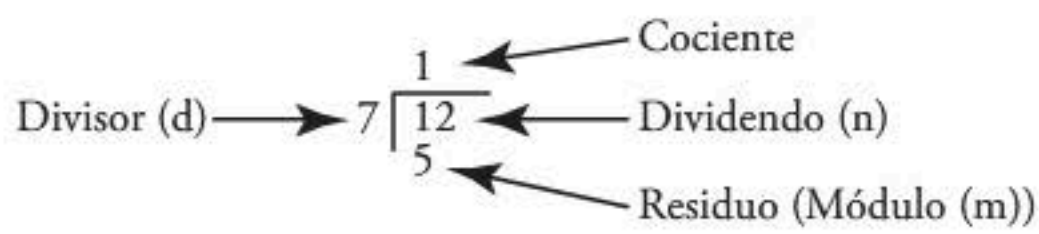
Respuesta al inciso (a)

$y = a/d - (7 - a/b*2)*c - 8 + (a + 3*(b - c\wedge 2))*d;$	
$y = 10/2 - (7 - 10/5*2)*3 - 8 + (10 + 3*(5 - 3\wedge 2))*2;$	Sustituyendo valores.
$y = 10/2 - (7 - 2*2)*3-8 + (10+3*(5 - 3\wedge 2))*2;$	Realizando la división del primer paréntesis.
$y = 10/2 - (7 - 4)*3-8 + (10+3*(5 - 3\wedge 2))*2;$	Haciendo la multiplicación del primer paréntesis
$y = 10/2 - 3*3 - 8 + (10+3*(5 - 3\wedge 2))*2;$	Realizando la resta del primer paréntesis.
$y = 10/2 - 3*3 - 8 + (10+3*(5 - 9))*2;$	Haciendo la exponenciación del paréntesis que está dentro.
$y = 10/2 - 3*3 - 8 + (10+3*(- 4))*2;$	Realizando la resta del paréntesis que está dentro.
$y = 10/2 - 3*3 - 8 + (10 - 12)*2;$	Realizando la multiplicación en el paréntesis.
$y = 10/2 - 3*3 - 8 + (- 2)*2;$	Realizando la resta. Se dejó (−2) para dar mayor claridad.
$y = 5 - 3*3 - 8 + (- 2)*2;$	Realizando la división.
$y = 5 - 9 - 8 + (- 2)*2;$	Realizando la primera multiplicación.
$y = 5 - 9 - 8 - 4;$	Realizando la multiplicación.
$y = - 4 - 8 - 4;$	Realizando la primera resta.
$y = - 12 - 4;$	Realizando resta
$y = - 16;$	Finalmente haciendo la última resta

Respuesta al inciso (b). Para $a = 10$, $b = 5$, $c = 3$, $d = 2$

$y = (a + b/(c - a*d + (4 - c)/b))*2 + a*c;$	
$y = (10 + 5/(3 - 10*2 + (4 - 3)/5))*2 + 10*3;$	Sustituyendo valores
$y = (10 + 5/(3 - 10*2 + 1/5))*2 + 10*3;$	Realizando resta.
$y = (10 + 5/(3 - 20 + 1/5))*2 + 10*3;$	Realizando la multiplicación
$y = (10 + 5/(3 - 20 + 0.2))*2 + 10*3;$	Realizando la división en el paréntesis interior.
$y = (10 + 5/(- 17 + 0.2))*2 + 10*3;$	Resta dentro del paréntesis
$y = (10 + 5/(- 16.8))*2 + 10*3;$	Haciendo restas en paréntesis de dentro.
$y = (10 - 0.2976)*2 + 10*3;$	Haciendo división.
$y = (9.7024)*2 + 10*3;$	Haciendo resta.
$y = 19.4048 + 10*3;$	Multiplicación de la izquierda.
$y = 19.4048 + 30;$	Última multiplicación
$y = 49.4048$	Haciendo finalmente la suma.

Operador módulo. Cuando se lleva a cabo una división se tiene un cociente y un residuo. El módulo es el residuo de esa división considerando un cociente entero.



Algunas hojas de cálculo utilizan la siguiente expresión matemática para calcular el módulo.

$$m = n \; mod \; d = n - d * \text{EnteroMenor}\,(n / d)$$

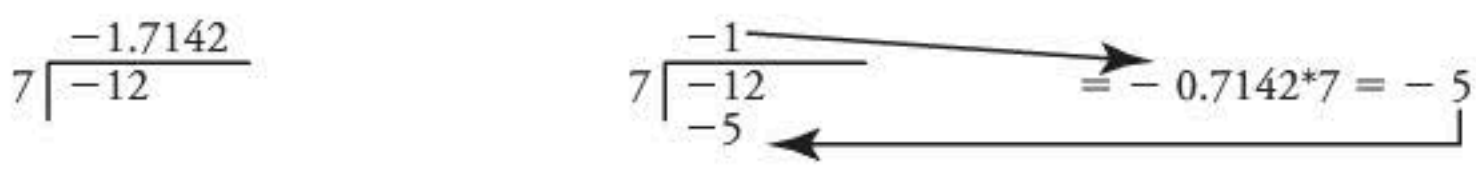
Con los datos anteriores:

$$m = 12 \; mod \; 7 = 12 - 7 * \text{EnteroMenor}\,(12 / 7) = 12 - 7*(1.7142) = 12 - 7*(1) = 5$$

Sin embargo hay diferencia en el resultado obtenido cuando las cantidades involucradas son de signo contrario. Ejemplo:

$$m = -12 \; mod \; 7 = -12 - 7 * \text{EnteroMenor}\,(-12/7) = -12 - 7*(-1.7142) = -12 - 7*(-2) = 2$$

En matemáticas si dos cantidades se dividen y esas cantidades son de signo contrario, el cociente es negativo. Usando la calculadora el residuo se puede encontrar multiplicando la parte fraccionaria del cociente por el divisor y redondeándolo al entero más cercano, como se muestra a continuación:



Algunos lenguajes de programación, usan la siguiente expresión para encontrar el residuo, que es el resultado esperado.

$$m = n \; mod \; d = n - d * \text{Cociente Entero}\,(n / d)$$

Con los datos anteriores se tiene:

$$m = -12 \; mod \; 7 = -12 - 7 * \text{Cociente Entero}\,(-12 / 7) = -12 - 7*(-1) = -5$$

En forma generalizada el módulo siempre tendrá el mismo signo del dividendo. Ejemplos.

$$m = 12 \; mod -7 = 12 - (-7) * \text{Cociente Entero } (12 / (-7)) = 12 + 7*(-1) = 5$$
$$m = -12 \; mod -7 = -12 - (-7) * \text{Cociente Entero } (-12 / (-7)) = -12 + 7*(1) = -5$$

En computación, si dos cantidades enteras se dividen y el resultado de la división se le asigna a otra variable entera, el resultado es el entero del cociente. Ejemplo: considerar que w es una variable entera.

$$w = 9 / 4 = 2 \qquad w = -7/2 = -3 \qquad w = 24 / 5 = 4$$

Ejemplo 2.5. Sean *a, b, c, d* y x variables enteras cuyos valores son $a = 3$, $b = -5$, $c = 2$, $d = 4$. Encontrar el resultado de las expresiones matemáticas expresadas en lenguaje algorítmico de cada uno de los incisos.

a) $x = (a / b + d) \; mod \; c$;
b) $x = b + a \; mod \; (b / c) - d \wedge (c \; mod \; a)$;
c) $x = (((b \; mod \; a) \; mod \; c)*d) / (b-c)$;

Respuesta:

a) $x = (a / b + d) \; mod \; c$;
$x = (3 /(-5) + 4) \; mod \; 2$; Sustituyendo valores:
$x = (0 + 4) \; mod \; 2$;
$x = 4 \; mod \; 2$;
$x = 0$

b) $x = b + a \; mod \; (b / c) - d \wedge (c \; mod \; a)$;
$x = -5 + 3 \; mod \; (-5 /2) - 4 \wedge (2 \; mod \; 3)$; Sustituyendo valores:
$x = -5 + 3 \; mod \; (-2) - 4 \wedge (2 \; mod \; 3)$;
$x = -5 + 3 \; mod \; (-2) - 4 \wedge (2)$;
$x = -5 + 3 \; mod \; (-2) - 16$;
$x = -5 + 1 - 16$;
$x = -20$

c) $x = (((b \; mod \; a) \; mod \; c)*d) / (b - c)$;
$x = (((-5 \; mod \; 3) \; mod \; 2)*4) / (-5 - 2)$; Sustituyendo valores:
$x = ((-2 \; mod \; 2)*4) / (-5 - 2)$;
$x = (0*4) / (-5 - 2)$;
$x = 0 / (-5 - 2)$;
$x = 0/ -7$
$x = 0$

Además de los operadores matemáticos es posible usar funciones matemáticas, material que se abordará en la siguiente unidad cuando se estudie el lenguaje a usar para la codificación de algoritmos.

Operadores relacionales. Se usan para comparar el valor de dos variables, el resultado de la comparación puede ser falso o verdadero. Por lo general este tipo de operadores se utilizan en las proposiciones de condiciones o para controlar las veces que se ejecutan las instrucciones que se encuentran en ciclos. El símbolo del operador no es el mismo en los diferentes lenguajes de programación, en su momento se indicará cuál es el símbolo en el lenguaje a utilizar para la codificación de los programas, pero los que se usarán en esta unidad se muestran en la siguiente tabla:

Operador	Significado	Ejemplo
>	Mayor que	x > y
<	Menor que	x < y
> =	Mayor o igual que	x >= y
< =	Menor o igual que	x <= y
= =	Igual a	x = = y
! =	Diferente de	x != y

Operadores lógicos. Se utilizan en instrucciones condicionales o en ciclos en donde se requiere determinar si son falsas o verdaderas más de una proposición. Los operadores **and** y **or** son binarios, lo cual significa que se utilizan para relacionar dos proposiciones y el operador **not** es unario y se utiliza para negar o complementar una proposición. La siguiente tabla muestra los operadores lógicos básicos.

Operador	Significado	Ejemplo
and	y	(x > y) and (m == n)
or	o	(x > y) or (m == n)
not	no	not (m == n)

Una proposición puede estar integrada por operadores aritméticos, operadores relacionales y operadores lógicos, por tal razón es conveniente tener en cuenta la jerarquía de operación en forma global.

Operador	Jerarquía de operación
( )	1a.
^	2a.
*, /, mod	3a.
+, −	4a.
>, >=, <, <=	5a.
= =, !=	6a.
not	7a.
and	8a.
or	9a.

Observar que la jerarquía de operación de los operadores relacionales es mayor que la de los operadores lógicos y que en la evaluación de la proposición si se presentan operaciones aritméticas, éstas se evalúan antes, ya que tienen mayor jerarquía que cualquiera de los operadores relacionales o lógicos

Ejemplo 2.6. Si m = −3, n = 2, w = 5 y x = −1. Determinar cuál es valor booleano resultante, considerando que: 1= verdadero y 0 = falso, para cada uno de los siguientes incisos:

a) **not**((m >= n) **or** (x = = w))
b) (x < m) **and not**(m > w **or** n = = x)
c) (n != x) **or** (**not**(m > w) **and** w <= x)
d) x != m **and not** (w >= n **or** m < n) **or** x = = w

Respuestas:

a) not((m >= n) or (x = = w))
not((−3>=2) or (−1 = = 5)) Sustituyendo valores:
not(0 or 0)
not(0)
1 La proposición es verdadera para esos valores de m, n, w, x

b) (x < m) and not(m > w or n = =x)
(−1 < −3) and not(−3 > 5 or 2 = =−1)
0 and not(0 or 0)
0 and not(0)
0 and 1
0 La proposición es falsa

c) (n != x) or (not(m > w) and w <= x)
(2 != −1) or (not(−3 > 5) and 5 <= −1)
1 or (not(0) and 0)
1 or (1 and 0)
1 or 0
1 Es verdadera la proposición

d) x != m and not (w >= n or m < n) or x = = w
−1 != −3 and not (5 >= 2 or −3 < 2) or −1 = = 5
1 and not (1 or 1) or 0
1 and not (1) or 0
1 and 0 or 0
0 or 0
0 La proposición es falsa

Ejemplo 2.7. Si a = 1, b = 3, c = −2 y d = −1. Determinar cuál es valor booleano: 1= verdadero 0 = falso, para cada uno de los siguientes incisos e indicar cuál es el mensaje impreso.

a) *si* (a <=d*c **or not**(c != b) **and** 2*b >= d)
imprimir "Rosa:", 7*c;
sino
imprimir 4*d," Lila"

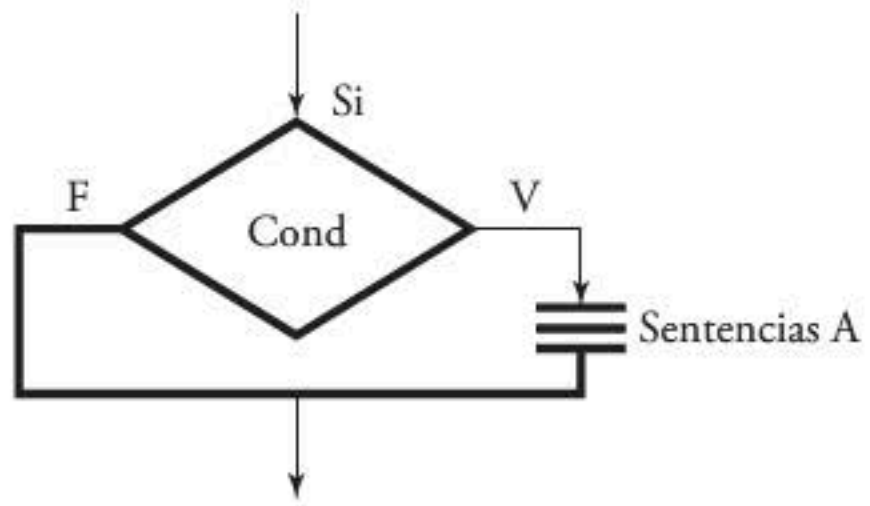
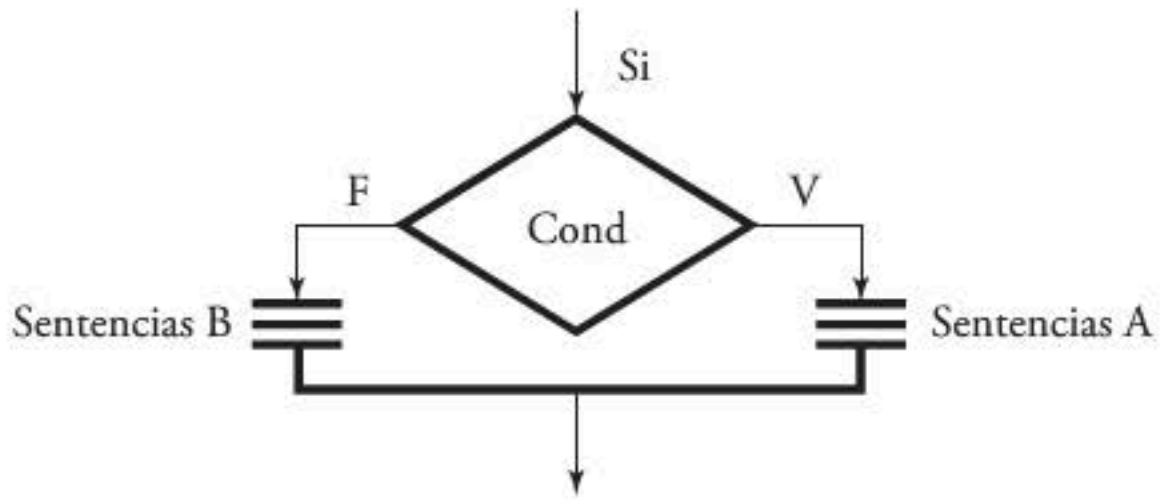
b) *si* (**not**((**not**(b < a) **or** d = = b*a) **and** a>=(d−3*a) **or not**(b != d)))
imprimir "Melón: " , 3*c;
sino
imprimir "Mango:", (5*a−d);

Evaluación de la proposición (a)
(a <=d*c or not(c != b) and 2*b>=d)
(1 <=(−1)*(−2) or not(−2 != 3) and 2*3>= −1) Sustituyendo valores
(1 <= 2 or not(−2 != 3) and 6>= −1) Primero operaciones aritméticas
(1 or not(1) and 1) Después operadores relacionales
(1 or 0 and 1) El not tiene jerarquía sobre los demás operadores lógicos

```
(1 or 0)                    El and tiene más jerarquía que el or
1                           Como la proposición es verdadera
El mensaje impreso es: Rosa:−14

Evaluación de la proposición (b)
(not((not(b < a) or d = = b*a) and a>=(d−3*a) or not(b != d)))
( not((not(3 < 1) or −1 = = 3*1) and 1>=(−1−3*1) or not(3 != −1)))
(not((not(3 < 1) or −1 = = 3) and 1>= −4 or not(3 != −1)))
(not((not(0) or 0) and 1 or not(1)))
(not((1 or 0) and 1 or 0))
( not(1 and 1 or 0))
( not(1 or 0))
(not(1))
 0
 Mensaje: Mango: 6
```

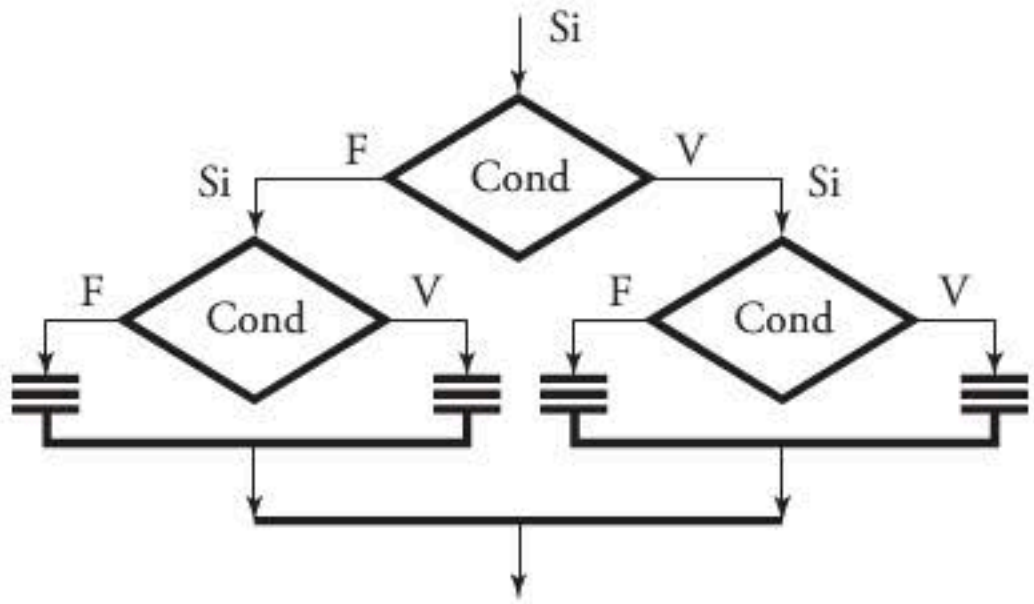
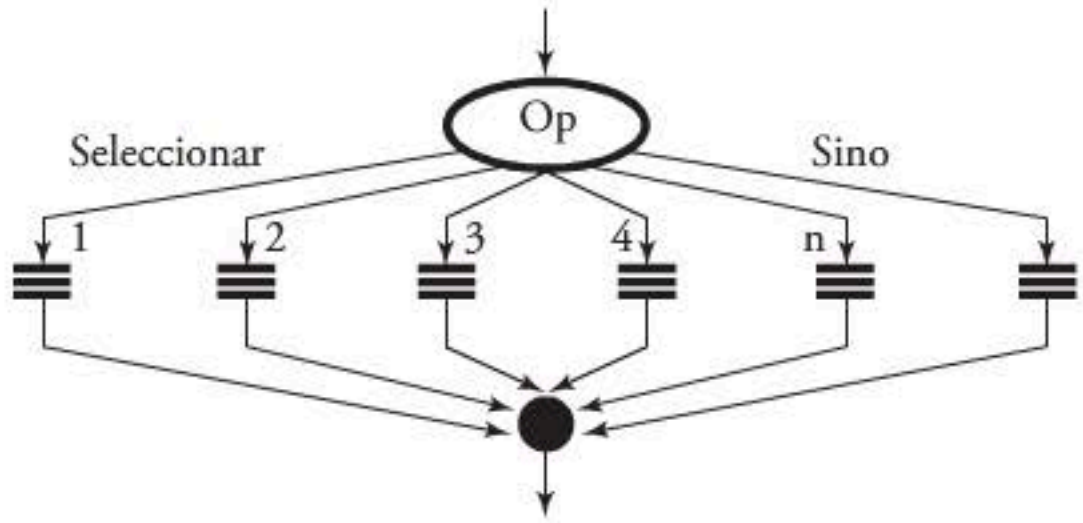
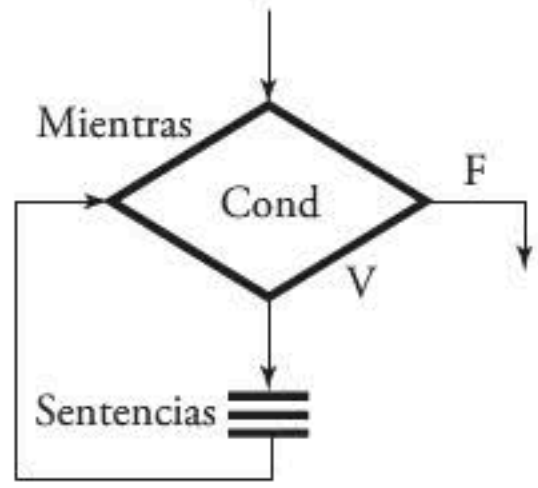
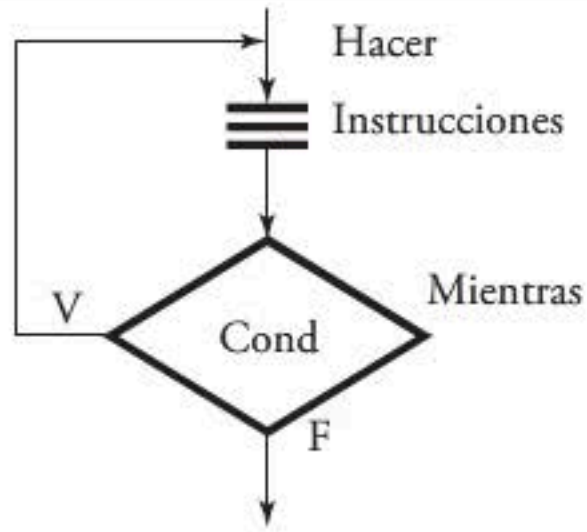
Observar que la jerarquía de operación señalada anteriormente se respeta. Considerar también que la información entre paréntesis se evalúa primero que la que está fuera, y si el paréntesis es interno; se debe evaluar primero que los paréntesis externos.

Los algoritmos también pueden representarse usando condicionales, ciclos, procedimientos y funciones. Existe representación grafica usada en los diagramas de flujo para indicar cada una de estas instrucciones compuestas como se muestra a continuación.

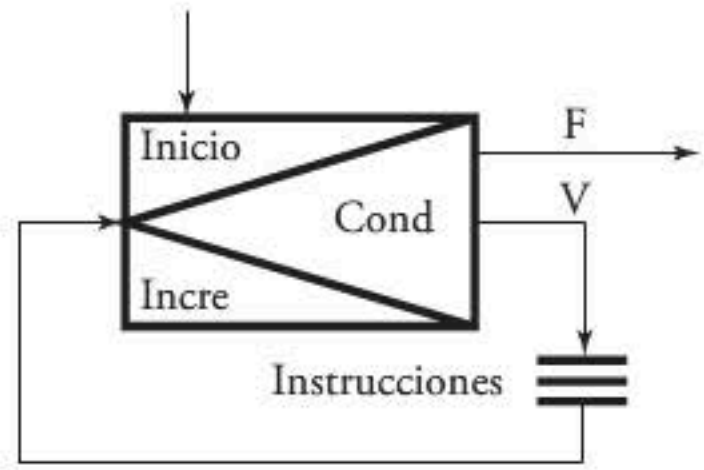
Instrucción	Representación gráfica
Condicional simple. Si la condición se cumple se ejecuta el conjunto de Sentencias A y sino se cumple no se ejecuta ninguna.	
Condicional doble. Si la condición se cumple se ejecuta el conjunto de Sentencias A y sino se cumple se ejecutan las sentencias B	

(*continúa*)

(*continuación*)

Instrucción	Representación gráfica
Condicional anidados. Si se cumple la condición se ejecuta otro condicional y si no se cumple también puede ejecutarse (o no) otra condicional diferente. Puede haber varias instrucciones condicionales anidadas.	
Condicional múltiple. Dependiendo del valor de la variable (Op) se ejecutará el bloque de instrucciones de la ruta 1, 2,…ó n. Cuando el valor de Op no está dentro de las opciones contempladas, podría no ejecutarse ninguna instrucción o bien otro bloque de instrucciones alternas.	
Mientras. Es un ciclo que ejecuta una serie de sentencias (o instrucciones) mientras se cumpla la condición indicada en el rombo, una vez que deje de cumplirse la condición, se sale del ciclo.	
Hacer—Mientras. Ejecuta una serie de instrucciones desde donde se encuentra la palabra "Hacer" mientras se cumpla la condición planteada dentro del rombo. Una vez que se deje de cumplir dicha condición abandona el ciclo.	

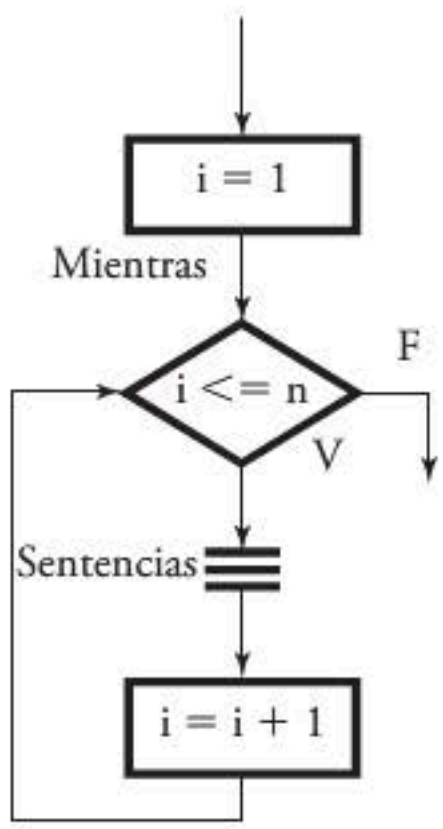
(*continuación*)

Instrucción	Representación gráfica
Desde/Para. Este ciclo ejecuta una serie de instrucciones. El número de veces que se ejecutan se controla por medio de una variable que se inicializa con un valor "*Inicio*", en cada iteración la variable sufre un incremento "*Incre*" (o decremento). El bloque de instrucciones se ejecuta mientras la condición "*Cond*" sea verdadera.	

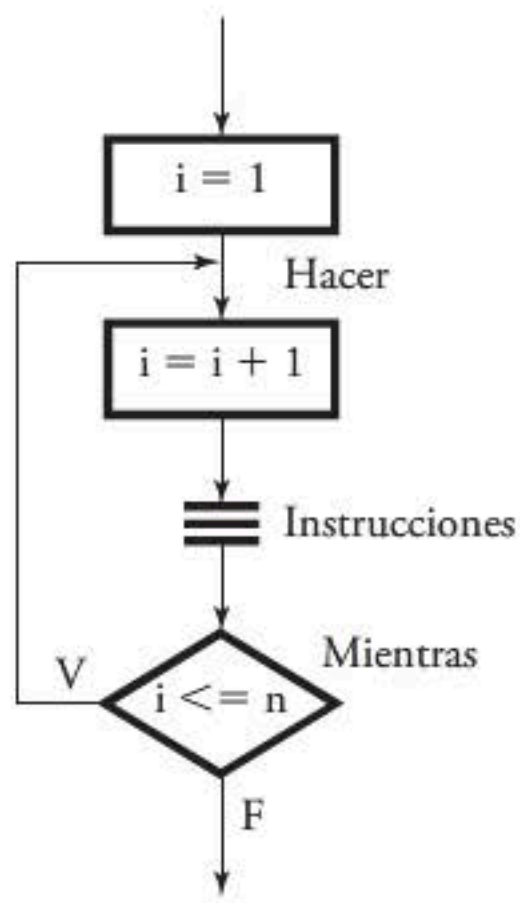
Todos los ciclos tienen tres elementos para controlar el número de veces que se ejecutan las instrucciones que se encuentran dentro de ellos. Estos elementos son:

- Un *inicio* (por ejemplo i =1, i = 0, i = x, etc.) que se coloca antes del ciclo para inicializar la variable controladora de las iteraciones
- Una *condición* (ejemplo i>n, i=<n, i!=n ó i<n) que cambia dependiendo del ciclo que se utilice y los requerimientos del mismo problema.
- Un *incremento,* decremento o actualización (ejemplo i=i+1, i=i+3, i=i−2, i=i+x, i=i mod 2, i=i*3, etc.) que en cada iteración incrementa (o actualiza) la variable i. Su colocación es dentro del ciclo con la finalidad de cambiar su valor en cada iteración.

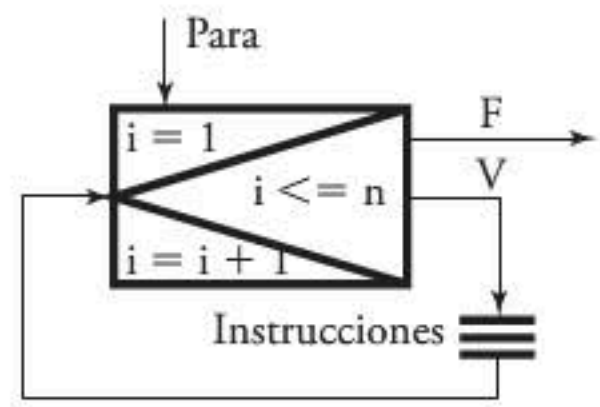
Los diagramas de flujo siguientes muestran estos tres elementos fundamentales de los ciclos:



(1)



(2)

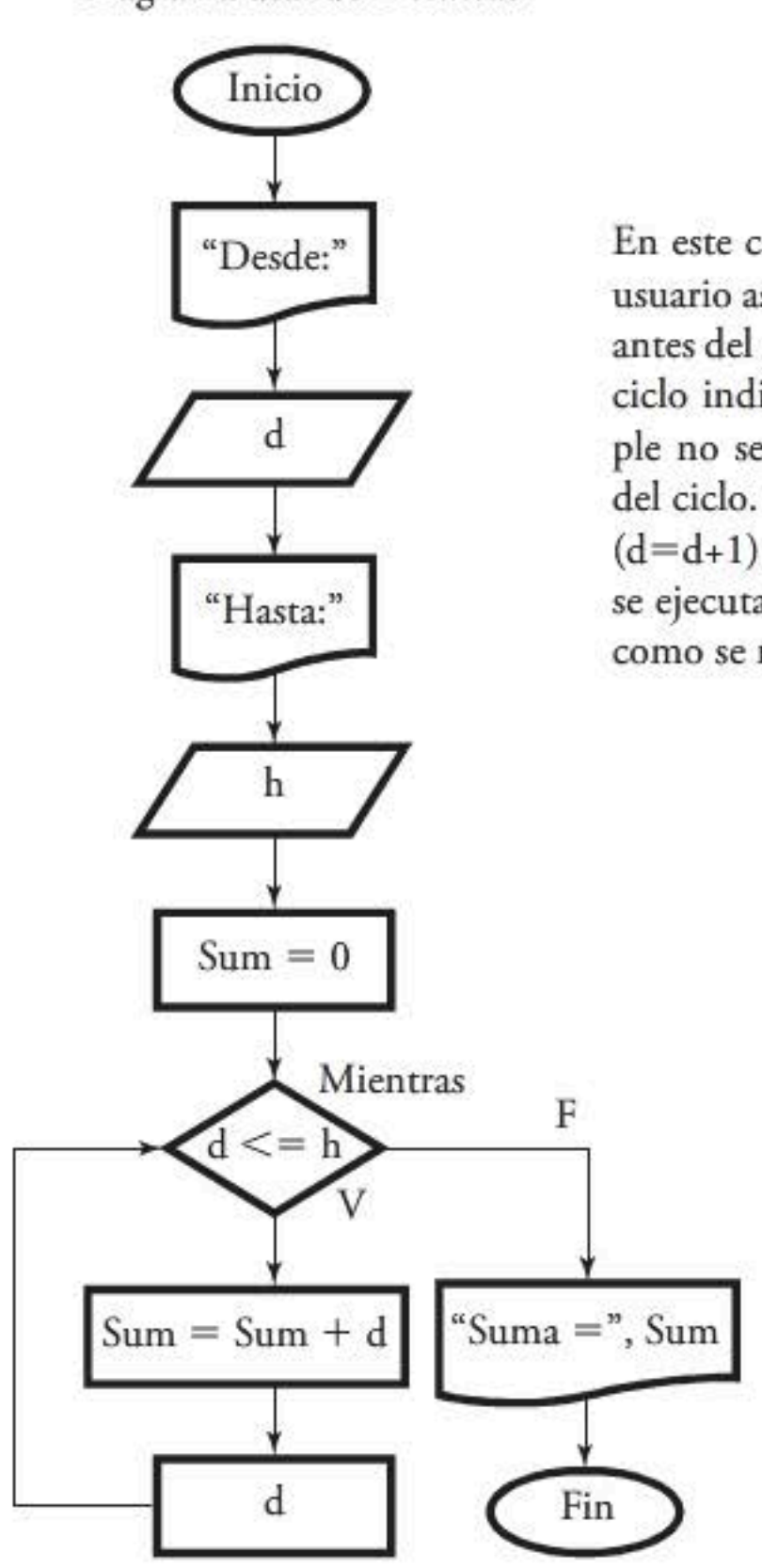


(3)

Los tres ciclos anteriores *Mientras*, *Hacer–Mientras* y *Desde* (o *Para*) están controlados por la variable i que inicialmente tiene un valor de uno (i = 1). La condición en el caso de *Mientras* y *Desde* está inmediatamente después de asignarle el valor inicial a la variable, de tal manera que si la condición (i<=n) es falsa no se ejecutarán ninguna vez las instrucciones o sentencias que se encuentran dentro del ciclo. En el caso de *Hacer–Mientras* las instrucciones que se encuentran dentro del ciclo, se llevan a cabo por lo menos una vez, debido a que la condición se encuentra al final del mismo. Los tres ciclos ejecutan n veces el bloque de instrucciones que están dentro de ellos y en los tres, la variable tiene un valor inicial que se incrementa en la unidad (i=i+1) en cada iteración.

Ejemplo 2.8. Elaborar los diagramas de flujo para sumar los números enteros entre d y h, usando para ello los ciclos (*Mientras*, *Hacer–Mientras* y *Desde*).

Diagrama usando *Mientras*

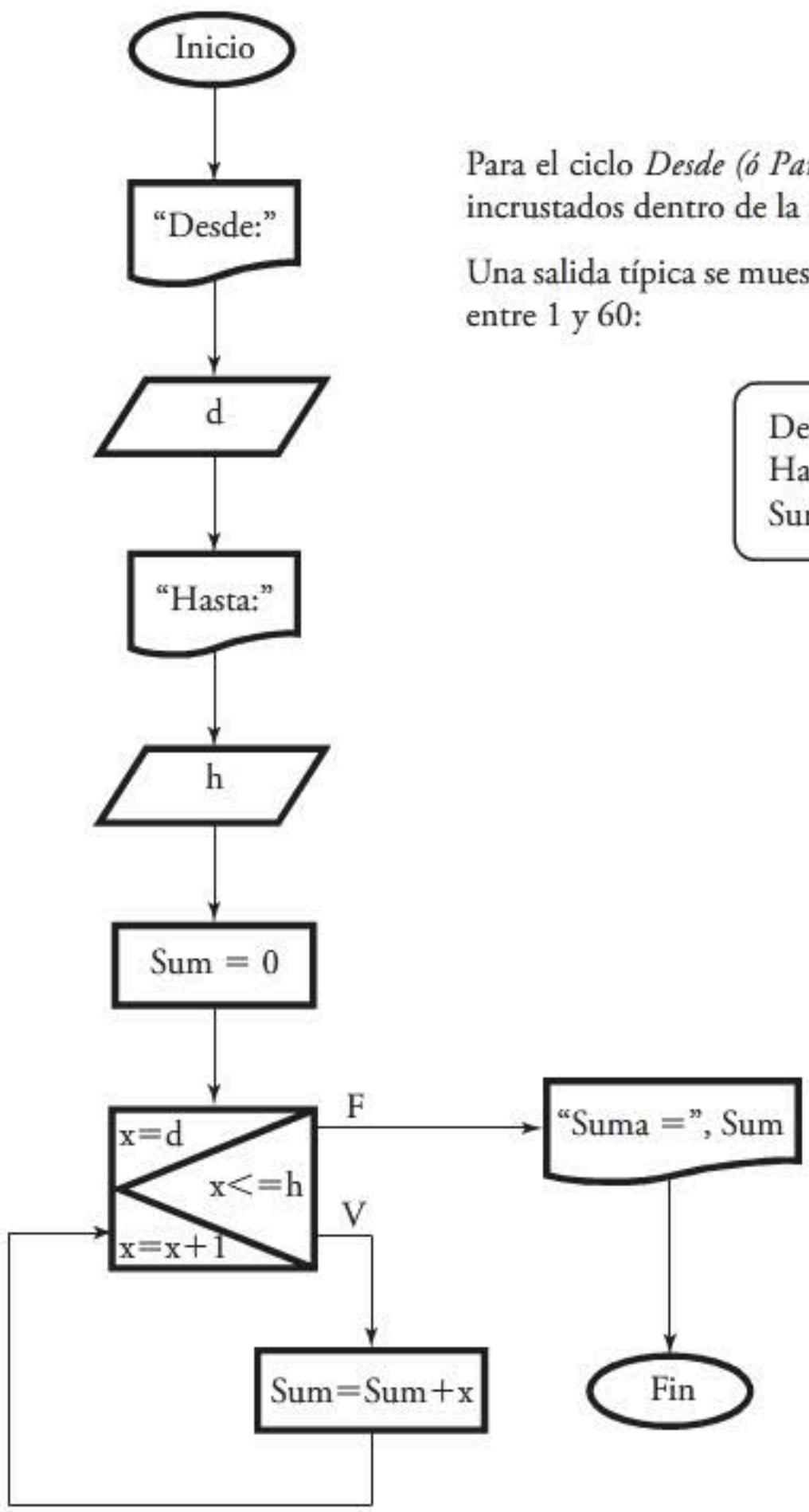


En este caso el inicio es el mismo valor de la variable d a la cual el usuario asigna el valor al momento de ejecutar el programa, se ubica antes del propio mientras. La condición está ubicada al principio del ciclo indicado por la figura del rombo (d<=h), si ésta no se cumple no se ejecutará ninguna de las instrucciones que están dentro del ciclo. El incremento de la variable controladora de iteraciones d (d=d+1) está explícitamente dentro del bloque de instrucciones que se ejecutan cierto número de veces. Una interfaz de salida típica es como se muestra a continuación:

```
Desde:  17
Hasta:  41
Suma = 725
```

Observa que para los valores d=12 y h=4 la computadora no ejecutará ninguna de las instrucciones que están dentro del ciclo, ya que no se cumple que (d<=h) de tal manera que la salida será: "Suma=0" debido a que no sumará ningún número.

Diagrama usando *Desde*



Para el ciclo *Desde (ó Para)* el inicio, condición e incremento están incrustados dentro de la misma figura (x=d; x<=h; x=x+1).

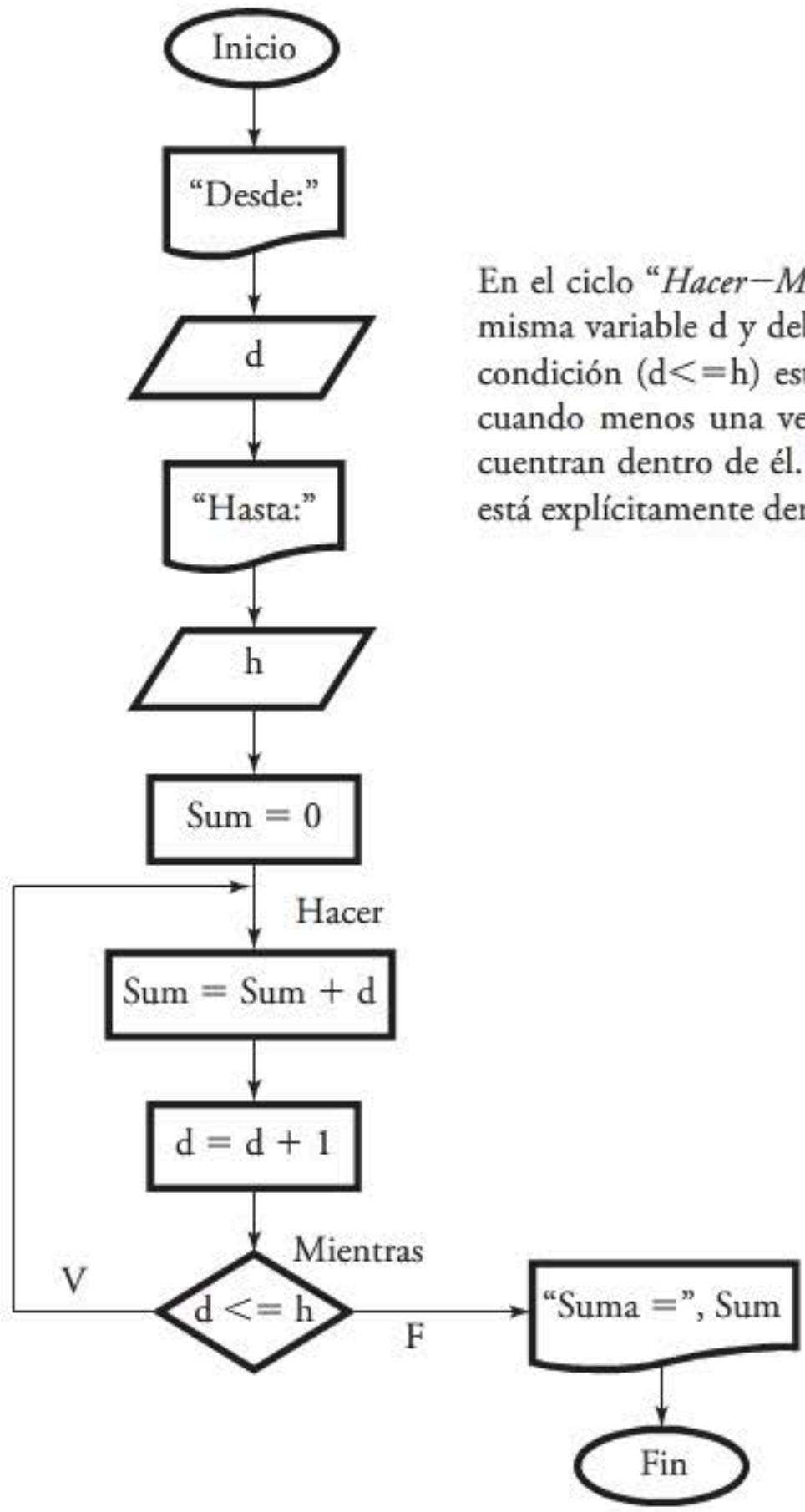
Una salida típica se muestra a continuación para sumar los números entre 1 y 60:

```
Desde:  1
Hasta:  60
Suma =  1830
```

En este caso la variable controladora de iteraciones es x a la cual se le asigna como valor inicial d (x=d), la computadora asigna este valor una única vez (cuando el flujo entra por primera vez al ciclo, ya que en las siguientes iteraciones que se ejecuten solamente incrementará el valor de x (x=x+1) e inmediatamente preguntará si se sigue cumpliendo la condición establecida (x<=h). En caso afirmativo seguirá iterando hasta que deje de cumplirse saldrá del ciclo e imprimirá el resultado de la suma.

Para los valores d=12 y h=4 tampoco ejecutará ninguna vez la instrucción Sum=Sum+x que está dentro del ciclo porque no se cumple la condición (x<=h) y por lo tanto la salida para estos valores es: "Suma = 0".

Diagrama usando *Hacer–Mientras*



En el ciclo "*Hacer–Mientras*" el inicio se considera en este caso la misma variable d y deberá conocerse dicho valor antes del ciclo, la condición (d<=h) está hasta el final del ciclo, de tal manera que cuando menos una vez se ejecutarán las instrucciones que se encuentran dentro de él. El incremento de la variable d (d = d + 1) está explícitamente dentro del ciclo.

Observar que la principal diferencia entre los ciclos *Mientras* y *Hacer–Mientras* es principalmente la colocación de la condición, en *Mientras* está al principio y en *Hacer–Mientras* está al final, por esta razón sufren un cambio en el planteamiento de dicha condición. Una salida típica para otros valores es como se muestra:

```
Desde:  −23
Hasta:  18
Suma =  −105
```

Para los valores d=12 y h=4, la salida es "Suma= 12" debido a que ejecuta una vez las instrucciones

Sum=Sum+d (Sum=0+12) y d=d+1 (d=12+1). Si se hubiera querido que no se ejecutaran ninguna vez las instrucciones que están dentro del ciclo, se tendría que determinar antes del ciclo si d es menor que h, en caso contrario invertir los valores, mandar el mensaje correspondiente (ejemplo "Intervalo no válido") o bien usar otro ciclo que no sea *Hacer–Mientras*.

En los tres casos anteriores; se tiene un contador de iteraciones controlado por la variable d, a la cual se le asigna un valor inicial, un incremento y una condición. Observar también que cuando se desea realizar una suma, la variable encargada de llevar a cabo dicha actividad se debe inicializar con cero (sum=0) antes del ciclo y se incrementará dentro del ciclo (sum=sum+d): En este caso se incrementa en cada paso porque así lo requiere el problema, pero algunas veces dicha variable se incrementará solamente si se cumple una condición (por ejemplo si desean sumar solamente los elementos divisibles entre 4, dentro del ciclo se requiere una instrucción condicional que permita incrementar la suma solamente cuando se cumpla la condición correspondiente).

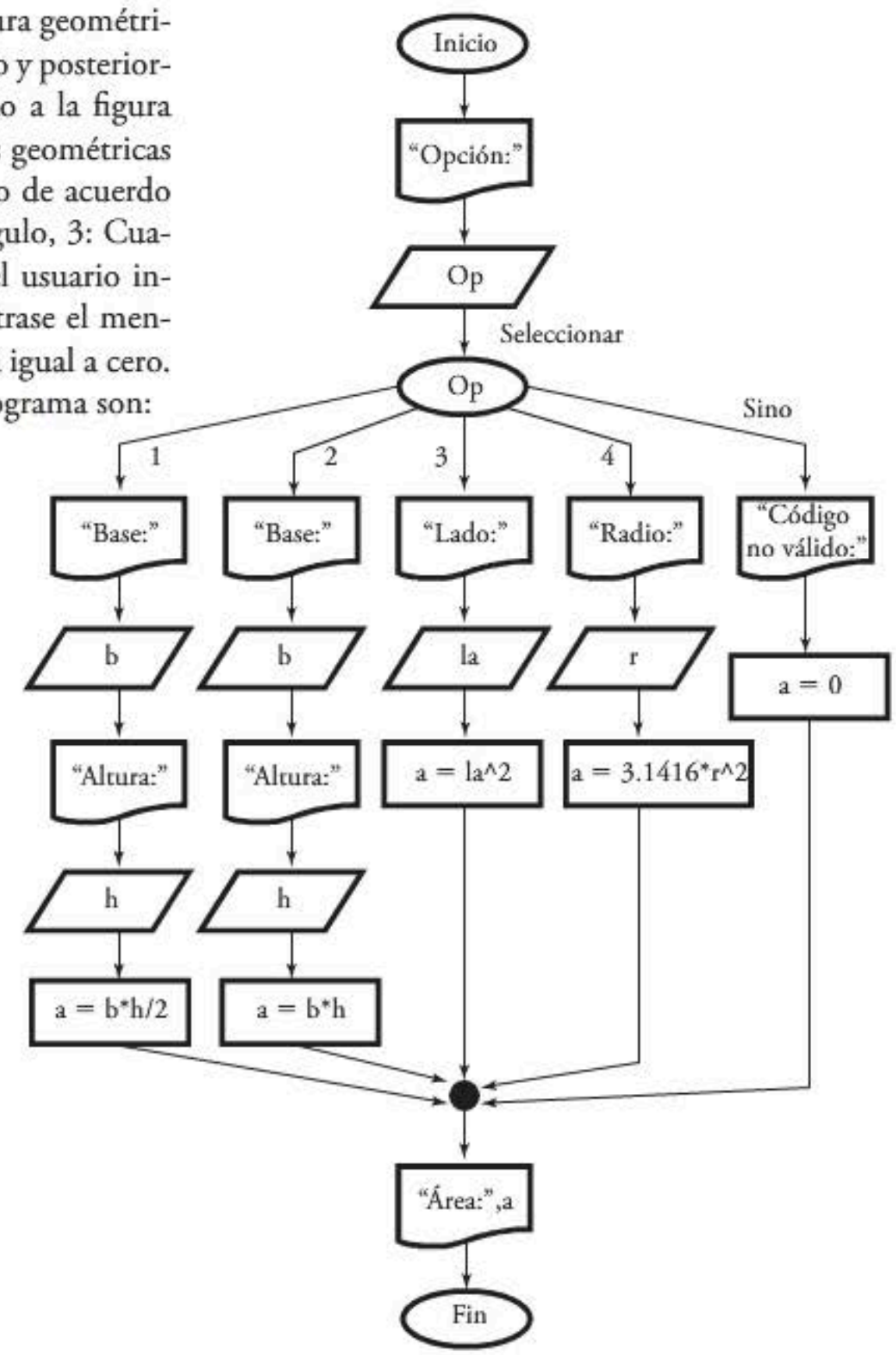
Ejemplo 2.9. Elaborar un diagrama de flujo para encontrar el área de las figuras geométricas básicas. Se desea leer primeramente la figura geométrica representada por un código numérico y posteriormente leer sus dimensiones de acuerdo a la figura que se trate. Considerar que las figuras geométricas se representan por un código numérico de acuerdo a lo siguiente: 1: Triangulo, 2: Rectángulo, 3: Cuadrado, 4: Círculo. Considerar que si el usuario ingresa un código inexistente, debe mostrase el mensaje "Código no válido" e imprima área igual a cero. Algunas salidas típicas al ejecutar el programa son:

Figura: 2
Base: 13
Altura: 9
Área = 117

Figura: 4
Radio: 7
Área = 153.93

Figura: 3
Lado: 5
Área = 25

Figura: 80
Código inexistente
Área = 0



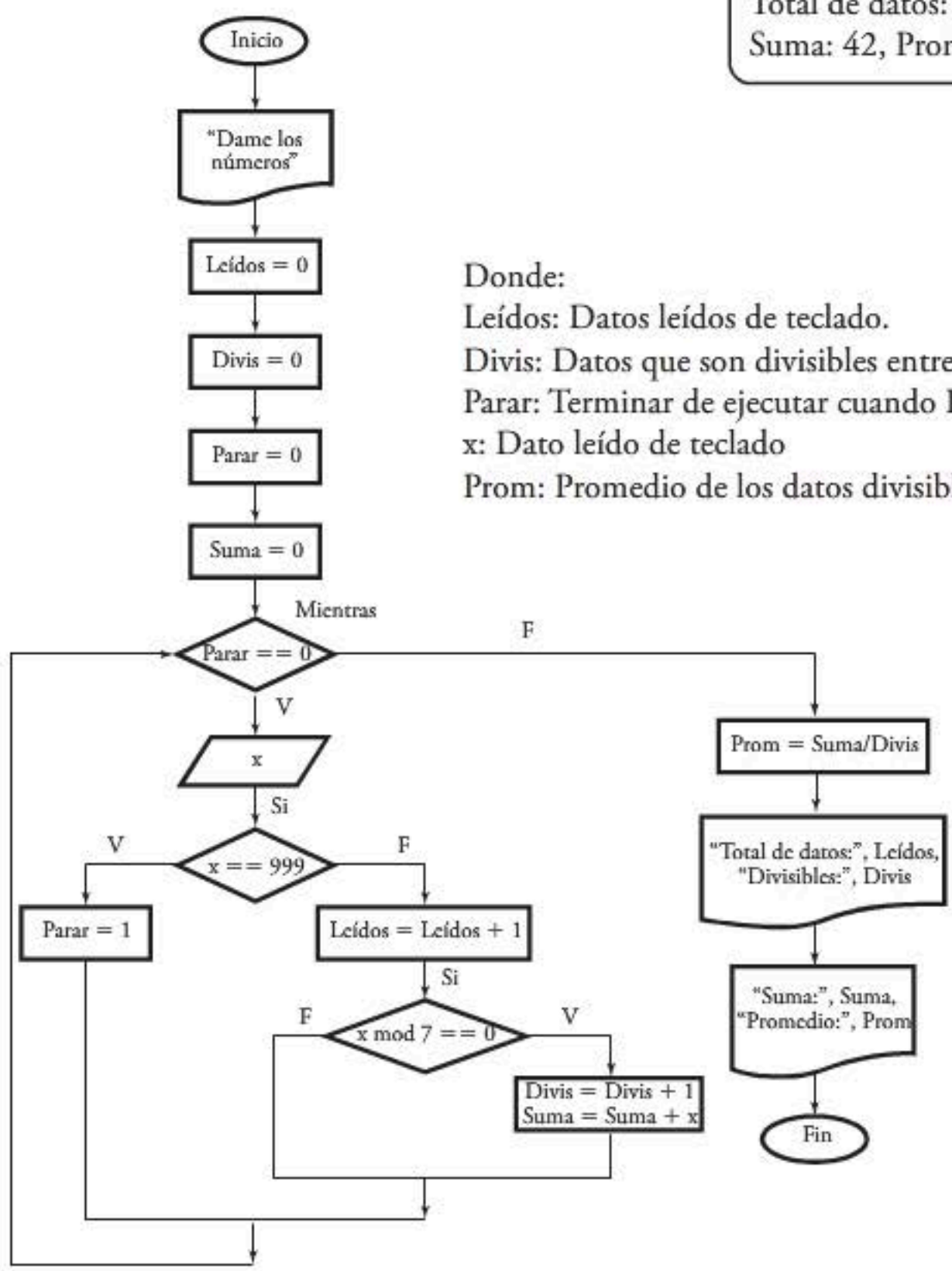
En el diagrama anterior se lee el código de la figura geométrica y dependiendo de dicho código se leerá de teclado la dimensión requerida para calcular el área de la figura, en el caso del triángulo y rectángulo se requieren la base y altura, para el cuadrado es suficiente un lado, así como para el círculo la dimensión que permite calcular su área es el radio. Después de leer las dimensiones de la figura correspondiente, se calcula el área e imprime su resultado.

En este caso se identificó la figura geométrica usando la instrucción condicional múltiple *Seleccionar* pero bien pudieron haberse usado condiciones dobles anidadas para determinar de qué figura geométrica se trata.

Ejemplo 2.10. Elaborar un diagrama de flujo para leer números de teclado, contar el total de datos leídos; contar, sumar y promediar los números que son divisibles entre 7. Dejar de leer datos hasta que el usuario introduzca como dato el número 999 sin considerar éste último.

Para programas pequeños no hay mejor análisis que imaginar la información que mostrará la pantalla al momento de ejecutar el programa. Considerar que la interfaz de salida es la siguiente:

```
Dame los números:
10
21
7
55
14
999
Total de datos: 5, Divisibles: 3
Suma: 42, Promedio: 14
```



Donde:
Leídos: Datos leídos de teclado.
Divis: Datos que son divisibles entre 7.
Parar: Terminar de ejecutar cuando Parar = 1
x: Dato leído de teclado
Prom: Promedio de los datos divisibles entre 7

En el diagrama de flujo anterior se utilizó el ciclo "*Mientras*" para ejecutar varias veces las instrucciones que se encuentran dentro de él. La computadora ejecutará dicho bloque de instrucciones hasta que "*Parar*" deje de valer 0, lo cual sucede cuando se lee el valor de 999 en x. También se puede observar que cuando se cuentan y suman cantidades es necesario inicializar en cero las variables que se utilizarán para llevar a cabo este control, en este caso son: *Leídos, Divis y Suma,* variables que se incrementarán en cada iteración.

Ciclos anidados. Al momento de elaborar programas es común que se requiera uno o más ciclos dentro de otro. Esto permite ejecutar varias veces un bloque de instrucciones que se encuentran dentro de ellos. Se pueden combinar los ciclos: *Mientras*, *Hacer–Mientras* y *Desde*, según sea necesario.

Ejemplo 2.11. Elaborar diagramas de flujo para imprimir las tablas de multiplicar del 1 al 10 Usando ciclos anidados de acuerdo a los siguientes incisos:

a) Dos ciclos *Desde*.
b) Un ciclo *Desde* dentro de un ciclo *Mientras*.
c) Un ciclo *Mientras* dentro de un ciclo *Hacer–Mientras*.

Considerar que una salida típica es como lo muestra la siguiente interfaz:

Tablas de Multiplicar del 1 al 10
Tabla del 1
$1 \times 1 = 1$
$1 \times 2 = 2$

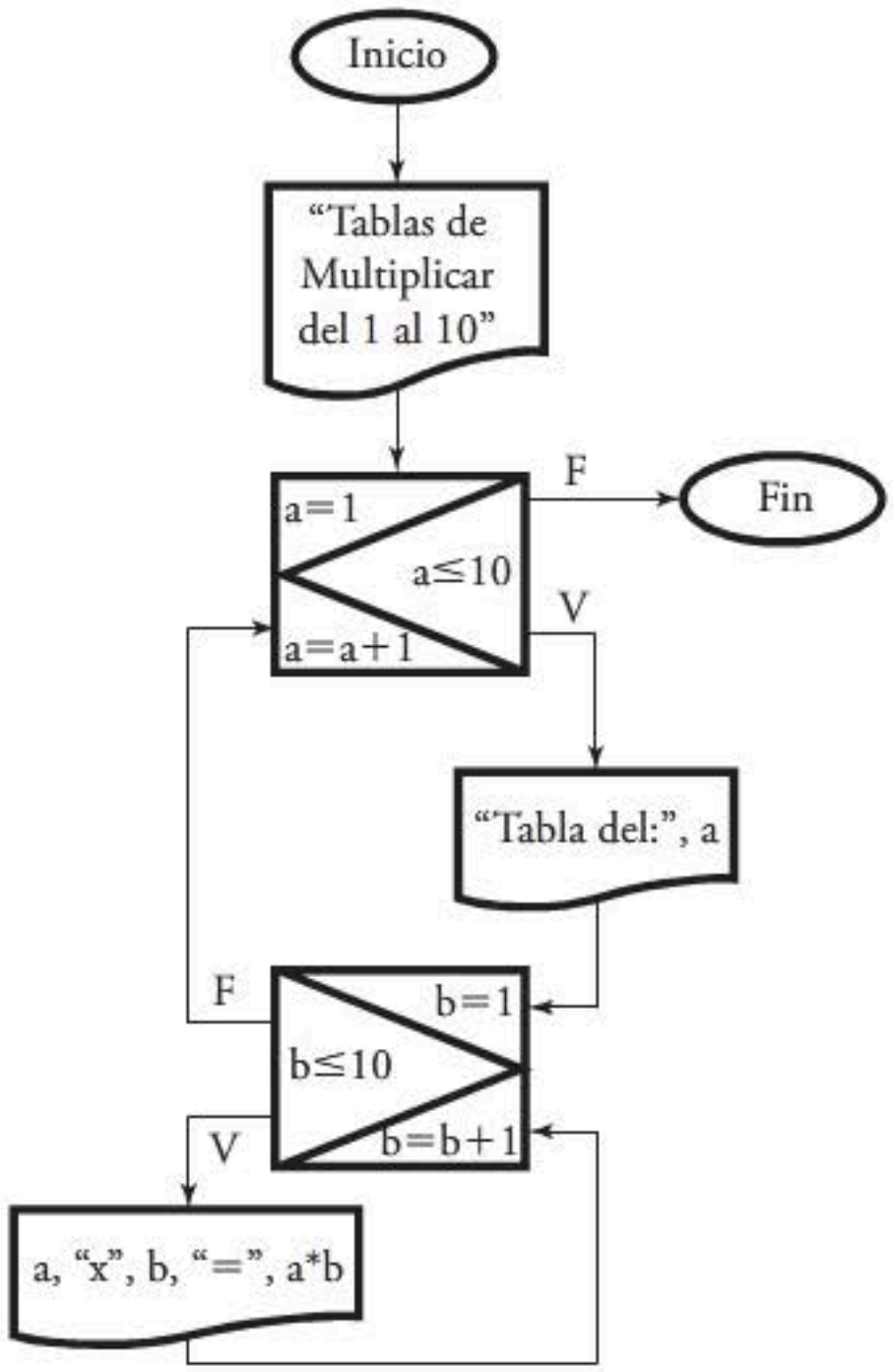
$1 \times 10 = 10$

Tabla del 2
$2 \times 1 = 2$
$2 \times 2 = 4$

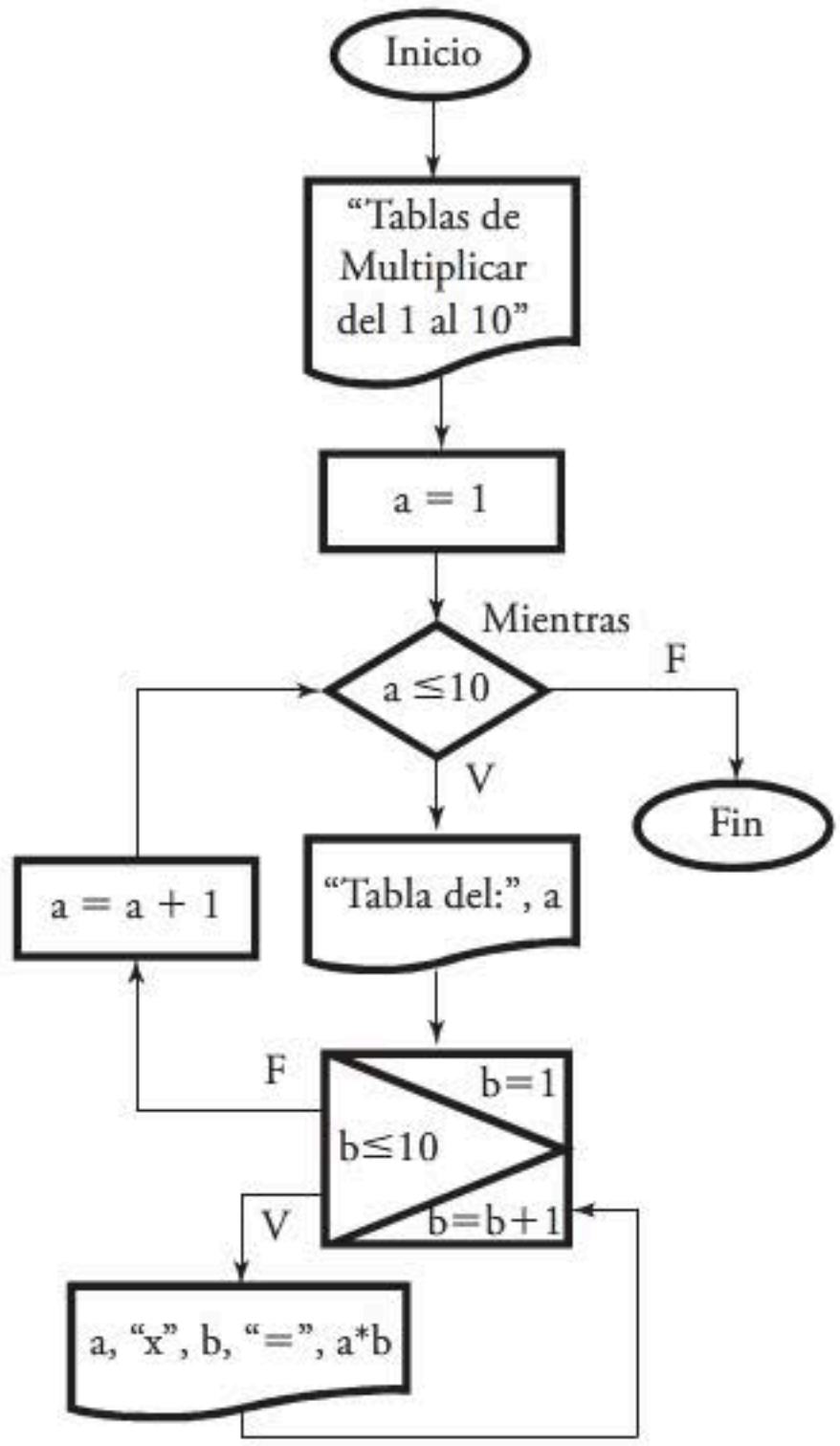
$2 \times 110 = 20$

Tabla del 10
$10 \times 11 = 10$
$10 \times 12 = 20$
$10 \times 110 = 100$

Con dos ciclos *Desde*



Un ciclo *Desde* dentro de un *Mientras*



En este caso no fue necesario leer nada de teclado porque el problema no lo pide, sino que al momento de ejecutar el programa la computadora desplegará en la pantalla las tablas de multiplicar del 1 al 10 en forma vertical, como lo muestra la interfaz de salida. En caso de que se deseen las tablas de multiplicar de n a m, se deberán leer los valores de n y m antes de los ciclos anidados y cambiar el 10 por la letra que corresponda. De la misma manera si se desea que las tablas estén distribuidas de otra forma se tendrán que hacer los arreglos correspondientes al diagrama de flujo. Observar que en el primer diagrama el símbolo gráfico del ciclo *Desde* se invierte con la finalidad de ocupar menor espacio.

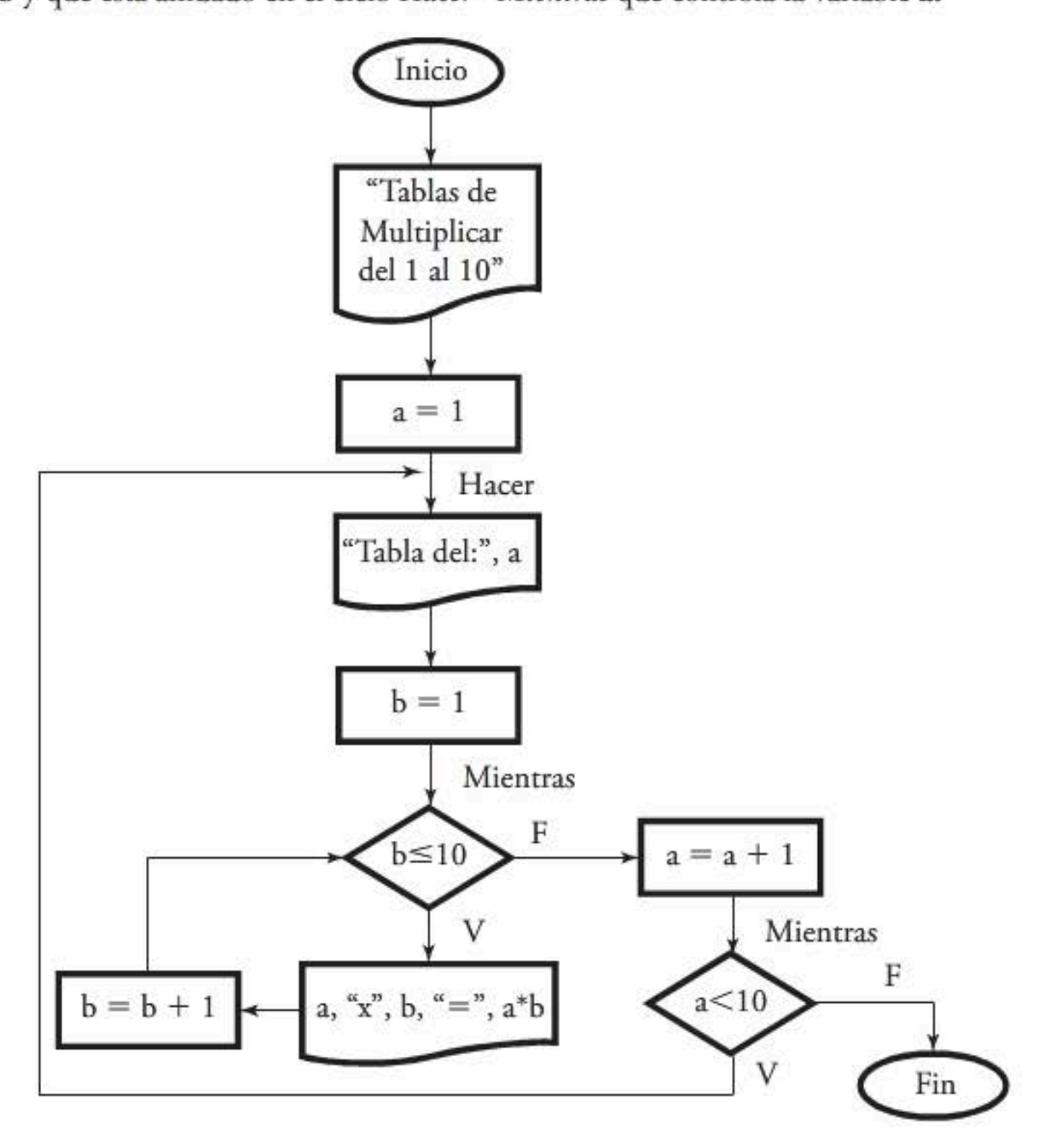
En ambos diagramas primeramente se imprime el mensaje "Tablas de multiplicar del 1 al 10", posteriormente se le asigna el valor inicial a la variable a (a=1). Después de asignar el valor inicial se preguntará si se cumple la condición (a≤10), si es verdadero imprime el mensaje de la primera tabla "Tabla del 1" considerando que el mensaje entre comillas lo imprime tal cual, y la letra "a" la sustituye por su valor. A continuación asigna el valor inicial a la variable b (b=1), pregunta si se cumple la condición (b≤10), en caso afirmativo imprimirá la primera línea de la primera tabla "1 × 1 = 1". Observar que los textos están entre comillas y las variables no deben tener comillas, ya que se requiere el valor de dichas variables no las

letras, observar también que las comas actúan como separadores de textos y variables pero no aparecen en la salida. Notar también que es posible realizar operaciones dentro de la instrucción imprimir (En lugar de esto se pudo haber multiplicado antes los valores de la variables, ejemplo c=a*b y después en la instrucción imprimir en lugar de a*b colocar la variable c).

Después se incrementa el valor de b (b=b+1) y se pregunta si se cumple la condición (b≤10), como es verdadero se imprime la segunda línea "1 × 2 = 2". Posteriormente la variable b toma el valor 3, 4, 5,…10 y 11, cuando b=11 se sale del ciclo e incrementa el valor de a (a=a+1) y pregunta si se cumple la condición (a≤10), como es verdadero imprime el mensaje de la segunda tabla "Tabla del 2", e inicializa nuevamente la letra b (b=1), pregunta si se cumple la condición (b≤10) y así sucesivamente; de tal manera que se puede observar que cada vez que se incrementa el valor de la variable **a** en 1, la variable **b** se incrementa 11 veces aunque este último valor solo le sirve para que deje de cumplirse la condición del ciclo y salir de él.

Un ciclo *Mientras* dentro de un *Hacer–Mientras.*

Observar que el ciclo *Hacer–Mientras* comienza a partir de donde está la palabra "Hacer" y termina donde está el segundo rombo con la palabra "Mientras". Se nota claramente que el ciclo *Mientras* controla la variable **b** y que está anidado en el ciclo *Hacer–Mientras* que controla la variable **a.**



2.3.3 Pseudocódigo

El Pseudocódigo es una manera aproximada de representar algoritmos usando para ello palabras de un idioma natural para simular un lenguaje de programación, indicando los pasos que integran dicho algoritmo. El pseudocódigo está hecho para que las personas representen o comprendan de una manera clara y sencilla el funcionamiento del algoritmo. Aunque las computadoras no pueden entender esta forma de representación porque no es un lenguaje de programación.

El pseudocódigo no obedece a reglas sintácticas de ningún idioma en particular y puede cambiar de una fuente de información a otra porque no es una representación estandarizada, ya que una persona puede representar los pasos del algoritmo de una manera muy cercana al lenguaje de programación que utilizará al momento de codificar el programa, o bien representar esos pasos del algoritmo apegado a una narración de lo que se hace, con lenguaje natural y apoyándose con algunos símbolos matemáticos. Cuando se utiliza pseudocódigo por lo general se omite la declaración de las variables y la llamada de funciones o procedimientos por lo general son reemplazadas por medio de una frase en lenguaje natural que indica lo que debe hacer la función o procedimiento que se está llamando.

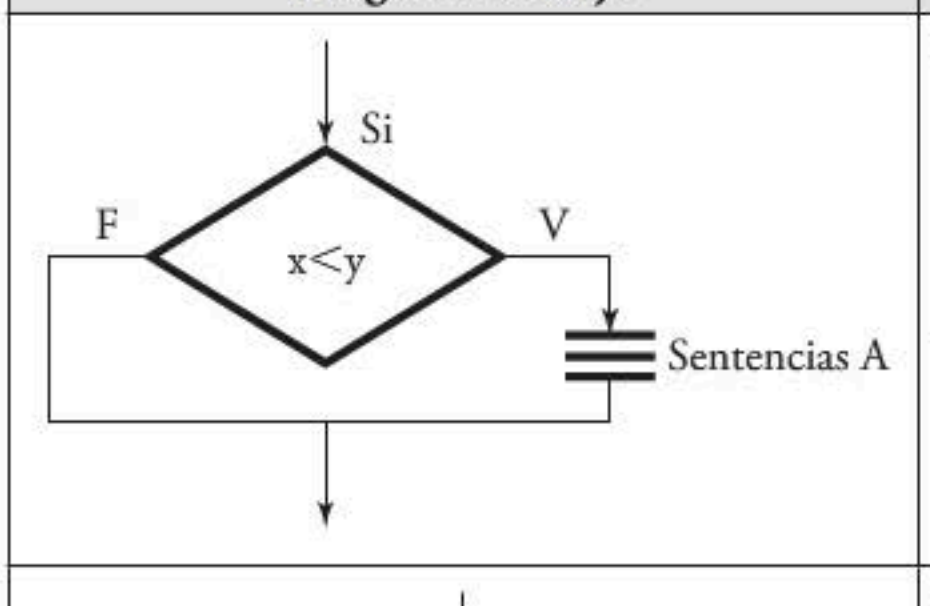
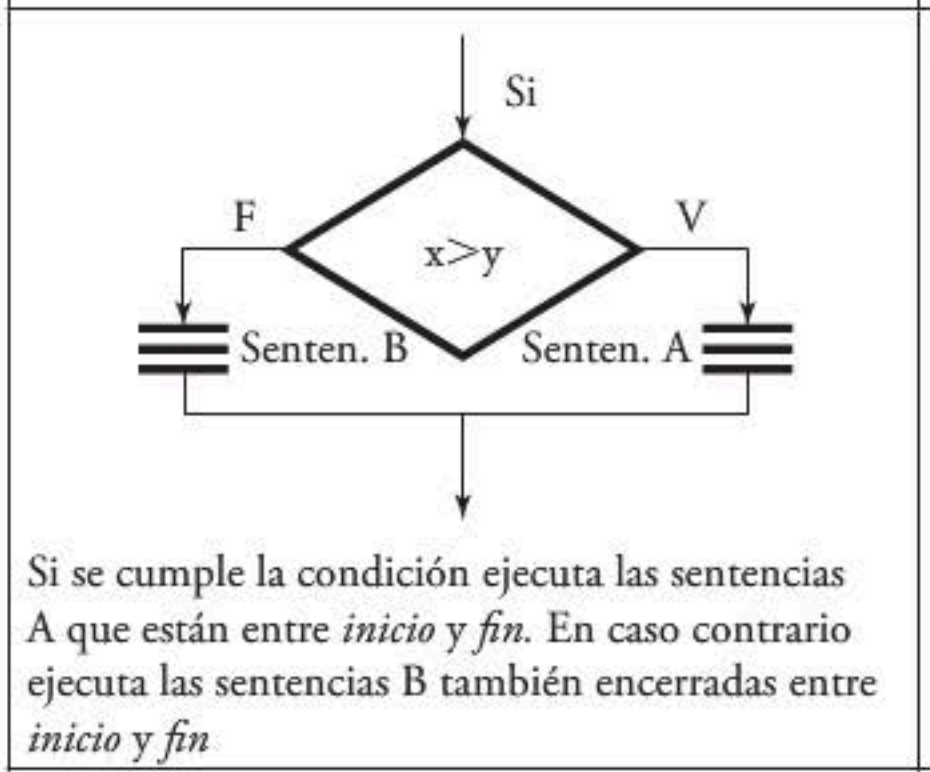
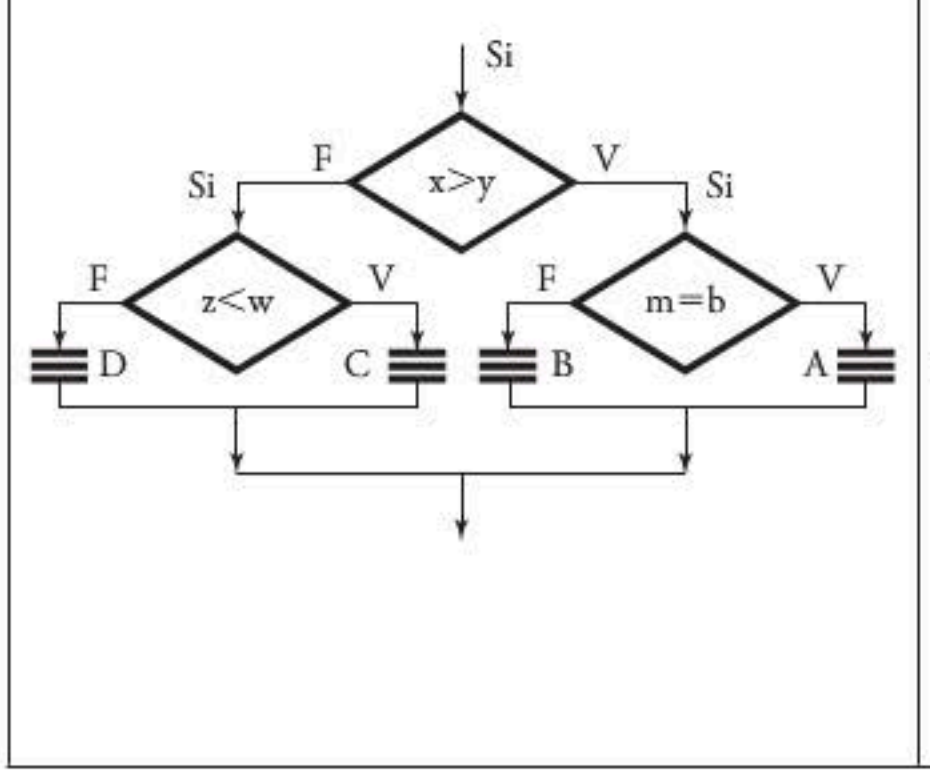
El pseudocódigo facilita la programación y solución de problemas, es fácil de utilizar y manejar, es independiente del lenguaje de programación que se usará para la representación del problema, lo cual lo hace transportable, los algoritmos representados con pseudocódigo son compactos ya que a diferencia de los diagramas de flujo no requieren de mucho espacio. Más allá de las ventajas anteriores, el pseudocódigo permite la representación de algoritmos de manera clara, concisa y comprensible para todas las personas, ya que no se requiere conocer algún lenguaje de programación para entender los algoritmos representados con pseudocódigo.

A continuación se presenta simbología de los diagramas de flujo con su equivalente en pseudocódigo. Las palabras clave propias del pseudocódigo se distinguen de la información restante, porque están escritas en minúscula y con letra cursiva.

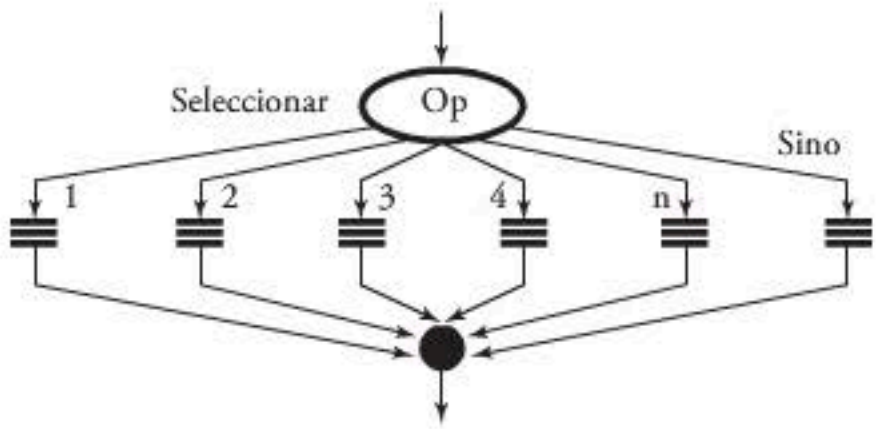
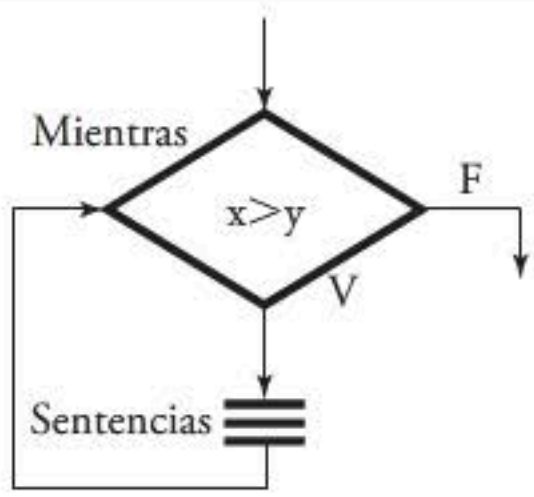
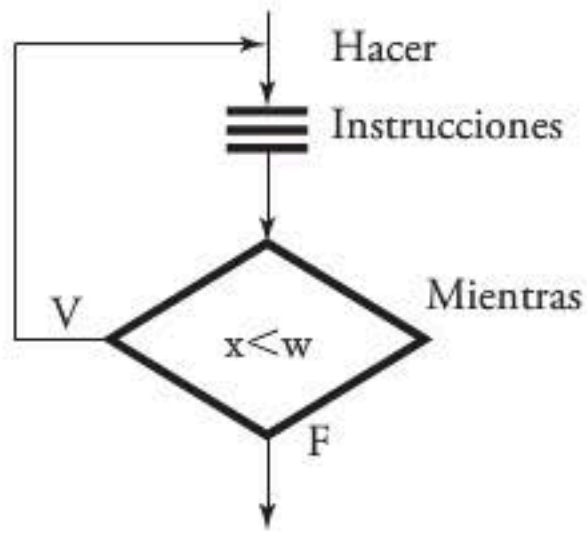
Equivalencias entre diagramas de flujo y pseudocódigo	
Diagrama de flujo	**Pseudocódigo**
	inicio o *fin*
x = y − 1	x = y – 1
x	*leer* x
"Suma:", s	*Imprimir* "Suma:",s

(*continúa*)

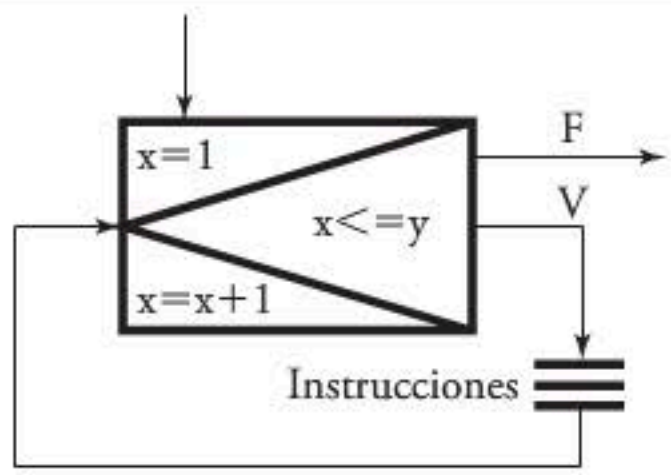
(*continuación*)

Equivalencias entre diagramas de flujo y pseudocódigo	
Diagrama de flujo	**Pseudocódigo**
	*si* (x<y) *inicio* – – – *Sentencias A* – – – – – *fin*
 Si se cumple la condición ejecuta las sentencias A que están entre *inicio* y *fin*. En caso contrario ejecuta las sentencias B también encerradas entre *inicio* y *fin*	*si* (x>y) *inicio* – – – – Sentencias A *fin* *sino* *inicio* – – – Sentencias B – – – *fin*
	*si* (x>y) *si* (m=b) *inicio* Sentencias A *fin* *sino* *inicio* Sentencias B *fin* *sino si* (z<w) *inicio* Sentencias C *fin* *sino* *inicio* Sentencias D *fin*

(*continuación*)

Equivalencias entre diagramas de flujo y pseudocódigo	
Diagrama de flujo	**Pseudocódigo**
 Dependiendo del valor de la variable op se ejecutará el bloque del caso 1,2,3,..,n o bien el bloque alterno (sino), en caso de que no se seleccione alguna de las opciones contempladas. Observar que después de ejecutar el bloque de sentencias correspondientes a cada opción se deberá romper para que no siga ejecutando las sentencias de otras opciones	Seleccionar (op) *inicio* caso 1: – – – – Sentencias A *romper* caso 2: – – – – – Sentencias B *romper* . . caso n: – – – – – Sentencias N *Romper* *sino* – – – – Sentencias alternativas *romper* *fin seleccionar*
	*mientras* (x>y) *inicio* – – – – Sentencias – – – *fin*
	*hacer* *inicio* – – – Instrucciones – – – *fin* *mientras* (x<w)

(*continuación*)

Equivalencias entre diagramas de flujo y pseudocódigo	
Diagrama de flujo	**Pseudocódigo**
	*desde* (x=1; x<=y; x=x+1) *inicio* – – – – Instrucciones – – – – *fin*

Ejemplo 2.12. Escribir un algoritmo en pseudocódigo para imprimir una tabla de equivalencias entre las temperaturas Centígrada (C), Fahrenheit (F) y Kelvin (K). Leer: inicio, final e incremento en grados centígrados y realizar los cálculos necesarios para estructurar la tabla, considerando que las fórmulas para convertir de una escala a otra son:

$$F = \frac{9}{5}C + 32 \qquad K = C + 273.1$$

```
inicio
  real C, F, K, incre, n;
  imprimir "Inicio:";
  leer C;
  imprimir "Incremento:";
  leer incre;
  imprimir "Final: ";
  leer n;
  imprimir "Centígrados   Fahrenheit   Kelvin";
  mientras (C<= n)
    inicio
     F = 9*C/5 + 32;
     K = C + 273.1;
    imprimir C, F, K;
    C = C + incre;
  fin
fin
```

Una salida típica es:

```
Inicio: −40
Incremento: 10
Final: 100
Centígrados   Fahrenheit   Kelvin
   −40          −40        233.1
   −30          −22        243.1
   ……           ……         ……
   100          212        373.1
```

En el algoritmo no se está validando la información. Por ejemplo se espera que el usuario ingresará una cantidad numérica para: inicio, incremento y final, pero no considera el caso en que se le den letras como datos, tampoco contempla el caso en que el límite de la tabla sea menor que el inicial. Por conveniencia se usa la temperatura centígrada leída inicialmente como C, para evitar utilizar otra variable.

Ejemplo 2.13. Escribir un algoritmo en pseudocódigo para desplegar en la pantalla la siguiente lista de precios:

Código	Producto	Precio
1	Hamburguesa	$25.00
2	Hot dog	$18.00
3	Refresco	$10.00

Considerar que se lee el código del producto y la cantidad de productos correspondientes a ese código y que con esos datos calculará un subtotal. Se estará leyendo el código y la cantidad de un producto hasta que el código leído sea "99" lo cual significa que ya no hay más productos que leer e imprimirá el total de la cuenta. Una salida típica lo muestra la siguiente interfaz.

```
Código    Producto        Precio
   1      Hamburguesa     $25.00
   2      Hot dog         $18.00
   3      Refresco        $10.00

Código: 2
Cantidad: 3
3 Hot dog = $54.00
Código: 3
Cantidad: 4
4 Refrescos = $40.00
Código: 99
Total = $94.00
```

```
inicio
  entero cod, tot, cant, sbtot;
  imprimir "Código      Producto        Precio";
  imprimir "    1       Hamburguesa     $25.00";
  imprimir "    2       Hot dog         $18.00";
  imprimir "    3       Refresco        $10.00";
  cod = 0;
  tot = 0;
  mientras (cod != 99)
     inicio
       imprimir "Código: ";
       leer cod;
     si (cod != 99)
       inicio
         imprimir "Cantidad:";
         leer cant;
         seleccionar (cod)
```

```
    caso 1:
        subtot = cant*25;
        imprimir cant, "Hamburguesas = ",subtot;
        romper;
    caso 2:
        subtot = cant*18;
        imprimir cant, "Hot dog = ",subtot;
        romper;
    caso 3:
        subtot = cant*10;
        imprimir cant, " Refrescos = ",subtot;
        romper;
  sino:
            imprimir " Código no válido:";
            subtot=0;
            romper
    fin seleccionar
  tot = tot + subtot;
 fin
 imprimir "Importe total= $",tot;
fin
```

La finalidad de dar un valor de cero a la variable cod (cod=0) antes del ciclo mientras, es para que pueda entrar al ciclo, igual funciona con otro valor diferente a 99 porque ese valor lo cambia inmediatamente al ingresar al ciclo. Observar que las instrucciones del ciclo *mientras* se deberán encerrar entre las palabras *inicio* y *fin*, para delimitar las sentencias que se ejecutarán mientras se cumpla la condición. La instrucción "***seleccionar* (cod)***"*, deberá encerrar entre *inicio* y *fin* todas sus opciones, para indicar hasta donde llega su alcance. Cuando el código es válido (1, 2 o 3) las sentencias a ejecutarse comienzan después de los dos puntos de la opción y terminan al llegar a la palabra *romper*. Si no se le pone *romper* a cada uno de los casos de *seleccionar*, la computadora seguiría ejecutando instrucciones aunque dichas sentencias pertenezcan a un caso diferente.

Es obvio que la salida no es muy estética porque no se están usando cajas de texto, tipos de letra, colores, coordenadas de pantalla y otras instrucciones gráficas propias de todos los lenguajes de programación para dar una presentación bonita, pero no hay que perder de vista que estos son los primeros algoritmos y que todavía no se está usando un lenguaje de programación.

2.3.4 Diagramas Nassi−Shneiderman (N−S)

Fueron desarrollados en 1972 por Isaac Nassi y Ben Shneiderman con la finalidad de representar los algoritmos combinando palabras clave del pseudocódigo (*inicio, fin, repetir, hasta, mientras, hacer, desde, si*, entre otras) con las figuras geométricas de los diagramas de flujo (*cuadros, rectángulos y triángulos*), agrupando los elementos de tal manera que permita ilustrar con mayor claridad la estructura de los algoritmos.

En todos los lenguajes de programación es necesario definir las variables para el manejo de información como algún tipo de dato conocido (entero, real, cadena, booleana, etc.). La definición de variables se considera en los diagramas N−S con el objetivo de acercarse más al lenguaje de programación a utilizar. La manera de declarar las variables es escribiendo el tipo de dato y la lista de variables que tendrán ese tipo de dato como se muestra a continuación:

entero base, altura;
real área, volumen, perímetro;
caracter símbolo;
cadena nombre;

Lo cual significa que las variables "base" y "altura" pueden manejar cantidades enteras, "área", "volumen" y "perímetro" cantidades reales, la variable "símbolo" puede manejar una letra, un número o cualquier otro caracter y finalmente la variable "nombre" maneja cadenas de caracteres conformadas por letras y/o números.

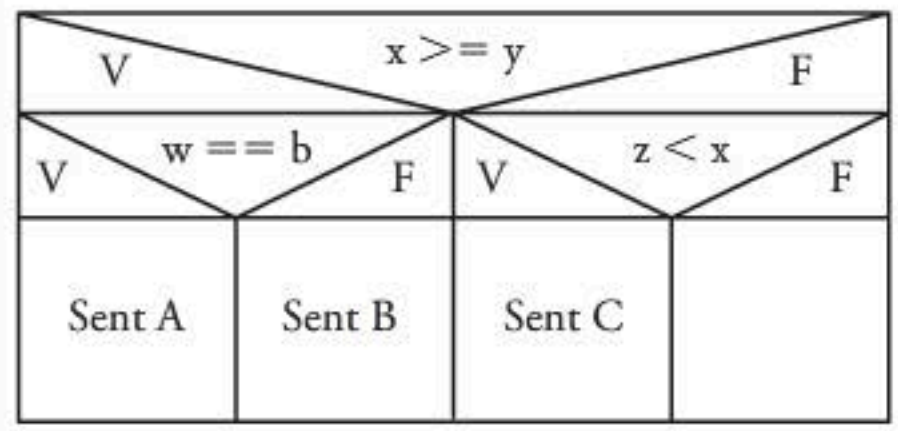
De la misma manera que en los diagramas de flujo y pseudocódigo los diagramas N−S tienen una forma de representación para cada una de las instrucciones usadas en la representación de algoritmos. A continuación se tiene una tabla de comparación entre pseudocódigo y los diagramas N−S.

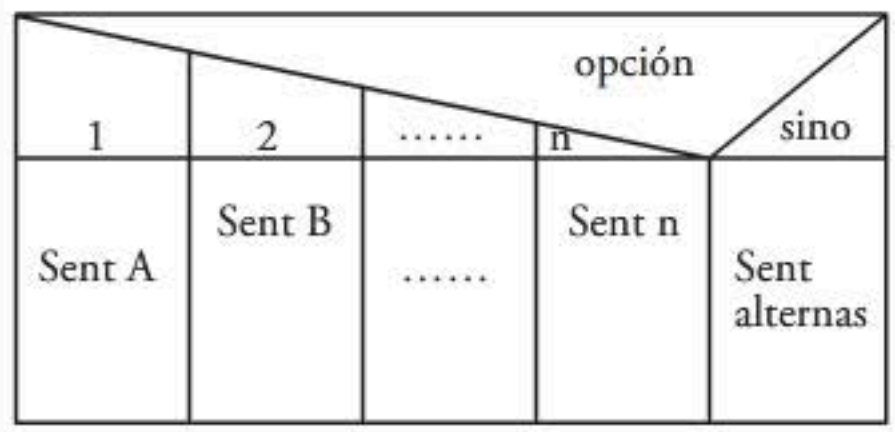
Pseudocódigo	Diagrama N-S
inicio	*inicio*
fin	*fin*
x = 3*y−1	x = 3*y−1
leer a,b	*leer* a,b
imprimir "Suma=", s	*imprimir* "Suma=", s
si (x<y) *inicio* – – – Sentencias A – – – – – *fin*	V x < y F Sentencias A
si (x>=y) *inicio* – – – – Sentencias A *fin* *sino* *inicio* – – – Sentencias B – – – *fin*	V x >= y F Sentencias A Sentencias B

(*continúa*)

(*continuación*)

Pseudocódigo	Diagrama N-S

si (x >= y)
  si (w = = b)
  inicio
  – – –
  Sentencias A
  fin
  sino
  inicio
  – – –
  Sentencias B
  fin
 	
sino si (z < x)
  inicio
  – – –
  Sentencias C
  fin
	

Seleccionar (opción)
caso 1: – – – – – – –
  – – – – – –
  Sentencias A
  romper
caso 2: – – – – – –
  – – – – – –
  Sentencias B
  romper
  .
  .
caso n: – – – – – –
  – – – – – –
  Sentencias n
  romper
  sino
  – – – – – –
  – – – – – – –
  Sentencias alternas
  romper
  fin // seleccionar
	

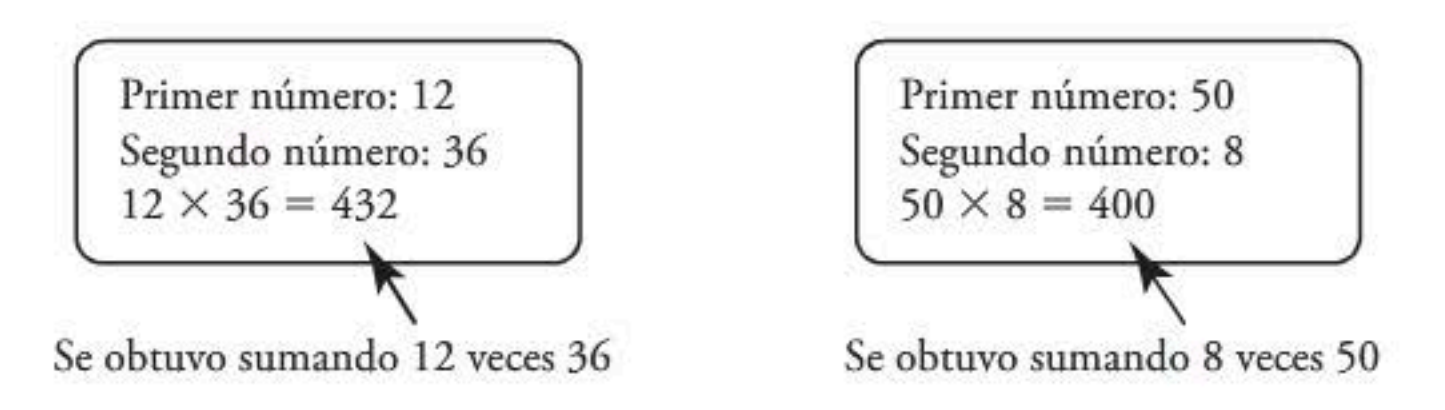
(*continúa*)

(*continuación*)

Pseudocódigo	Diagrama N-S
mientras (x<y) *inicio* – – – – Sentencias – – – *fin*	mientras (x<y) inicio …… Sentencias …… fin
hacer *inicio* – – – – – Instrucciones – – – – – – *fin* *mientras* (x<w)	hacer inicio Instrucciones fin mientras (x<w)
desde (x = inicial; x<=final; x=x+1) *inicio* – – – – Instrucciones – – – – *fin*	desde (x = inicial; x<= final; x=x+1 inicio Instrucciones fin

Es importante mencionar que cuando hay más de una instrucción secuencial como: *leer, hacer, imprimir, inicio* y *fin*, se colocan en rectángulos individuales pero sin dejar espacio entre ellos (aunque también podrían colocarse en un mismo rectángulo).

Ejemplo 2.14. Escribir un algoritmo usando diagramas N−S para leer dos números enteros positivos de teclado. La salida deberá ser el resultado de multiplicar los dos números leídos, pero encontrando dicho resultado por medio de sumas sucesivas. Considerar además que se hará la menor cantidad de sumas para obtener el resultado. Salidas típicas se muestran a continuación.



Respuesta:

```
Inicio
    entero a, b, May, Men, Prod, Paso;
    imprimir "Primer número:";
    leer a;
    imprimir "Segundo número:";
    leer b;
    V        a > b        F
    May = a;             May = b;
    Men = b;             Men = a;
    Prod = 0;
    Paso = 0;
  mientras (Paso < Men)
        Inicio
            Prod = Prod + May;
            Paso = Paso + 1;
          fin
        imprimir a, "x", b, "=", Prod,
        fin
```

Ejemplo 2.15. Escribir un algoritmo en:

a) Diagrama N−S
b) Diagrama de flujo.
c) Pseudocódigo.

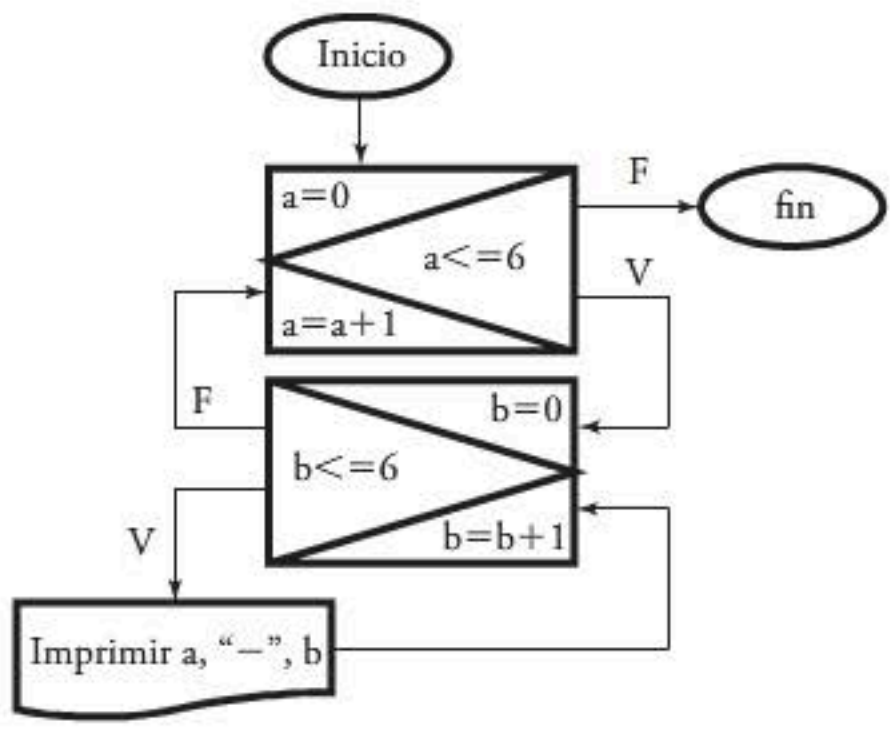
Para imprimir todos los posibles valores de las fichas del dominó. La salida se muestra en la interfaz.

```
0-0
0-1
0-2
– – –
– – –
6-6
```

a) Diagrama N−S

```
Inicio
entero a, b
  desde (a=0; a<=6; a=a+1)
      desde (b=0; b<=6; b=b+1)
          imprimir a, "−", b
fin
```

b) Diagrama de Flujo



c) Pseudocódigo

```
inicio
desde (a= 0;a<=6;a=a+1)
   desde (b=0;b<=6,b=b+a)
imprimir a, "−", b
fin
```

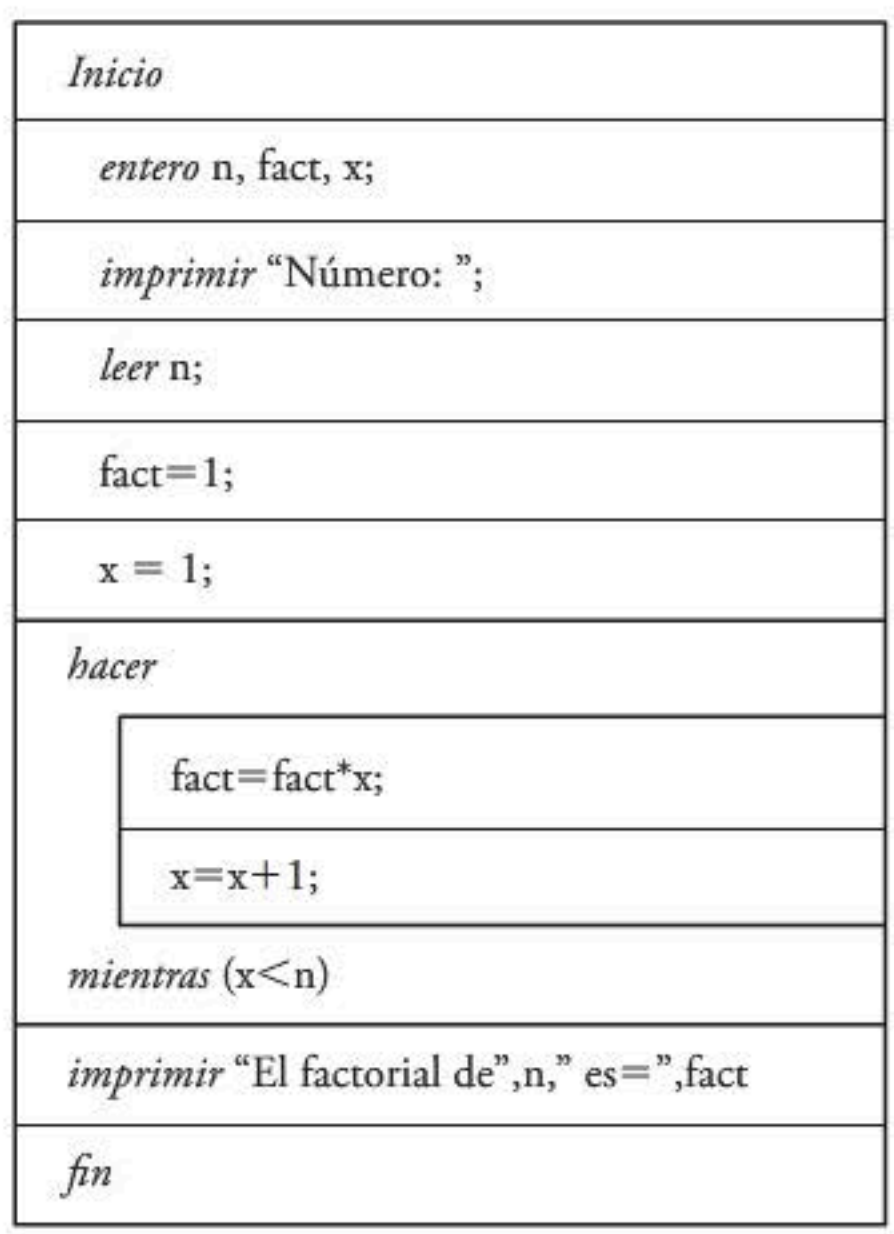
Al correr el programa en este caso no es necesario leer ninguna información de teclado ya que la salida será la misma cada vez que se ejecuta. Observar que dentro del ciclo *desde* (a=0; a<=6; a=a+1) solamente está la instrucción *desde* (b=0; b<=6; b=b+1), y que dentro de este último ciclo está solamente la instrucción (*imprimir* a, "−",b), razón por la cual no fue necesario utilizar las palabras *inicio* y *fin* para agrupar las instrucciones que pertenecen a cada uno de los ciclos. Recordar que se usan *inicio* y *fin* solamente cuando hay más de una instrucción dentro del ciclo.

Ejemplo 2.16. Escribir un algoritmo para encontrar el factorial de un número entero positivo en:

a) Diagrama N−S usando ciclo *hacer−mientras*
b) Pseudocódigo usando ciclo *desde*.
c) Diagrama de flujo usando ciclo *mientras*

Respuestas:

a) Diagrama N−S con ciclo repetir



Interfaz de salida

```
Número: 7
El factorial de 7 es = 5040
```

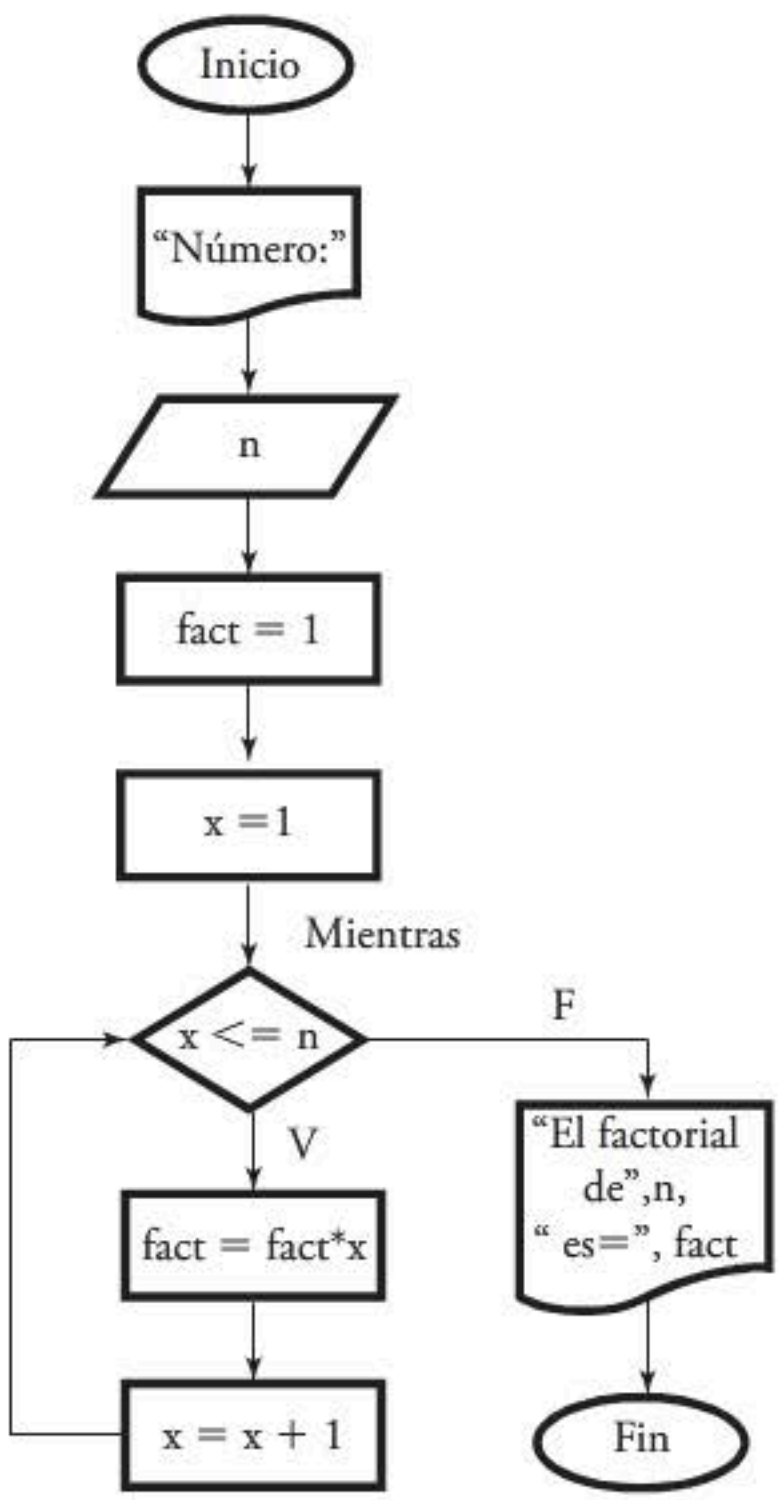
Recordar que:
0!=1
1!=1
n!=1×2×. ×n

b) Pseudocódigo usando ciclo desde

```
inicio
   entero n, fact, x;
   imprimir "Número:
   leer n;
   fact = 1;
   desde (x = 1; x<= n; x=x+1)
     fact = fact*x;
    imprimir "El factorial de ",n," es =",fact;
 fin
```

Observar que la variable contador x se inicializa (x =1) e incrementa (x=x+1) en el mismo ciclo. Tampoco es necesario usar *inicio* y *fin* para enmarcar una sola instrucción que está en el ciclo desde.

c) Diagrama de flujo usando ciclo mientras.



Hay diferencias en el algoritmo propias a las características de los ciclos pero la salida se conserva.

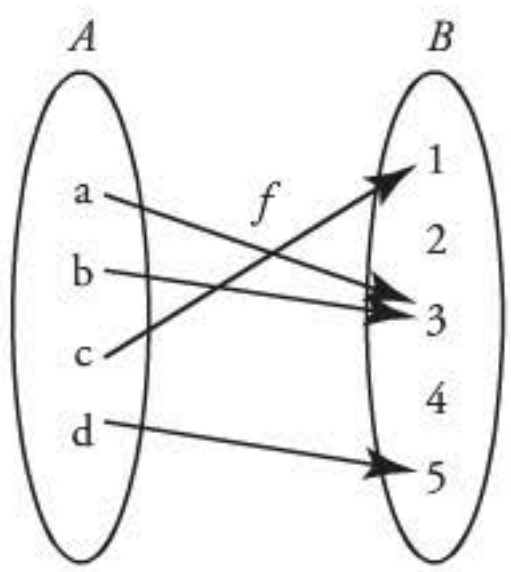
2.4 Diseño de algoritmos de funciones

Cuando los programas son grandes y complejos es conveniente dividirlos en pequeños subprogramas que permitan resolverlos de una manera más fácil, cada uno de esos subprogramas puede ser una función, la cual se ejecuta en el momento requerido, esto facilita la programación y revisión del programa. Además es posible aprovechar el código de las funciones desarrolladas, en programas en donde sean requeridas posteriormente.

Definición de función en matemáticas.

Sean dos conjuntos no vacíos ***A*** y ***B***. Una función ***f*** de ***A*** en ***B*** (***f:A→B***) es aquella que a cada elemento ***a*** de un conjunto ***A*** asigna un único elemento ***b*** de un conjunto ***B***.

Ejemplo 2.17. Sean los conjuntos A={a,b,c,d} y B={1,2,3,4,5}, y la función ***f:A→B*** en donde f={(a,3),(b,3),(c,1),(d,5,)}



Para cada elemento del conjunto A, existe solamente un elemento del conjunto B.

Todos los elementos de A deben estar relacionados. Algunos elementos de B podrían estar relacionados con más de un elemento de A. Algunos elementos de B podrían no estar relacionados.

Considerar que los elementos del conjunto ***A*** son artículos de una tienda de abarrotes, a=refresco, b=galletas, c=dulces, d=cigarros y que los elementos del conjunto ***B*** son precios en pesos: 1, 2, 3, 4,…. Para cada elemento del conjunto ***A*** le corresponde un elemento del conjunto ***B***, esto significa que todos los elementos del conjunto ***A*** tienen un solo precio. Si algún producto de la tienda de abarrotes no tiene un solo precio entonces no se trata de una función ya que un mismo producto no puede tener dos valores distintos a la vez (esto se puede observar en la gráfica anterior, porque de cada elemento del conjunto **A** sale solamente una flecha que apunta a un elemento del conjunto **B**). Además todos los elementos del conjunto ***A*** deben estar relacionados con algún elemento del conjunto ***B***, lo cual significa que cada producto de la tienda de abarrotes debe tener un precio (si algún elemento del conjunto ***A*** no tuviera flecha tampoco sería función). Sin embargo es válido que un elemento del conjunto ***B*** esté relacionado con más de un elemento del conjunto **A**, por ejemplo los elementos **a** y **b** tienen el mismo precio **3.** También es válido que no estén relacionados elementos del conjunto ***B*** con algún elemento del conjunto ***A***, como ocurre con los elementos 2 y 4 (lo que significa que la tienda de abarrotes no tiene productos que tengan un valor de 2 ó 4 pesos). En la grafica anterior ***f*** es el nombre de la función, **a, b, c** y **d** son argumentos (o parámetros) con los cuales se evalúa la función y **1, 2, 3, 4, 5** son posibles valores de la función. En este caso:

f(a)=3 f(b)=3 f(c)=1 f(d)=5

Algunas veces las funciones pueden requerir más de un parámetro para llevar a cabo la evaluación, pero solamente debe ser un valor el obtenido por las funciones para que se clasifiquen como tales. Por

ejemplo considerar que en un comercio se tienen cervezas de las marcas:{corona, sol, tecate}, que cada una de ellas puede usar los envases:{lata, botella} y dos diferentes presentaciones:{clara, obscura}. La función encargada de determinar el precio de una cerveza necesita tres parámetros y el resultado de la evaluación es solamente el precio de la cerveza. Una llamada típica a la función es:

y = precio(marca, envase, presentación)

Con los valores de los parámetros marca="sol", envase="lata" y presentación="clara", la función cuyo nombre es "**precio**" determina el valor de la cerveza y se lo asigna a la variable "**y**".

Todos los lenguajes de programación tienen sus propias funciones estándar, ejemplos de funciones estándar son.

y = cos(a); z = abs(b); x = sqrt(c);

Dependiendo del lenguaje de programación, las funciones estándar pueden variar pero prácticamente todos los lenguajes manejan funciones: trigonométricas, valores absolutos de cantidades, raíz cuadrada, elevar una cantidad a cierta potencia, redondear un valor real al entero más cercano, etc. Además de las funciones estandarizadas es posible crear funciones adicionales para realizar alguna operación que no se pueden llevar a cabo con las funciones ya existentes.

2.4.1 Funciones en computación

En el área de la computación una función es un subprograma que recibe valores de entrada llamados argumentos y regresa un solo valor llamado resultado (de la misma manera en que lo hace en matemáticas). Las funciones se pueden llamar desde cualquier parte del programa. Las funciones tienen la siguiente forma general:

tipo nombre(*tipo* p_1, **tipo** p_2,....,*tipo* p_n)
inicio
tipo v_1,v_2;
tipo v_3;
– – – – –
Instrucciones de la función
– – – – –
retornar valor;
fin

Donde:
nombre: Es el nombre de la función
p_1,p_2,....p_n: Parametros 1,2,....n
tipo: Tipo de dato (entero, real, cadena, caracter, booleano, flotante, etc.)
v_1,v_2,....,v_n: Variables

Ejemplo 2.18. A continuación se tiene una función para evaluar el polinomio: $y = x^2-5x+7$

```
real polinomio(entero x)
 inicio
   real resultado;
   resultado=x^2-5*x+7;
   retornar resultado;
 fin
```

El nombre de la función es "**polinomio**" y el valor obtenido de dicha función es del tipo "*real*". Dentro del paréntesis es posible definir los parámetros de la función de algún tipo de dato, en este caso

solamente se tiene el parámetro **x** y se define como tipo "*entero*". También se pueden definir variables dentro de la función como ocurre con la variable **resultado** definida tipo *real*. La función regresa el valor calculado a la línea donde se llamó dicha función por medio de *retornar* **resultado**.

Por lo general el llamado de una función no se realiza en una línea individual sino que es parte de una instrucción. A continuación se tienen ejemplos del llamado de la función polinomio.

y=polinomio (w); *imprimir* polinomio(w); *si* (polinomio(w) > 0)

En el primer caso el valor obtenido de la función para el argumento w se le asigna a la variable "y". En el segundo se imprime directamente el valor calculado de la función, en el tercer caso; si el valor obtenido de la función es mayor a cero ejecutará la lista de instrucciones que se indiquen.

Como se mencionó anteriormente las funciones solamente regresan un valor, pero podrían necesitar más de un parámetro cuando se llama la función, esto depende de la forma en que se haya programado dicha función.

Ejemplo 2.19. Escribir un algoritmo en pseudocódigo para leer tres datos enteros de teclado. Encontrar por medio de una función el mayor de los datos leídos e imprimir el resultado.

```
Inicio /* Programa principal */
 entero  a,b,c;
  imprimir "Valor de a:";
  leer a;
  imprimir "Valor de b: " ;
  leer b;
  imprimir "Valor de c:";
  leer c;
  imprimir "El mayor de ",a,",",b, "y",c, " es:",mayor(a,b,c);
fin

entero  mayor(entero x, entero y , entero z)
  inicio
    entero res;
si (x>=y and y>=z)
res = x;
      sino
        si  (y>=x and x>=z )
          res = y;
          sino
           res = z;
    retornar res;
  fin
```

```
Valor de a: 7
Valor de b: 32
Valor de c: 5
El mayor de 7,32 y 5 es: 32
```

Al ejecutar la función con **mayor(a, b, c)**, "**x**" toma el valor de "**a**", "**y**" toma el valor de "**b**" y "**z**" toma el valor de "**c**", se realizan los cálculos y cuando se regresa el resultado de la función se destruyen los valores de **x**, **y**, y **z** ya que solamente son reconocidos dentro de la función.

Observar que la función es un pequeño subprograma el cual se ejecuta al momento de invocarlo con el nombre de la función y los parámetros correspondientes **mayor(a,b,c)**. En este ejemplo se invoca la

función en la instrucción que imprime el mayor de los números leídos, pero podría haberse asignado a una variable previamente definida antes y posteriormente imprimir dicha variable, a la cual se le asignó el valor con los siguientes dos líneas:

```
r = mayor(a, b, c);
imprimir "El mayor de ",a, ",",b, "y",c, " es:",r;
```

Es importante mencionar que en este libro se presenta solamente una de las maneras en que puede desarrollarse un determinado algoritmo y seguramente la solución que se muestra ni siquiera es la mejor. En programación las cosas se pueden realizar de diferente manera, por ejemplo la lectura de las variables a, b y c podría haberse realizado con el par de instrucciones siguientes en lugar de leerlas por separado.

```
imprimir "Dame los valores de las variables a, b y c, separados por comas: ";
leer a, b, c;
```

La salida cambia ya que en este caso la lectura se lleva a cabo en una sola línea y como se muestra en el algoritmo anterior requiere de una línea para la lectura de cada una de las variables. Lo que determina las instrucciones y su colocación en este caso, es la presentación de la salida.

Es conveniente tener en cuenta que el número de argumentos y tipos de datos de cada uno de ellos, deben de coincidir con el número, posición y tipo de dato de los parámetros de la función. Esto significa que si la función requiere tres datos de los cuales el primero es el nombre de un producto (dato tipo cadena), el segundo es una letra (dato tipo caracter) y el tercero un número (dato entero o real) y el resultado de la función es una cantidad real entonces la función podría tener el siguiente encabezado:

```
real precio (cadena marca, caracter presentación, entero capacidad)
inicio
   real pres;
   ------
   Definición de variables
   ------
   Instrucciones de la función
   ------
   retornar pres;
fin
```

La llamada a la función: **p = precio(x, y, z)** en donde x=marca="corona", y=presentación="o" y z=capacidad=600 encontraría el precio (tipo *real*) de la cerveza marca corona, presentación obscura (o) y capacidad 600 (600 ml) y lo retornará con la variable **pres**, para asignarlo finalmente a **p**. Los parámetros marca (x), presentación (y), capacidad (z), correspondientes deben coincidir en posición y tipo de dato.

Ejemplo 2.20. Elaborar un algoritmo para sumar n términos de la siguiente expresión matemática:

$$\frac{x^2}{1!}+\frac{3x^2}{2!}+\frac{5x^4}{3!}+\frac{7x^5}{4!}+\ldots$$

Usar una función para encontrar el valor del numerador y otra para calcular el valor del denominador en cada paso y considerar que x es un número real que se lee de teclado. Una salida típica es como se muestra a continuación.

Valor de n: 5
Valor de x: -2.74
Resultado=10.314

$$\frac{(-2.74)^2}{1!}+\frac{3(-2.74)^3}{2!}+\frac{5(-2.74)^4}{3!}+\frac{7(-2.74)^5}{4!}+\frac{9(-2.74)^6}{5!}$$

```
inicio    /* Programa principal */
 real x, suma;
 entero n, coeficiente, exponente, t;
 imprimir "Valor de n:";
 leer n;
 imprimir "Valor de x:";
 leer x;
 exponente = 2;
 coeficiente = 1;
 t=1;
 suma=0;
 mientras ( t < = n)
  inicio
     suma = suma + numerador(coeficiente, x, exponente) / factorial(t);
       exponente = exponente + 1;
       coeficiente = coeficiente + 2;
    t = t + 1;
  fin
   imprimir "Resultado=",suma;
fin

 real numerador (entero coeficiente, real x, entero exponente)
  inicio
   real calculo;
    calculo = coeficiente * (x^exponente);
    retornar calculo;
   fin

 entero factorial(entero w)
 inicio
    entero  i, f;
    f = 1;
    desde (i=1, i<=w, i=i+1)
        f=f*i;
    retornar f;
   fin
```

Este algoritmo está integrado por dos funciones y el programa principal.

Programa principal. Se definen las variables **x, suma** del tipo *real,* porque ambas variables deberán manejar cantidades de tipo real. Las variables **n, coeficiente, exponente, t** son de tipo *entero,* porque son las características de los datos que usarán. Se lee información de teclado y se inicializan algunas variables con los valores más convenientes antes del ciclo *mientras*, dichas variables se incrementan dentro del ciclo porque así se requiere. Observar también que la línea:

suma = suma + numerador(coeficiente, x, exponente) / factorial(t);

Hace el llamado a las funciones **numerador** y **factorial** invocándolas con: **numerador(coeficiente, x, exponente)** y **factorial(t)** respectivamente. Al momento del llamado de la función, los parámetros toman los valores correspondientes.

Función **numerador(coeficiente, x, exponente).** Cuando se invoca esta función, los valores de los parámetros de la función toman el valor de las variables del programa principal, se realizan los cálculos correspondientes y se retorna el resultado obtenido. Observar que el nombre de las variables del programa principal es el mismo que el nombre de los parámetros de la función, pero la computadora crea diferentes localidades para las variables y para los argumentos. Dentro de la función el valor con el que realiza las operaciones es el de los argumentos, de la misma manera que dentro del programa principal el valor que considera es el de las variables. Cuando sale de la función destruye sus argumentos y variables locales que se hayan definido dentro de esa función y se queda solamente con los valores de las variables del programa principal.

Función **factorial (t).** Esta función solo tiene un parámetro (w) y se definieron las variables locales enteras (i, f) que solamente son conocidas dentro de la propia función. La función se invoca con: **factorial (t),** en eso momento **w** toma el valor de **t** (w = t) y con él calcula su factorial, las variables **i** y **f** se utilizan como auxiliares para realizar el cálculo, pero tanto el parámetro **w** como las variables **i** y **f** se destruyen al salir de la función.

2.4.2 Funciones recursivas

El resultado de una función recursiva depende del resultado de la misma función para otros valores. Una característica de las funciones recursivas es que se invocan de sí misma.

Se debe tener cuidado al momento de programar una función recursiva, ya que su ejecución podría prolongarse indefinidamente y no terminar jamás. En forma natural cuando se ejecuta una función recursiva la computadora va introduciendo valores indeterminados en una pila hasta llegar a un valor conocido. A ese valor conocido se le llama "Punto de retorno". Ese valor lo sustituye en la expresión que genera el último valor no determinado que está encima de la pila, para calcular el nuevo valor conocido, mismo que sustituye en la nueva expresión que está ahora encima de la pila y así sucesivamente hasta que la pila se quede vacía y obtener de esa manera el resultado buscado.

Las funciones recursivas reciben también el nombre de funciones "bumerán" (boomerang) porque van guardando valores no determinados en la pila, hasta obtener un valor conocido y en ese momento regresa sobre su propia trayectoria, sacando cada elemento de la pila y determinando su valor, hasta llegar a la primera expresión de la pila que le permitirá calcular el valor buscado.

Ejemplo 2.21. Encontrar por medio de una función recursiva el factorial de una cantidad.

```
entero factorial(entero n )
 inicio
  entero w;
  si (n<=1)
    w = 1;
 sino
```

```
    w = factorial(n−1)*n;
    retornar w;
fin
```

Una simulación de la ejecución de la función recursiva factorial se muestra a continuación para n=6.

n=6

factorial(5)*6

En el programa solamente está determinado el valor w=1 cuando n<=1, de tal manera que para n=6 guarda en la pila la expresión matemática que no le permite saber su **valor**

n=5

factorial(4)*5
factorial(5)*6

Nuevamente guarda en la pila la expresión que no le permite encontrar el valor del factorial para n=5

n=4

factorial(3)*4
factorial(4)*5
factorial(5)*6

Guarda en la pila la expresión que no le permite encontrar el valor del factorial para n=4

n=3

factorial(2)*3
factorial(3)*4
factorial(4)*5
factorial(5)*6

Guarda en la pila la expresión que no le permite encontrar el valor del factorial para n=3

n=2

factorial(1)*2
factorial(2)*3
factorial(3)*4
factorial(4)*5
factorial(5)*6

Para n=2 nuevamente no se puede determinar el valor de tal manera que debe guardarlo nuevamente en la pila.

n=1

factorial(1)*2
factorial(2)*3
factorial(3)*4
factorial(4)*5
factorial(5)*6

Para n=1 ya es posible determinar su valor ya que factorial(0)*1=1. Este es conocido como el punto de retorno. Lo cual significa que factorial(1)=1

factorial(2)*3
factorial(3)*4
factorial(4)*5
factorial(5)*6

Se saca de la pila la expresión que está en la cima y se lleva a cabo la evaluación con valores conocidos. factorial(1)*2=1*2=2. Esto implica que factorial(2)=2

factorial(3)*4
factorial(4)*5
factorial(5)*6

Se saca nuevamente de la pila la expresión que está en la cima y se lleva a cabo la evaluación con valores conocidos. factorial(2)*3=2*3=6. Esto implica que factorial(3)=6

factorial(4)*5
factorial(5)*6

Se saca la expresión de la cima. factorial(3)*4=6*4=24. Esto implica que factorial(4)=24

factorial(5)*6

Extrayendo la expresión de la pila y realizando la evaluación:. factorial(4)*5=24*5=120. Esto significa que factorial(5)=120

Finalmente se extrae de la pila la última expresión y realiza la operación para encontrar el resultado final factorial(5)*6=120*5=600. Lo cual significa que el resultado buscado es: factorial(6)=600

Es obvio que si se compara la función que no es recursiva para encontrar el factorial, con la que si lo es, posiblemente se concluya que es más difícil de entender la función recursiva, pero dependiendo del problema que se trate, la programación de funciones recursivas permite crear algoritmos más compactos, rápidos y potentes, ya que aprovechan su propio código.

Ejemplo 2.22. Escribir un algoritmo que permita encontrar el número de Fibonacci que está en la posición n cuya serie es la siguiente: 1, 1, 2, 3, 5, 8, 13, 21,…. Los dos primeros elementos de la serie de Fibonacci son 1 y a partir del tercer elemento los demás se encuentran sumando los dos elementos anteriores a él, ejemplo: 1+1=2, 1+2=3, 2+3=5, etc. Escribir una función no recursiva y posteriormente obtener la función recursiva equivalente. Salidas típicas a este programa son:

Elemento: 6 Es = 8	Elemento: 3 Es = 2	Elemento: 10 Es = 55

```
inicio  /* Programa principal */
  entero n;
  imprimir "Elemento: ";
  leer n;
  imprimir "Es = " , fibonacci(n);
 fin

entero fibonacci(entero x)   /* función recursiva */
inicio
entero  c;
  si (x <= 1)
    c = 1;
  sino
    c = fibonacci(x-1) + fibonacci (x-2);
 retornar c
fin
```

```
entero fibonacci (entero x)     /* función no recursiva */
inicio
entero  elemento, a, b, c;
  si (x < = 1)
  c = 1;
  sino
   inicio
     a = 1;
     b = 1;
     c = a + b;
     elemento = 3;
     mientras (elemento < x)
       inicio
         a = b;
         b = c;
         c = a + b;
         elemento = elemento + 1;
       fin
     fin
  retornar c
fin
```

En el caso de la función recursiva. La computadora guarda cada uno de los valores indeterminados en la pila como se muestra a continuación:

Considerar que se quiere encontrar el elemento 6 de la serie de Fibonacci. Por lo tanto: n = 6.

Cuando se llama la función, x toma el valor de n, (x = 6). De tal manera que los elementos que entran a la pila para los correspondientes valores de x se muestran a continuación.

x	
2	= fibonacci(1) + fibonacci(0)
3	= fibonacci(2) + fibonacci(1)
4	= fibonacci(3) + fibonacci(2)
5	= fibonacci(4) + fibonacci(3)
6	= fibonacci(5) + fibonacci(4)

El elemento que se encuentra en la cima de la pila ya es posible determinarlo porque se cumple que: (x < = 1). De tal manera que sacando los elementos de la pila y realizando las evaluaciones correspondientes se tiene:

c = fibonacci(1) + fibonacci(0) = 1 + 1 = 2
c = fibonacci(2) + fibonacci(1) = 2 + 1 = 3
c = fibonacci(3) + fibonacci(2) = 3 + 2 = 5
c = fibonacci(4) + fibonacci(5) = 5 + 3 = 8

Finalmente la función regresa el valor obtenido.

La función no recursiva. Utiliza las variables **a** y **b**, mismas que se suman para encontrar el nuevo elemento **c** de la serie. En cada iteración se realizan intercambios de valores entre **a, b** y **c**, de tal manera que el valor de **b** lo toma **a** y el valor de **c** se le asigna a **b,** y se vuelve a llevar a cabo la suma de ellas para encontrar el valor de **c** (c = a + b). La variable **elemento** se usa para contar los elementos de la serie de Fibonacci.

Es posible observar que los algoritmos recursivos por lo general son más compactos salvo algunas excepciones como sucedió con el factorial, de tal manera que la decisión de cuál utilizar depende del programador.

Al momento de codificar el programa solamente debe elegirse una de las dos funciones, la recursiva o la no recursiva dado que el resultado obtenido es el mismo aunque la codificación de las funciones sean diferentes.

2.4.3 Procedimientos

Procedimientos. Cuando una función no regresa valor alguno se llama procedimiento. Se puede decir que un procedimiento es un grupo de instrucciones para llevar a cabo alguna actividad. Los procedimientos también pueden requerir (o no) parámetros en el momento de la ejecución. A diferencia de las funciones que cuando se hace el llamado son parte de una instrucción, los procedimientos pueden ocupar una sola línea al momento de llamarse para su ejecución. Los procedimientos tienen el siguiente formato general:

nombre(*tipo* p_1, *tipo* p_2,....*tipo* p_n)
inicio
tipo v_1,v_2;
tipo v_3;
– – – – –
Instrucciones del procedimiento;
– – – – –
fin

Donde:
nombre: Es el nombre del procedimiento
p_1,p_2,....p_n: Parámetros 1,2,....n
tipo: Tipo de dato (entero, real, cadena. caracter, booleano, flotante, etc.)
v_1,v_2,....,v_n: Variables locales

Argumentos por valor. El paso por valor consiste en mandar el valor de las variables al correspondiente argumento del procedimiento (o función). Realmente lo que se envía es una copia del valor de la variable. El valor manejado en los parámetros dentro del procedimiento es local, de tal manera que la modificación de los mismos no afecta el valor de las variables que están fuera de ese procedimiento.

Argumentos por referencia. Si un procedimiento tiene un argumento por referencia, dicho argumento no recibe una copia del valor sino la posición y valor en memoria de una variable. Por lo tanto; cualquier modificación del parámetro tiene efectos sobre la variable cuya localidad se le envió (argumentos). Se debe especificar claramente el paso por referencia de un argumento anteponiendo una palabra reservada. En esta unidad se le antepondrá la palabra "***ref***", pero dependiendo del lenguaje, la palabra reservada que se antepone es diferente, de tal manera que se hará la indicación que corresponda cuando se trabaje con el lenguaje de programación.

En el procedimiento siguiente **w** se está definiendo como un argumento por valor, y **x** es un argumento por referencia.

rojo(*entero* w, *entero ref* x)
inicio
entero a,b;
– – – – –
Instrucciones del procedimiento
– – – – –
fin

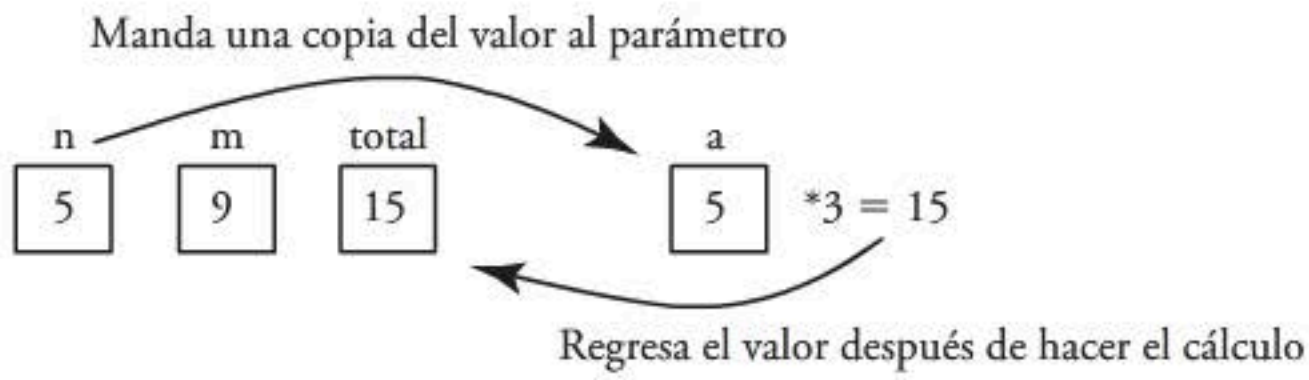
Si la llamada es **rojo(edad, altura)** para el procedimiento anterior, el valor de **edad** no sufre los cambios que se le realicen dentro del procedimiento a la variable **w**. Sin embargo la variable **altura** registra los cambios o modificaciones de la variable **x** porque no se está modificando el valor de **x** sino que se trabaja con el valor de la variable **altura**, aunque en el procedimiento se le haya dado el nombre de **x**. Ejemplo.

```
inicio /* Programa principal */
 entero total, n, m;
  imprimir "Valor de n:";
  leer n;
  imprimir "Valor de m:";
  leer m;
  total = verde(n);
  azul(m);
  imprimir "n = ",n, "m=",m, "total=",total;
  rojo(m,n);
  imprimir "n = ",n, "m=",m, "total=",total;
fin

entero verde(entero a)
 inicio
  a=a*3;
  retornar a;
 fin

rojo (entero w, entero ref x)
 inicio
  w=w*3;
  x=x*2;
 fin

azul(entero ref b)
 inicio
  b=b*3;
 fin
```

Una salida típica es:

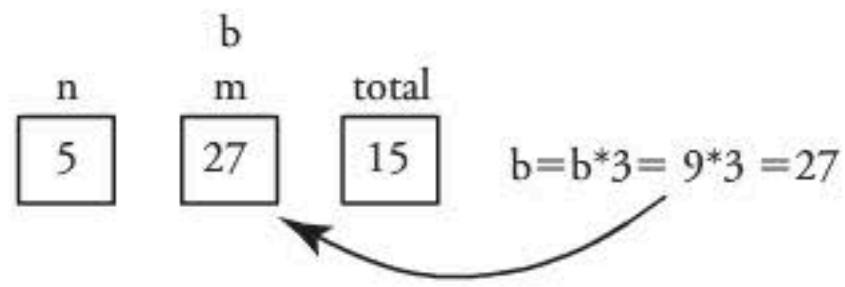
```
Valor de n: 5
Valor de m: 9
n = 5 m=27 total= 15
n = 5 m=54 total= 15
```

Suponiendo que los valores leídos de teclado son n = 5 y m = 9 como se muestra en la salida. Cuando se llama la función **verde(n)** se crea una copia del valor de la variable n y se le denomina "a". Después se realizan los cálculos correspondientes y el valor encontrado se lo asigna a la variable "total". Al salir de la función se destruye la localidad "a" ya que dicha variable es conocida solamente dentro de la función.



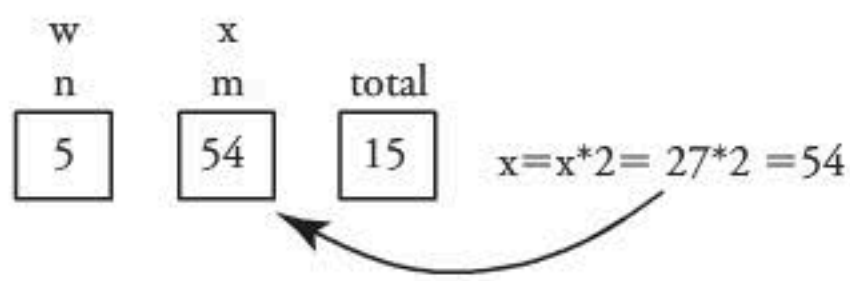
Cuando se llama el procedimiento **azul(m)**

Al argumento b se le manda la dirección y valor de m que en ese momento vale 9. Posteriormente se multiplica por 3 para encontrar 27, que es el nuevo valor de b y m. Cuando sale del procedimiento b se destruye y solamente queda m con la modificación que se hizo en b.



Cuando se llama el procedimiento **rojo(n,m)**

Al parámetro w se le manda el valor de n que en ese momento vale 5 y a x se le manda el valor y la dirección de m que en ese momento vale 27 Posteriormente w se multiplica por 3 para encontrar el 15 y x se multiplica por 2 para obtener 54 Cuando sale del procedimiento w y x se destruyen y solamente queda m con la modificación que se hizo en x y n con su valor original de 5.



El paso por referencia y por valor se utiliza tanto en funciones como en procedimientos y el uso de los mismos depende del problema y el programador ya que ambos son de utilidad.

Es importante mencionar que el paso por referencia no acepta constantes porque se espera la dirección de la variable y no el valor. Por ejemplo una llamada que marca error es.

```
azul(9);
```

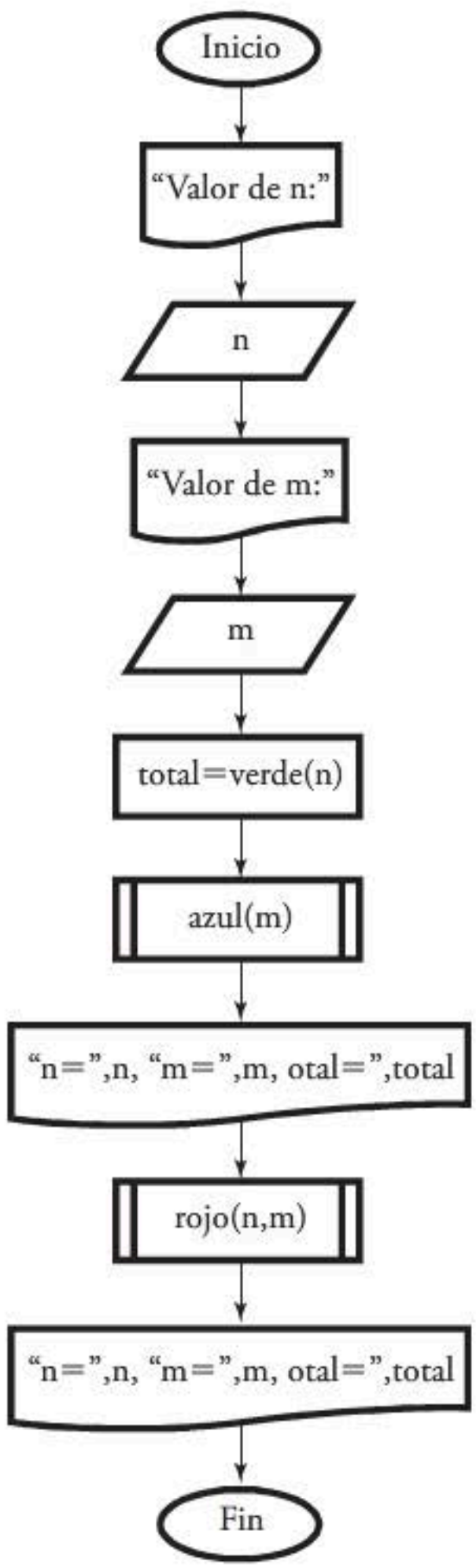
Sin embargo en el paso por valor si es válido:

```
total = verde(5);
```

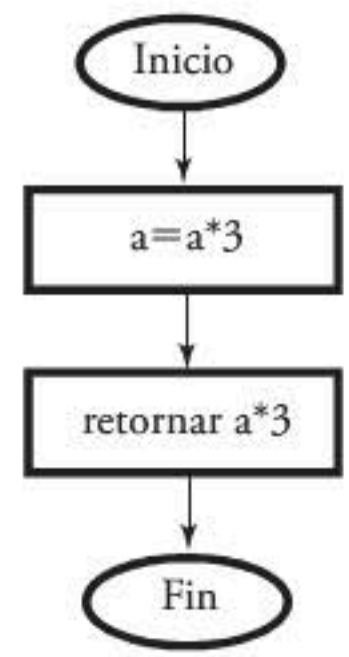
Es posible representar los procedimientos y funciones por medio de diagramas de flujo. Pero se debe considerar que cada procedimiento y cada función es un diagrama individual. El llamado de un procedimiento se representa por medio de un rectángulo con doble línea a los lados y el llamado de las funciones en los diagramas de flujo es parte de una instrucción, asignación, impresión o condición.

Es importante mencionar que en lenguajes como C++ y Java los procedimientos no existen, sino solamente las funciones que regresan un valor nulo (o no se regresa valor) y que en el caso de Java los procedimientos y las funciones se conocen como métodos, pero en su momento se harán las indicaciones correspondientes.

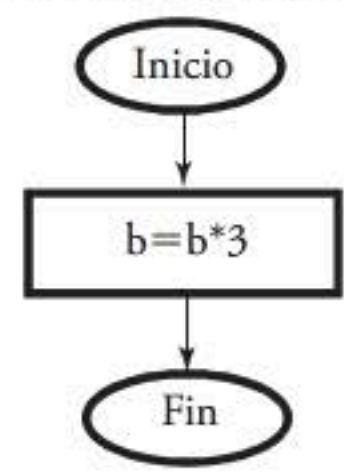
Programa principal



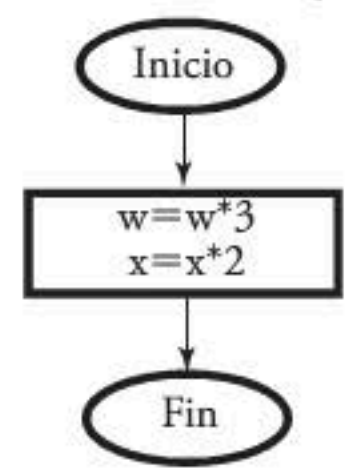
Función verde



Procedimiento azul



Procedimiento rojo



Ejemplo 2.23. Escribir un algoritmo que tenga dos procedimientos. En el primero de ellos; deberá leer el nombre de un producto, su precio y su descuento. En el segundo procedimiento deberá imprimir el nombre, precio y descuento leídos en el primer procedimiento y calcular e imprimir el precio real del producto considerando la información leída. Se deberá seguir leyendo nuevos datos hasta que se indique por medio de la letra 'n' ó 'N' que ya no debe continuar. El importe total se deberá incrementar con la suma de precios de nuevos productos. Una salida típica se muestra al lado:

```
inicio  /* programa principal */
   cadena   nombre;
   real precio, total;
   entero descuento;
   caracter op;

    total = 0;
    op ='s';
    mientras (op != 'n' or  op !='N')
     inicio
       lectura (nombre,  precio,  descuento);
       despliega (nombre, precio, descuento, total);
       imprimir "Continuar ¿s/n?:";
       leer op;
     fin
     imprimir "Total a pagar = $",total;
 fin

 lectura(cadena ref nombre, real ref  precio, entero ref descuento)
 inicio
    imprimir "Nombre: " ;
   leer nombre;
   imprimir "Precio: " ;
  leer precio;
  imprimir "Descuento: ";
  leer descuento;
 fin

despliega(cadena nom, real pre, entero des, real ref tot)
inicio
real precionuevo;
    precionuevo = pre – (des/100)*pre;
  imprimir nom, ", $",pre, "," ,des, "%, $ ",precionuevo;
  tot = tot + precionuevo;
fin
```

```
Nombre: Jabón
Precio: 5.30
Descuento (%): 20
Jabón, $5.30, 20%, $4.24
Continuar ¿s/n?: s
Nombre: Aceite
Precio: 12.56
Descuento (%): 25
Jabón, $12.56, 25%, $9.42
Continuar ¿s/n?: n
Total a pagar = $13.66
```

Explicación:

Cuando se ejecuta el programa, el programa principal crea las localidades de memoria para las variables nombre, precio, descuento, total y op. Para las primeras variables se tiene el lugar pero no se guarda ningún valor. A las últimas dos variables se les asigna el valor inicial 0 y 's' respectivamente, como se muestra a continuación:

nombre	precio	descuento	total	op
			0	s

Cuando se hace el llamado al procedimiento **lectura (nombre, precio, descuento)** se lee de teclado el nombre, precio y descuento del primer producto y se guardan en las localidades correspondientes creadas con anterioridad debido al paso por referencia, como se muestra en la siguiente figura:

nombre	precio	descuento	total	op
Jabón	5.30	20	0	s

Al llamar al procedimiento **despliega(nombre, precio, descuento, total)**, se crean las localidades para los parámetros: nom, pre, des y se les envía una copia de la información guardada en las variables nombre, precio y descuento. Para la variable total se crea la localidad tot que tomará su valor. Al momento de cambiar el valor de tot hará que cambie a la vez el valor de total debido al paso por referencia. De tal manera que las variables después de calcular el nuevo precio, imprimir información y actualizar el total tienen los siguientes resultados antes de salir del procedimiento.

nombre	precio	descuento	total	op
Jabón	5.30	20	4.24	s

nom	pre	des	tot	precionuevo
Jabón	5.30	20	4.24	4.24

Al momento de salir del procedimiento **despliega** se destruyen los parámetros nom, pre, des, tot, así como la variable precionuevo, debido a que son locales, quedando solamente las variables que no están tachadas:

nombre	precio	descuento	total	op
Jabón	5.30	20	4.24	s

nom	pre	des	tot	precionuevo
~~Jabón~~	~~5.30~~	~~20~~	~~4.24~~	~~4.24~~

Después pregunta si se desea continuar. Considerando que se indica que "si" tecleando la letra 's' o cualquier caracter diferente de 'n' o 'N' se leerán nuevos valores.

Antes de llamar el procedimiento: **lectura (nombre, precio, descuento)** la información que tienen las variables es:

nombre	precio	descuento	total	op
Jabón	5.30	20	4.24	s

Después de llamar el procedimiento: **lectura (nombre, precio, descuento)** y considerando que la información que se da por teclado es la que se muestra en la salida, las variables tienen la información:

nombre	precio	descuento	total	op
Aceite	12.56	25	4.24	s

Después de ejecutar el procedimiento: **despliega(nombre, precio, descuento, total)**, que calcula el precio del nuevo producto, imprime información y actualiza la variable total, los valores de las variables son:

nombre	precio	descuento	total	op
Aceite	12.56	25	13.66	s

Observar que el nombre, precio y descuento del producto va cambiando en cada una de las iteraciones y que la variable total va sumando los precios de cada uno de los productos después de realizar el descuento correspondiente.

Es posible seguir leyendo información de productos hasta que se lea del teclado la letra 'n' o 'N' para que al final imprima el precio total de los productos.

2.5 Resumen

Los programadores son personas creativas que utilizan la computadora para dar solución a todo tipo de problemas. En la solución del problema hay pasos que se deben realizar para obtener mejores resultados. La lista de pasos puede cambiar pero en general son los siguientes: análisis del problema, diseño del algoritmo, codificación del algoritmo, compilación, ejecución, verificación, depuración, mantenimiento y documentación.

En el análisis del problema se deben considerar los datos de entrada, el proceso de cálculo de la información y los resultados de salida. El diseño del algoritmo es una etapa importante en la solución del problema, debido a que son las instrucciones precisas y ordenadas para resolverlo. Existen varios métodos para la representación de algoritmos, entre los cuales se pueden mencionar: descripción narrada, diagramas de flujo, pseudocódigo y diagramas N−S. Cada una de estas representaciones algorítmicas tiene sus ventajas y desventajas con respecto a las demás. Los diagramas de flujo y los diagramas N−S buscan ilustrar de forma clara el algoritmo usando formas geométricas para distinguir cada una de las instrucciones, y lo logran en algoritmos pequeños, pero dejan de ser útiles a medida que los programas crecen.

Por su parte la descripción narrada presenta algoritmos con pasos muy generales y alejados de los lenguajes de programación ya que son bosquejos de la solución usando frases que la mayoría de las veces no tienen una traducción fácil en los lenguajes de programación, sin embargo cumplen con su cometido al describir de manera general la solución de problemas.

El pseudocódigo es una representación algorítmica que utiliza palabras en idiomas conocidos que son muy fáciles de codificar al momento de traducir el algoritmo al lenguaje de programación. Otra ventaja que tiene el pseudocódigo, es que ocupa poco espacio, propiedad que le permite representar algoritmos grandes y complejos, sin embargo su desventaja es que no son tan ilustrativos y fáciles de revisar como los son los diagramas de flujo que con un simple vistazo es posible saber su funcionamiento.

La codificación es la traducción del algoritmo a instrucciones de un lenguaje de programación que entiende y puede ejecutar la computadora. Por medio de la compilación la computadora se encarga de revisar que no existan errores en el programa apoyándose en el compilador, si existieran errores en el programa que se está editando, el compilador manda el mensaje correspondiente indicando la línea del programa que tiene el error. Es importante mencionar que el programa no se puede ejecutar hasta que esté libre de errores de compilación.

La ejecución, verificación y depuración permiten afinar el programa ejecutando, verificando y realizando los ajustes correspondientes hasta que el programa funcione perfectamente bien. Esta etapa permite revisar que el programa hace lo que se espera de él, con la presentación, rapidez y exactitud requeridas.

Finalmente el mantenimiento y documentación del programa, permitirán mantenerlo actualizado, realizar cambios que no se consideraron en el momento del análisis y diseño del problema porque no se requerían en ese momento, pero a medida que pasa el tiempo hay nuevos requerimientos y en lugar de hacer un nuevo programa es necesario aprovechar los buenos sistemas de programación actualizando o complementando algunas de las actividades que realiza. De tal manera que el mantenimiento permite tener funcionando al sistema y la documentación explica a los usuarios todas las actividades que puede realizar el sistema, pero también ayudan a entender el código y estructuración del programa con la finalidad de realizar actualizaciones sin afectar el funcionamiento correcto del sistema en general.

2.6 Problemas

1. Usando descripción narrada, escribir los algoritmos para llevar a cabo las actividades de cada uno de los siguientes incisos. Considerar que se tienen los elementos para realizar cada una de las actividades.

   a) Pintar un automóvil

   b) Preparar un café con leche.

   c) Enviar una fotografía y un mensaje por medio de Hotmail.

   d) Ingresar al cine para ver una película.

2. Usando descripción narrada, escribir los algoritmos para llevar a cabo las actividades de cada uno de los siguientes incisos.

   a) Surtir la despensa en tienda departamental.

   b) Dejar un mensaje a un amigo en Facebook.

   c) Preparar carne asada.

   d) Hacer una prenda de vestir para una persona.

3. Representar con notación algorítmica usando los símbolos aritméticos básicos y paréntesis para agrupar información cuando sea necesario, las expresiones matemáticas de cada uno de los siguientes incisos. Considerar a las letras como variables a excepción de "mod" que representa el módulo.

   a) $y = \dfrac{ac-5}{7b} - 9a$

   b) $y = b\left(\dfrac{a+7f}{5}\right)^3 + 4$

   c) $y = -5x^3 + 2x - 9$

   d) $y = \dfrac{\dfrac{2a-b^3}{7}}{(5e+3)c} + 5(b+8c^4)$

   e) $y = d(3c+e) - \dfrac{f-2a}{\dfrac{b+6}{7(4d-a)^2}}$

   f) $y = (3a-c)\bmod\left(\dfrac{b^3+7}{d+4c}\right) - ab$

   g) $y = \dfrac{\dfrac{c^4+3b}{(d+5a)\bmod 7}}{(b+8c)}$

   h) $= \left(5a^2 + \dfrac{4ab-c}{(d-3b)^3}\right)^7 + \dfrac{5}{(a+dc)b^2}$

4. Representar con notación algorítmica las expresiones matemáticas de cada uno de los siguientes incisos.

a) $y = (a - 5b)^3 + 3c - d$

b) $y = \dfrac{(4d + a^2)^3}{c - 5b} + 6a$

c) $y = (d + 5a)\left(\dfrac{d + \frac{3}{4}(a + 7b)}{\frac{-3c + d}{b^2}}\right)$

d) $y = \dfrac{3b - d}{(a + c^2)^3}(4a - 2c)$

e) $y = \left(\dfrac{d + 3c}{a^4}\right)(5d + 3)^7 + \dfrac{2}{5(a - b)}$

f) $y = (d + 3c)^{\frac{1}{3}}\left(\dfrac{4b + a}{5c}\right) + \dfrac{1}{7}$

g) $y = (3a + 2b - d^4)^5 \bmod(a + b) + \dfrac{4c}{(5d + 8)}$

h) $y = (cd + 8a)\left(\dfrac{5b - d}{4c} - 3a\right) + \dfrac{a^2}{3}$

5 Representar por medio de una expresión matemática cada una de las líneas de código de los incisos siguientes:

a) y = (a−b)/(3*(5*c+b))+4*(d+3* c)/(2*a);

b) y = 5*c^2−d*(a+b*c) ^(2/5);

c) y = (4+8*b)*((3*a)/4−9*c);

d) y = 5*c+3*b^2−(a+b/c) ^7;

e) y = b/((2*c−e) ^4/a−((5*c+6)/(4*d))/e;

f) y = ((a+3*e−7*c^2*b)*4)/(5*(b−c) ^2);

g) y = (3−d*e)/(4−5*a/d)+7/(a−b^2);

h) y = 8*a*b^2−(2*c+3)/(d+7) ^4+1/3;

6. Representar por medio de una expresión matemática cada una de las líneas de código de los incisos siguientes:

a) y = (4*c*d−8)/(3*b)+(5*(2*a+c) ^4)/(7−d);

b) y = ((c−7*a)/b)/e^5+(3/2)*((5*c−8)/(9−a^2));

c) y = (3*c*a−b^2)*(9/(5−4*c^2)+d);

d) y = ((7*d+4)/(3*a*b))/6−(a^3+d*c)/(b+4);

e) y = ((d+a*b)/4)/((5*c)/(3*a+c^3));

f) y = (b+8*a)/((c^5−1)/d)+(2*c)/(b+d);

g) y = (5*c+b^3−(a+8)/5)/(c+4*a+1/(2*b));

h) y = (((3+c*d) ^4)*(c−2))/(1/(5*a−8));

7. Para *a=1, b=−2, c=3, d=4*. Evaluar la expresión matemática representada en notación algorítmica de cada uno de los incisos. Realizar una operación en cada paso cuidando su jerarquía.

   a) *y = b+3*(c−b*(5+a^2)−3*d)+d;*

   b) *y = (a+c^3)/b+5*c+(c+(4*b−7) ^2);*

   c) *y = 5*d/(22−c^3)+a*b−(−32/(2*b−d)) ^2;*

   d) *y = (4*c−b*d/a+3+b^2)/c;*

8. Para *a=4, b=−1, c=−3, d=2*. Evaluar la expresión matemática representadas en notación algorítmica de cada uno de los incisos. Realizar una operación en cada paso cuidando su jerarquía.

   a) *y = ((4*a+d)/8−b^2/(c+a))/((d*(c^2−3*a))/b);*

   b) *y = (a*b−c)*d−d/b*c*(3*b^2−4);*

   c) *y = (((b*c−a) ^4+d)*7/(2*(a−c)))/(5*a+3*c);*

   d) *y = b+a*c/d−(b+c*d^2)*a;*

9. Sean *a, b, c, d* y x variables enteras cuyos valores son $a = -1$, $b = 7$, $c = 3$, $d = 5$. Encontrar el resultado de las expresiones matemáticas expresadas en lenguaje algorítmico de cada uno de los incisos.

   a) x = *(b * c− d mod a) mod (a – b);*

   b) x = *(((a − b) mod c) mod (c / a)) mod (b*c);*

   c) x = *((d+ a mod c)* b – (d mod a^3)) mod c;*

   d) x = *a mod d mod b – 3*c mod d ^ 2 + a/c mod b;*

10. Sean *a, b, c, d* y x variables enteras cuyos valores son: $a = 3$, $b = 2$, $c = -7$, $d = -5$. Encontrar el resultado de las expresiones matemáticas expresadas en lenguaje algorítmico de cada uno de los incisos.

    a) x = *(c mod a^3)/ (d * c− a/b mod a);*

    b) x = *d−c^2 mod (a*b) ^3−7*c mod b;*

    c) x = *b+3*(c mod 5 –b) ^d mod a+b^(4 mod d);*

    d) x = *a*(7*b/3 mod (c−5*a)) *(−d) – 3*a+d mod c^b;*

11. Si a = 2, b = −1, c = 4 y d = −3. Determinar cuál es el valor booleano 1=verdadero 0=falso, e indicar cuál es el mensaje que imprime la computadora, para cada uno de los siguientes incisos:

    a)
    ```
    si ((a>b) and (not((c <= d) or (d != a))))
        imprimir "Manzana";
      sino
        imprimir "Pera:", a*b
    ```

b) *si not*((a = = b) *or* (c<d) *and* (c >=a))
imprimir "Morelia ", (a+b);
sino
imprimir "Tlaxcala ", d;

c) *si* (a <d ***or not*** (c != b) *and* c>=a*3)
imprimir "Hola: ", 5*c;
sino
imprimir b," Adios";

d) *si* (c>=b*a *and* ((d != a) *or* (4*a>=c)) *and not* (b>c))
imprimir "Oaxaca= ", 3*d;
sino
imprimir "Michoacán= ", (c−d*a);

12. Si a = −2, b = 1, c = 3 y d = −5. Determinar cuál es el valor booleano 1=verdadero 0=falso, e indicar cuál es el mensaje que imprime la computadora, para cada uno de los siguientes incisos:

a) *si* ((b = = c) or not (*not*((a >d) and (c >= a))))
imprimir "Sandía: ",4*(a+b);
sino
imprimir "Fresa: ", a*c;

b) *si* ((a != d) and (c<d) or not(d >=b))
imprimir "Francisco: ", (5*b+d);
sino
imprimir "Alicia: ", 6*c;

c) *si* (b <a or not(3*c< a) or not(c != a) and d>=a)
imprimir c*d," Carlos: ";
sino
imprimir " José: ",(a−3*c);

d) si (d>=b*a and not((b != a) or (a <d*c)))
imprimir "Paola= ", 3*(d−2);
sino
imprimir "Marcelo= ", (5−d*a);

13. Escribir un algoritmo para leer dos números de teclado y determinar cuál de ellos es el mayor e indicar si es par o impar. Representar el algoritmo en pseudocódigo y diagrama de flujo.

A: −75 B: 3 Mayor es el impar: 3	A: 18 B: 13 Mayor es el par: 18	A: 24 B: 24 Mayor es el par: 24

14. Escribir un algoritmo para leer tres números de teclado e indicar cuál de ellos es el mayor. Representar dicho algoritmo en pseudocódigo y diagrama N−S.

A: 8 B: −53 C: 6 El mayor es: 8	A: 17 B: 49 C: 32 El mayor es: 49	A: 16 B: 16 C: 5 El mayor es: 16

15. Escribir un programa para leer números de teclado, determinar cuántos de ellos son divisibles entre 7 y encontrar el promedio de ellos. Terminar cuando el dato leído sea 99. Representar el algoritmo en pseudocódigo y diagrama N−S.

Dame datos
45
14
−49
99
Divisibles =2 , Promedio=−17.5

16. Algoritmo para leer tres números de teclado e imprimirlos en forma ascendente. Representar el algoritmo en pseudocódigo y diagrama de flujo.

A: 8 B: −53 C: 6 Ordenados: −53, 6,8	A: 17 B: 49 C: 32 Ordenados: 17, 32,49

17. Escribir un algoritmo para leer números enteros positivos de teclado y determinar cuál de los números leídos múltiplos de 7 es el mayor y cuál de los múltiplos de 3 es el menor. En caso de no leer algún número de los involucrados desplegar el mensaje "No hay múltiplos de 7" y/o "No hay múltiplos de 3" según proceda. Terminar cuando el número leído de teclado sea 99 sin considerarlo en la evaluación. Representar el algoritmo con pseudocódigo, diagrama de flujo y diagrama N−S.

Números: 21 10 42 14 51 99 Mayor múltiplo de 7 es = 42 Menor múltiplo de 3 es = 21	Números: 20 7 37 56 28 99 Mayor múltiplo de 7 es = 56 No hay múltiplos de 3

18. Algoritmo para leer el valor de N. Imprimir los primeros 10 números enteros inferiores a N, sumarlos e imprimir el resultado. Representar el algoritmo en pseudocódigo con los ciclos: *Mientras*, *Hacer-Mientras* y *Desde*.

```
N: 6
5
4
...
−4
Suma = 5
```

```
N: 73
72
71
...
63
Suma = 675
```

19. Algoritmo para calcular la suma de los cuadrados de los números enteros comprendidos entre m y n, sin incluirlos a ellos. Representar el algoritmo en pseudocódigo con los ciclos: *Mientras*, *Hacer−Mientras* y *Desde*.

```
m: 12
n: 21
Suma= 132
```

```
m: 5
n: -3
Suma= 7
```

```
m: 10
n:  11
Suma= 0
```

20. Una bacteria se reproduce al doble en cada minuto. Escribir un algoritmo para observar por medio de una tabla la reproducción de dicha bacteria en cierto tiempo. Representar el algoritmo en pseudocódigo con los ciclos: *Mientras*, *Hacer-Mientras* y *Desde*.

Tiempo (minutos): 60

Tiempo	Bacterias
0	1
1	2
2	4
...	.....
60	1529215046068500 00

21. Se dice que un número es perfecto si la suma de sus divisores sin considerarlo a él mismo, es igual al número inicialmente leído. Escribir un algoritmo para leer de teclado un número, desplegar sus divisores menores a él, sumarlos e indicar si se trata de un número perfecto. Representar el algoritmo en pseudocódigo con los ciclos: *Mientras*, *Hacer-Mientras* y *Desde*.

```
Número: 6
Divisores:
1+2+3 =6
El 6 es perfecto
```

```
Número: 8
Divisores:
1+2+4 =7
El 8 no es perfecto
```

22. Escribir un algoritmo en pseudocódigo para leer un número entero positivo y desplegar en la pantalla todos los elementos de la serie de Ulam. Para encontrar los elementos de la serie de Ulam se debe considerar:

- Comenzar con cualquier entero positivo
- Si es par; dividirlo entre 2; si es impar; multiplicarlo por 3 y sumar 1 al resultado.
- Obtener varios enteros positivos repitiendo este proceso, hasta llegar finalmente a 1.

```
Número: 26
Serie de Ulam:
26
13
40
20
10
5
16
8
4
2
1
```

```
Número: 160
Serie de Ulam:
160
80
40
20
10
5
16
8
4
2
1
```

23. Algoritmo para desplegar un menú de operaciones básicas (suma, resta, multiplicación, división y módulo), leer el código de la operación a realizar, leer dos números enteros, e imprimir el resultado en entero. Si el código de operación no existe, que lea otro código. Terminar cuando el código de operación sea 99. Representar el algoritmo usando para ello condicionales anidadas y condicionales de opción múltiple.

```
Operaciones
1.- Suma
2.- Resta
3.- Multiplicación
4.- División.
5.- Módulo
            99.- Salir
Operación: 1
a: 35
b:  17
Suma = 52
……
Operación: 4
a: 26
b:  3
División= 8
Operación: 99
```

24. Escribir un algoritmo en pseudocódigo, para leer un número entero positivo y posteriormente escribir su número equivalente en binario.

Cantidad (decimal): 26
Binario= 11010

Cantidad (decimal): 45
Binario= 101101

25. Escribir un algoritmo en pseudocódigo, para imprimir una tabla de equivalencias entre las unidades centímetros, pulgadas, pies y yardas. Se deberá leer el inicio, incremento y final en centímetros, considerando las siguientes equivalencias:

1 pul= 2.54 cms 1 pie = 12 pul 1 yarda = 3 pies

Inicio: 0
Incremento: 10
Final: 100

Centímetros	Pulgadas	Pies	Yardas
0	0	0	0
10	3.9370	0.3280	0.1093
20	7.8740	0.6561	0.2187
...	.....	.....	.....
100	39.3700	3.2808	1.0936

26. Escribir un algoritmo en pseudocódigo para leer una cantidad y el interés anual que se pagará en una institución bancaria. Desplegar en una tabla el año y la cantidad acumulada hasta que la cantidad sea mayor o igual al doble de la cantidad inicial.

Cantidad: 1000
Interés anual (%): 6

Año	Interés	Cantidad
0	0.00	1000.00
1	60.00	1060.00
2	63.60	1123.60
...	.....	.....

27. Escribir un algoritmo para leer las horas de trabajo a la semana de un trabajador y el pago por hora. Considerar que las horas normales de trabajo a la semana son 40 y que las horas extras que sobrepasan ese límite se deberán pagar un 60% adicional del pago de la hora de trabajo normal. Representar el algoritmo usando para ello una función para calcular el pago semanal y ese mismo algoritmo pero sin el uso de dicha función.

H. Trabajo: 47
Pago por H: 12
Pago de semana = $614.40

H. Trabajo: 50
Pago por H: 12
Pago de semana = $672.00

28. Algoritmo para leer una cantidad entera positiva de teclado e imprimir los dígitos que la integran por separado. Representar dicho algoritmo usando para ello:
    a) Una función para determinar el máximo divisor múltiplo de 10 que puede dividir a la cantidad leída.
    b) Usar un procedimiento para separar e imprimir los dígitos.

```
Cantidad: 30456
Dígitos:
3
0
4
5
6
```

```
Cantidad: 9330
Dígitos:
9
3
3
0
```

29. Algoritmo para leer una cantidad entera positiva de teclado e imprimir los dígitos de dicha cantidad al revés. Representar dicho algoritmo con dos procedimientos:
    a) Para leer la cantidad.
    b) Para separar e imprimir los dígitos.

```
Cantidad: 8307
Dígitos:
7
0
3
8
```

```
Cantidad: 576
Dígitos:
6
7
5
```

30. Escribir un algoritmo en pseudocódigo para leer el nombre de un alumno y tres calificaciones para cada alumno. Calcular el promedio de sus calificaciones, si el promedio es mayor o igual a 70 y no tiene materias reprobadas mandar el mensaje "Felicidades" su nombre "aprobaste con un promedio de:" el promedio que corresponda. En caso de tener un promedio mayor o igual a 70 pero tener materias reprobadas se deberá imprimir su nombre y el promedio e indicar que se tienen materias reprobadas. Finalmente si el promedio no es mayor o igual a 70 no se debe emitir mensaje alguno. Calcular también el promedio de todo el grupo, el porcentaje de aprobados y porcentaje de reprobados. La información se deberá leer de teclado y dejará de leer información cuando el nombre sea "Adios".

```
Nombre: Elena
Materia A: 90
Materia B: 86
Materia C: 89
Felicidades Elena aprobaste con promedio de: 88
Nombre: Juan
Materia A: 64
Materia B: 82
Materia C: 58
Nombre: Carlos
Materia A: 100
Materia B: 50
Materia C: 98
Carlos, tienes promedio de: 83 pero tienes
materias reprobadas
Nombre: Adiós
P.Grupo = 79,   Aprobados = 33%,   Reprobados = 67%
```

31. Escribir un algoritmo que utilice una función recursiva para sumar n términos de la siguiente sumatoria:

$$1 + 4 + 9 + 16 + \ldots + n^2$$

Términos: 3 Resultado = 14	Términos: 6 Resultado = 91

32. Escribir un algoritmo que utilice una función recursiva para aproximar el valor de e^x, sumando n términos de la siguiente sumatoria:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

Términos: 12 x: 4.53 e^x= 92.5234	Términos: 20 x: 7.2 e^x= 1339.3440

INTRODUCCIÓN A LA PROGRAMACIÓN

3

Si la depuración es el proceso de eliminar errores, entonces la programación debe ser el proceso de introducirlos.

Edsger Wybe Dijkstra

Competencia de la unidad

Conocer las características principales de un lenguaje de programación. Codificar algoritmos en un lenguaje de programación.

- Realizar un mapa conceptual sobre los tipos de *software* y los conceptos básicos de programación.
- Buscar y analizar información necesaria para instalar y configurar el compilador del lenguaje de programación a utilizar.
- Realizar cambios en expresiones lógicas y algebraicas de un programa modelo y analizar los resultados obtenidos.
- Mostrar al estudiante programas completos de menor a mayor grado de dificultad, y con base en cada una de las instrucciones que los componen enseñar la sintaxis del lenguaje.

Introducción a la programación

Contenido

3.1. Introducción.
3.2. Lenguaje de Programación Java.
3.3. Entornos de Desarrollo para Java.
3.4. Traducción de un Programa.
3.5. Ejecución de un Programa.
3.6. Estructura Básica de un Programa.
3.7. Elementos del Lenguaje Java.
3.8. Salida de Datos.
3.9. Lectura de Datos.
3.10. Conversión de Tipos de Datos.
3.11. Clase Math.

3.1 Introducción

En la actualidad, existen muchos lenguajes de programación, los cuales permiten crear aplicaciones muy interesantes como videojuegos, páginas Web, sistemas ERP (*Enterprise Resource Planning*, Planificación de Recursos Empresariales), aplicaciones de escritorio, aplicaciones para móviles, etc; uno de los lenguajes que presenta mayor versatilidad para crear este tipo de proyectos de *software* es Java.

Java es un lenguaje que tiene muchas ventajas frente a otros lenguajes de programación: es *open source* (código abierto), esto permite ver el código fuente y modificarlo; es gratuito, no es necesario pagar una licencia para utilizarlo; es portable, puede escribirse el código en un sistema operativo y ejecutarse en otro diferente; es simple, tiene estructuras fáciles de utilizar; además, es ideal para aprender a programar.

3.2 Lenguaje de programación Java

Java es un lenguaje de programación de alto nivel orientado a objetos que se creó con el patrocinio de Sun Microsystems en 1991 como parte del *Green Proyect* compuesto por trece personas y dirigido por James Goslin. El proyecto cumplió con el objetivo principal que era desarrollar un lenguaje basado en C++, al cual se le llamó inicialmente *Oak* debido a un roble que tenía Goslin a la vista desde su ventana en las oficinas de Sun; luego pasó a denominarse *Green* tras descubrir que *Oak* era ya una marca comercial registrada para adaptadores de tarjetas gráficas y finalmente se le cambió el nombre a Java.

Existen varias versiones que explican por qué se le llamó Java, pero la hipótesis que más fuerza tiene es que Java debe su nombre a un tipo de café disponible en una cafetería cercana a Sun Microsystems, de ahí que el ícono de Java sea una taza de café caliente (véase **Figura 3.1**).

Figura 3.1. Ícono de Java.

La promesa inicial de Gosling era crear un lenguaje que permitiera a los programadores escribir el código una vez y ejecutarlo donde fuera (en inglés *"Write Once, Run Anywhere"*), proporcionando un lenguaje independiente de plataforma y un entorno de ejecución ligero y gratuito para los sistemas operativos más populares.

3.2.1 Características del Lenguaje Java

Algunas de las características principales de Java se describen a continuación:

a) Lenguaje simple.

Java se diseñó para eliminar las complejidades de otros lenguajes como C y C++. Aunque Java posee una sintaxis similar a C, con el objeto de facilitar la migración de C hacia a Java, Java es semánticamente muy distinto a C:

Java no posee aritmética de apuntadores: El uso de apuntadores es el origen de muchos errores de programación que no se manifiestan durante la depuración y que una vez que el usuario los detecta pueden llegar a ser difíciles de resolver.

No se necesita liberar la memoria: Determinar el momento en el cual se debe liberar el espacio ocupado por un objeto es un problema difícil de resolver correctamente.. Java se auxilia del *Garbage Collector* (Recolector de Basura) para mejorar el rendimiento de la memoria.

No hay herencia múltiple: En C++ esta característica da origen a muchas situaciones en donde es difícil predecir cuál será el resultado. Por esta razón en Java se opta por herencia simple que es más fácil de aprender y dominar.

b) Orientado a objetos.

Java hace uso de un paradigma de programación que posee muchas virtudes: el orientado a objetos. Este paradigma usa los objetos en sus interacciones para diseñar aplicaciones y programas. Está basado en varias técnicas, incluyendo herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento.

c) Multiplataforma.

Los programas en Java pueden ejecutarse en Windows, Unix, Linux y Mac OS sin necesidad de hacer cambios.

d) OpenSource.

El código fuente del lenguaje Java puede ser visto por cualquier usuario y modificado.

c) Libre.

No es necesario pagar una licencia para programar en Java, su uso es libre y gratuito.

Desde sus inicios hasta ahora, Java ha experimentado numerosos cambios, así como un enorme incremento en el número de clases y paquetes que componen la biblioteca estándar, proporcionando nuevos elementos que ayudan a los programadores a construir mejores aplicaciones.

3.3 Entornos de desarrollo para Java

Un entorno de desarrollo es una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas.

Actualmente existen muchos entornos que permiten desarrollar aplicaciones en Java, entre los cuales se pueden mencionar: JCreator, Eclipse, JGrasp y NetBeans IDE.

Cada uno de los entornos anteriores presenta algunas ventajas y desventajas; sin embargo, se considera que NetBeans IDE es una buena opción para aprender a programar en Java porque aunque utiliza muchos recursos de la computadora es muy didáctico, permite identificar errores de sintaxis en el momento en el cual se está escribiendo un programa y no hasta después de compilarlo, esto se debe a que NetBeans IDE va interpretando cada instrucción escrita. De esta manera, en este libro se empleará NetBeans IDE para programar.

3.3.1 NetBeans IDE

NetBeans IDE es un entorno de desarrollo que nació en 1996 como un proyecto estudiantil (llamado *Xelfi*) en la Facultad de Matemáticas y Física en la Universidad Carolina en Praga. El proyecto atrajo

suficiente interés, por lo que los estudiantes, después de graduarse, decidieron convertirlo en un proyecto comercial, llamándolo: NetBeans.

En 1999, la empresa Sun Microsystems comenzó a interesarse en NetBeans al ser una buena herramienta para desarrollo en Java y decidió comprarlo; NetBeans se convirtió en el primer proyecto de código abierto patrocinado por ellos.

En el año 2009, Sun Microsystems fue adquirida por Oracle Corporation, pasando a su propiedad también NetBeans.

Actualmente NetBeans es un producto libre y gratuito sin restricciones de uso, que permite desarrollar en JAVA, C, C++, Groovy, PHP y Ruby.

Para instalar NetBeans IDE, véase el **Apéndice 1,** en el cual se incluye una explicación detallada de cómo debe instalarse el compilador (JDK) y después el entorno de desarrollo de NetBeans IDE.

3.4 Traducción de un programa

Los programas escritos en Java pasan a través de tres fases: edición, compilación y ejecución (véase **Figura 3.2**)

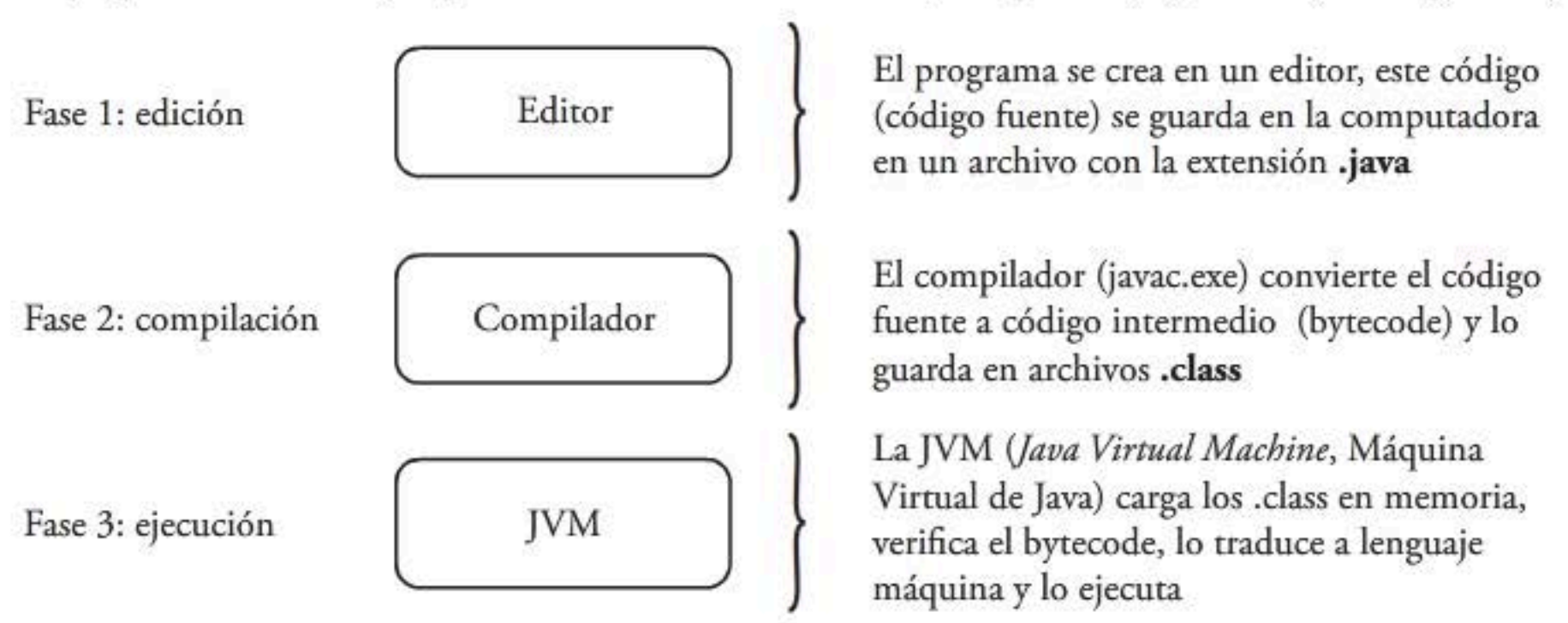


Figura 3.2. Fases de traducción en Java.

En el diagrama anterior se puede observar que la JVM es un elemento importante dentro del proceso de traducción porque permite ejecutar el código intermedio (bytecode) generado por el compilador de Java en cualquier plataforma o sistema operativo; esto brinda portabilidad al lenguaje porque se puede tener un programa .class escrito en Windows (por ejemplo) y ejecutarlo en otra plataforma como Linux.

3.5 Ejecución de un programa

Para crear y/o ejecutar un programa en Java utilizando NetBeans IDE es necesario dar doble clic sobre el ícono del programa (véase **Figura 3.3**)

Figura 3.3. Ícono de NetBeans IDE.

Se abrirá una ventana como la que se muestra en la **Figura 3.4**.

Figura 3.4. Pantalla de inicio de NetBeans IDE.

Para crear un nuevo proyecto:

En la ventana de inicio, es necesario dar clic en el menú *File* (Archivo), éste aparece en la parte superior izquierda y seleccionar la opción de *New Proyect* (Nuevo Proyecto) como se muestra en la **Figura 3.5**.

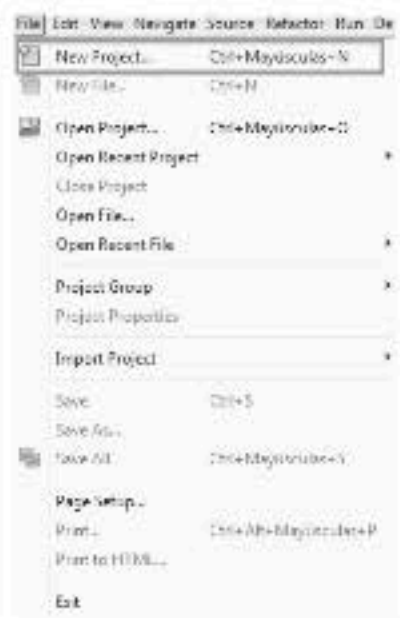


Figura 3.5. Crear nuevo Proyecto.

Después, aparecerá una ventana emergente como la que se muestra en la **Figura 3.6**, en la que se debe elegir el tipo de proyecto que se desea crear, para lo cual se seleccionará de la lista de *Categories* (Categorías) la opción de *Java* y de la lista de *Projects* (Proyectos) la opción de *Java Application* (Aplicación en Java) y se dará clic en el botón *Next* (Siguiente).

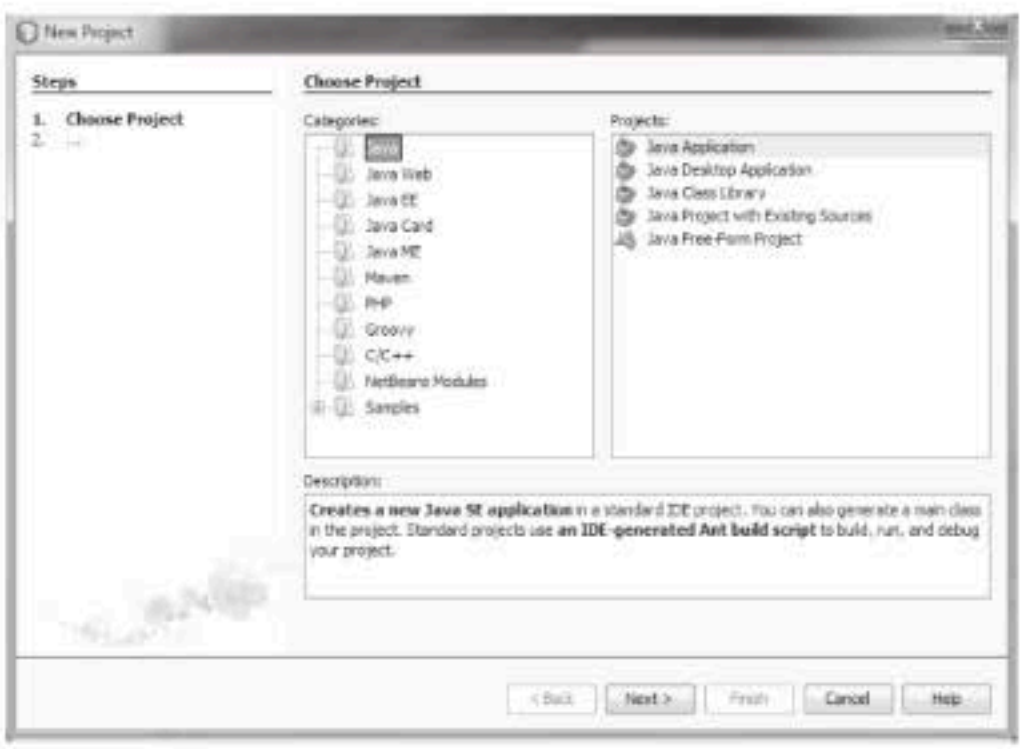


Figura 3.6. Seleccionar tipo de proyecto.

En seguida se mostrará otra ventana (véase Figura 3.7), en la cual se deberá indicar el Project Name (Nombre del Proyecto), el Project Location (Ubicación del Proyecto) y dar clic en Finish (Terminar).

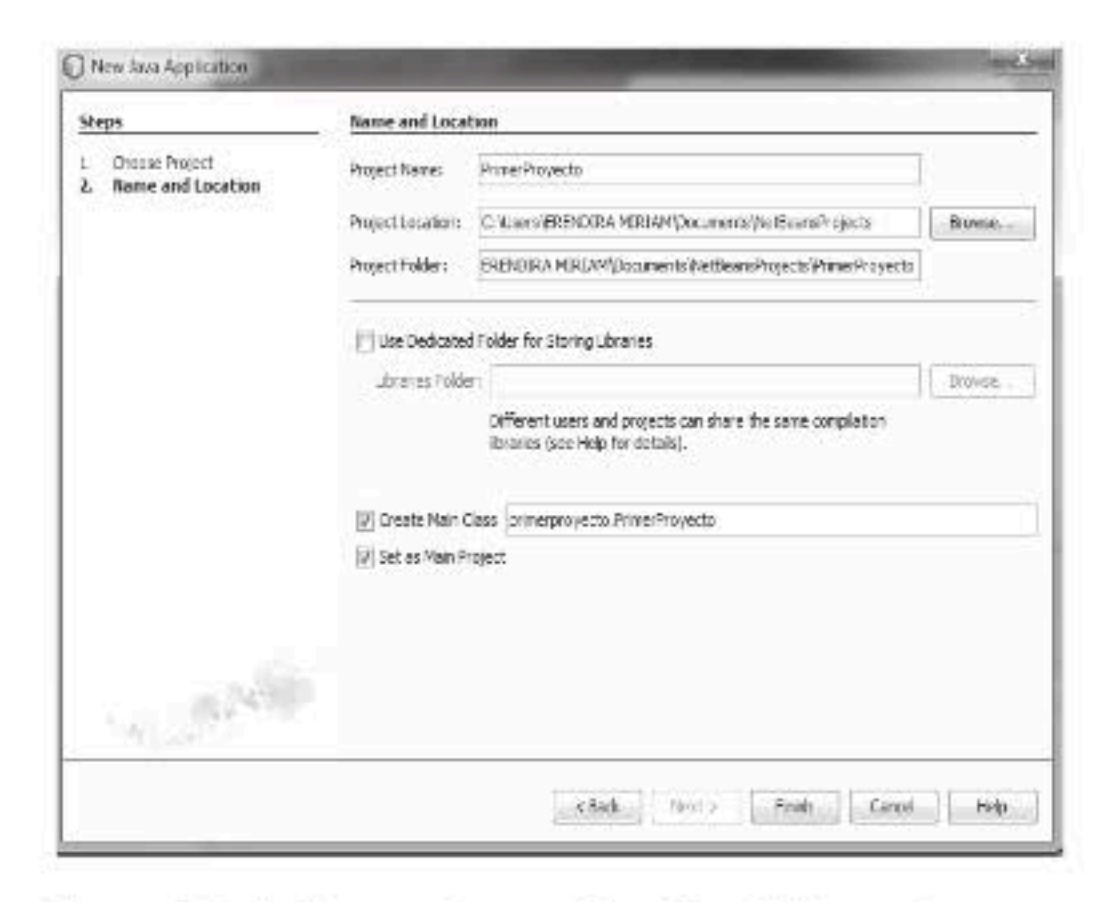


Figura 3.7. Indicar nombre y ubicación del Nuevo Proyecto.

Al oprimir el botón de Finalizar, en la pantalla de NetBeans IDE aparecerá un fólder del lado izquierdo en la sección de *Projects* (Proyectos) con el nombre del proyecto previamente asignado y del lado izquierdo aparecerá un archivo con extensión .java de manera automática (véase **Figura 3.8**) porque en la pantalla de la **Figura 3.7** se marcó la opción de *Create Main Class* (Crear Clase Principal).

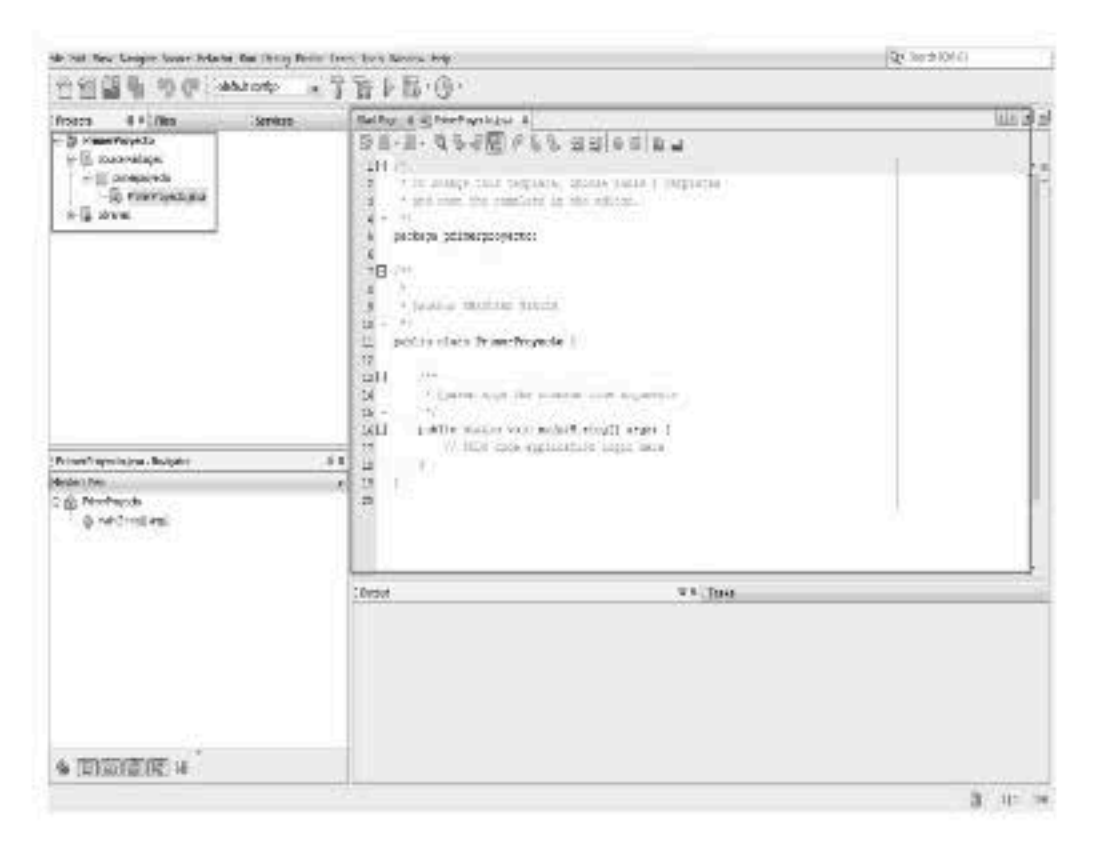


Figura 3.8. Proyecto Nuevo en NetBeans IDE.

Para comprobar el funcionamiento, a manera de ejemplo, se recomienda modificar la línea `//TODO code application logic here` que aparece automáticamente adentro de la instrucción **`public static void main`** `(String [] args)`:

```
public static void main (String [] args) {
      //TODO code application logic here
   }
```

Por la línea **System.out.println**("Hola mundo");tal como se muestra a continuación:

```
public static void main (String [] args) {
   System.out.println("Hola mundo");
}
```

Se deben conservar las demás líneas que aparecen, lo único que se modificará es la línea indicada, para después ejecutar el programa.

Para ejecutar un programa:

Es necesario seleccionar el archivo que se desea ejecutar con extensión .java (tiene un triángulo verde como ícono) que aparece en el fólder del proyecto abierto, para esto se da clic con el botón izquierdo del ratón sobre dicho archivo y luego clic con el botón derecho para que salga un menú emergente y seleccionar la opción *Run File* (Ejecutar Archivo) como se muestra en la **Figura 3.9**

Si el programa tiene salida y/o entrada de datos por consola (ver recuadro rojo en la siguiente Figura), ésta se modificará como en el ejemplo de la **Figura 3.10**.

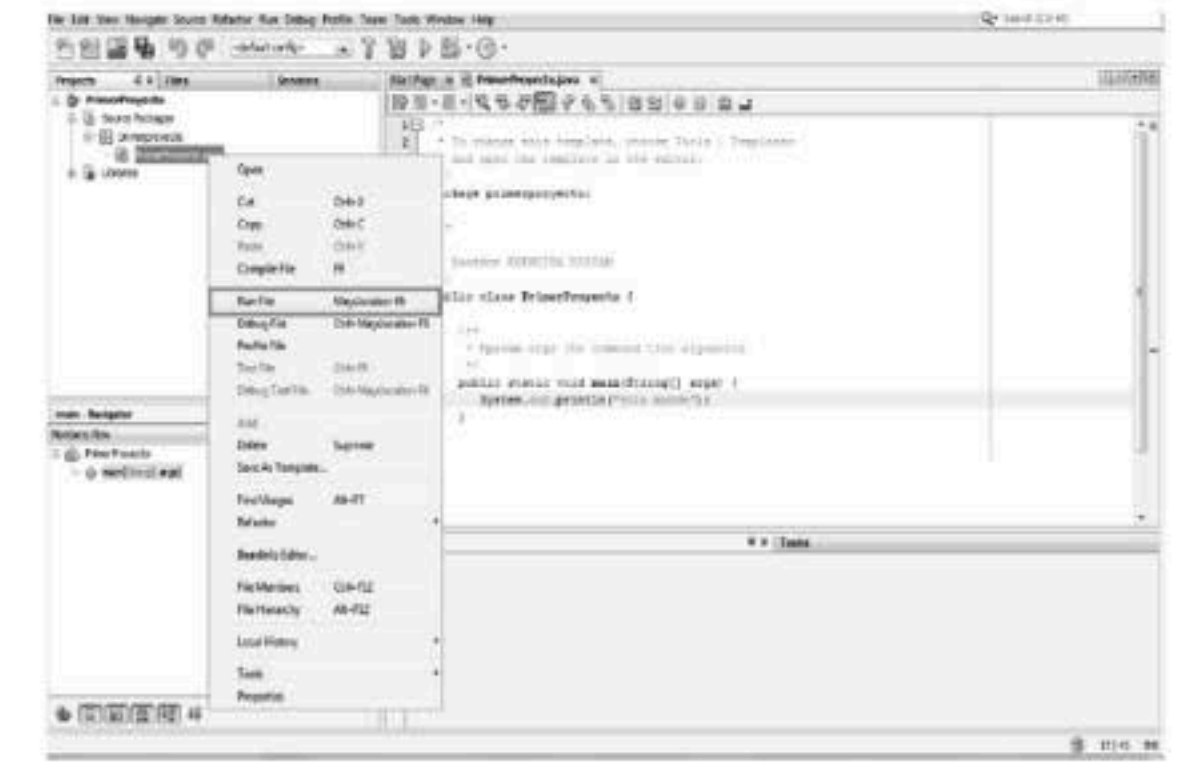


Figura 3.9. Ejecutar un programa en NetBeans IDE.

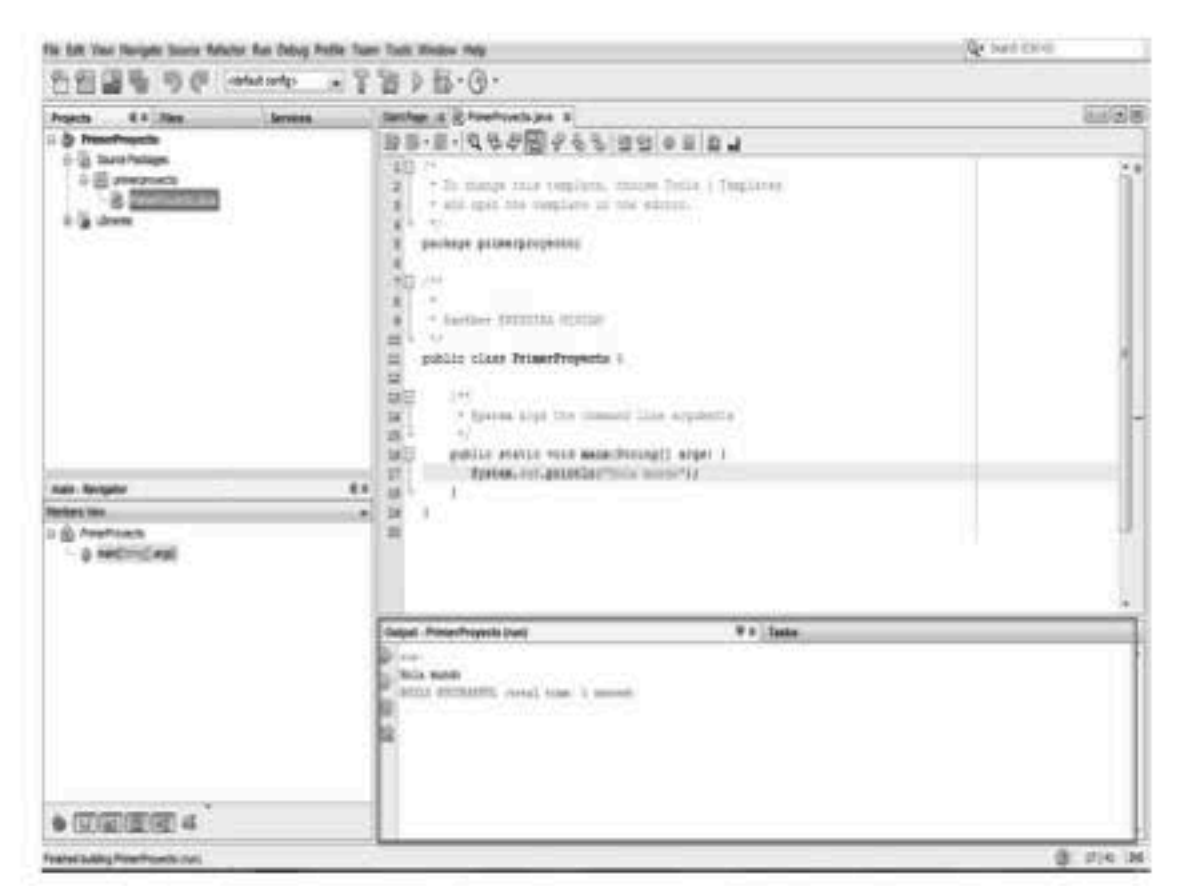


Figura 3.10. Consola de NetBeans IDE.

Para agregar un nuevo programa al proyecto:

En un mismo proyecto se pueden tener varias clases o programas en Java, para lo cual es necesario seleccionar el fólder de la sección *Projects* (Proyectos) que aparece del lado izquierdo de la ventana de NetBeans IDE y dar clic en el botón que aparece con un símbolo de + a su izquierda para que aparezca una carpeta llamada *Source Packages* (Paquetes Fuente) y luego una carpeta con un cubo amarillo con el nombre del paquete en el cual se dará clic con el botón izquierdo del ratón para seleccionarlo y después clic con el botón derecho para que aparezca un menú emergente. A continuación seleccionar *New (*Nuevo) y *Java Class* (Clase de Java) tal como se muestra en la Figura 3.11.

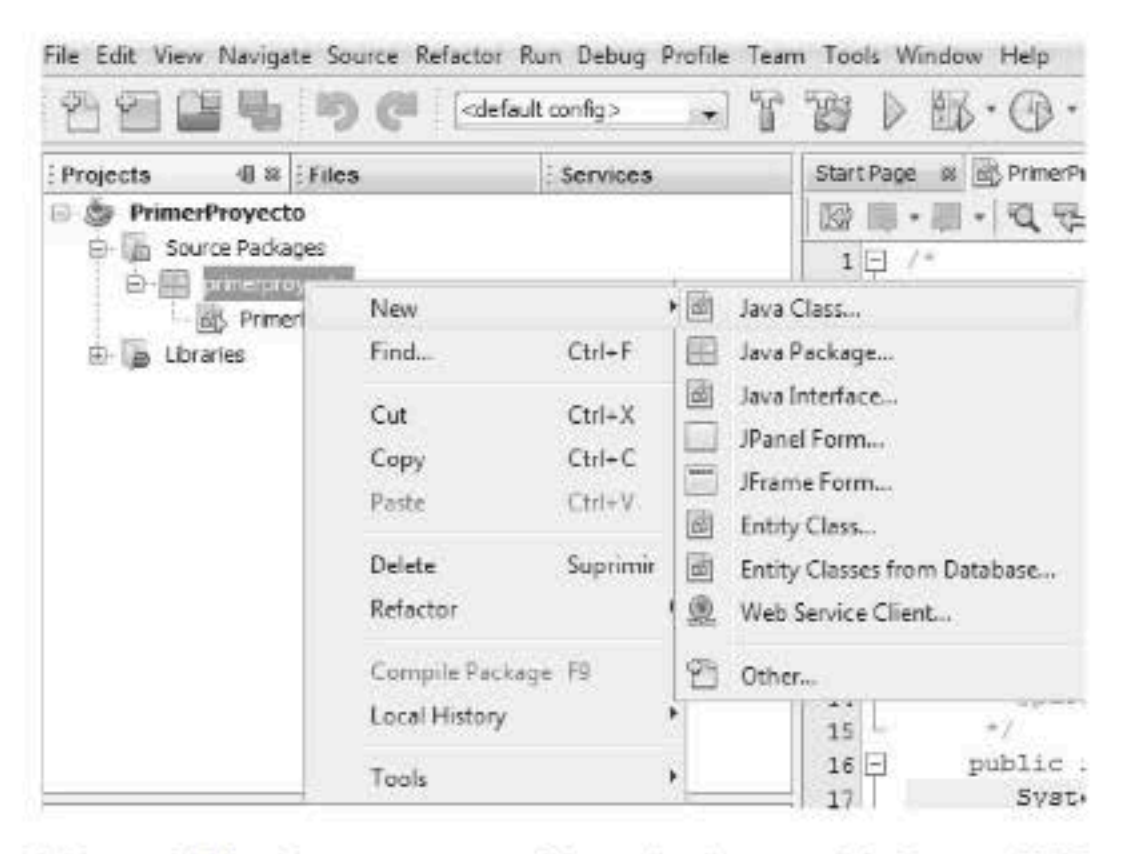


Figura 3.11. Agregar nueva Clase de Java en NetBeans IDE.

Después aparecerá una ventana como la que se muestra en la **Figura 3.12,** en la cual se debe introducir el nombre de la nueva clase en el campo de *Class Name* y dar clic en *Finish* (Terminar)

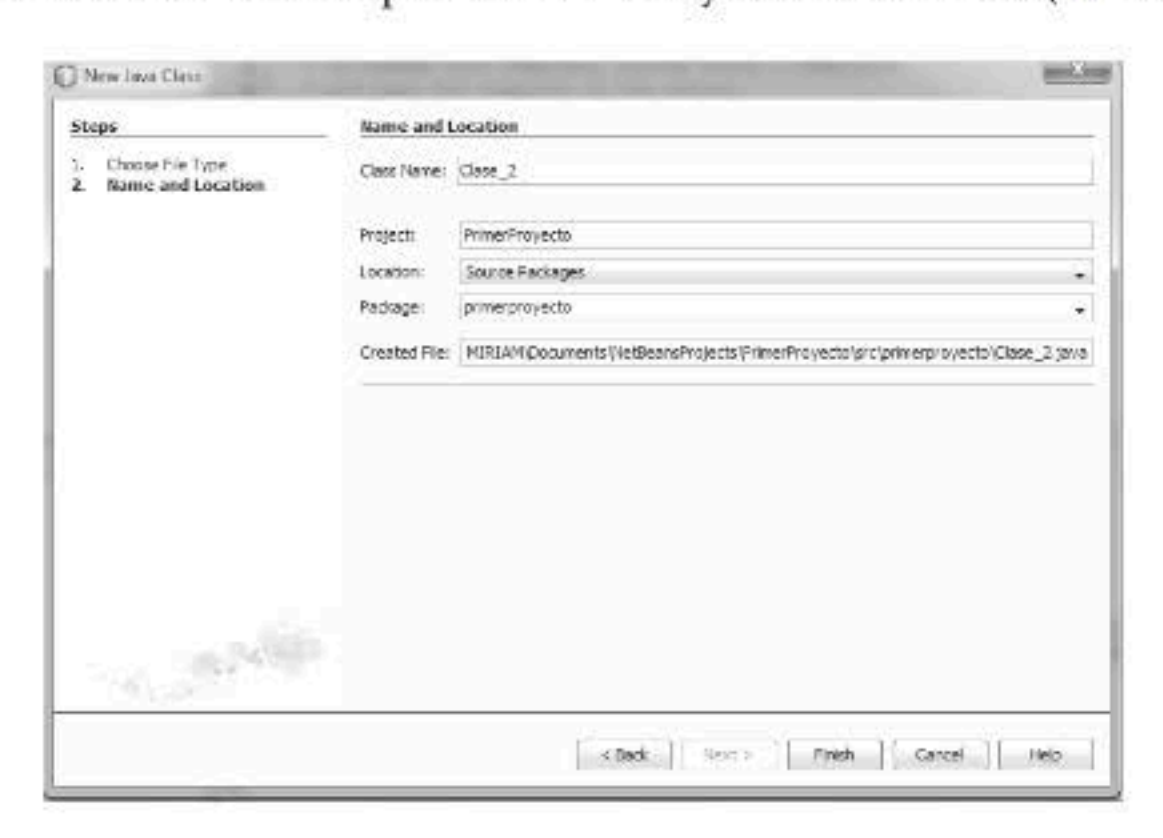


Figura 3.12. Nueva Clase de Java en NetBeans IDE.

Finalmente en la ventana de NetBeans IDE aparecerá del lado derecho en Projects (Proyectos) el nuevo archivo y del lado derecho la nueva clase (véase Figura 3.13), como en los recuadros rojos.

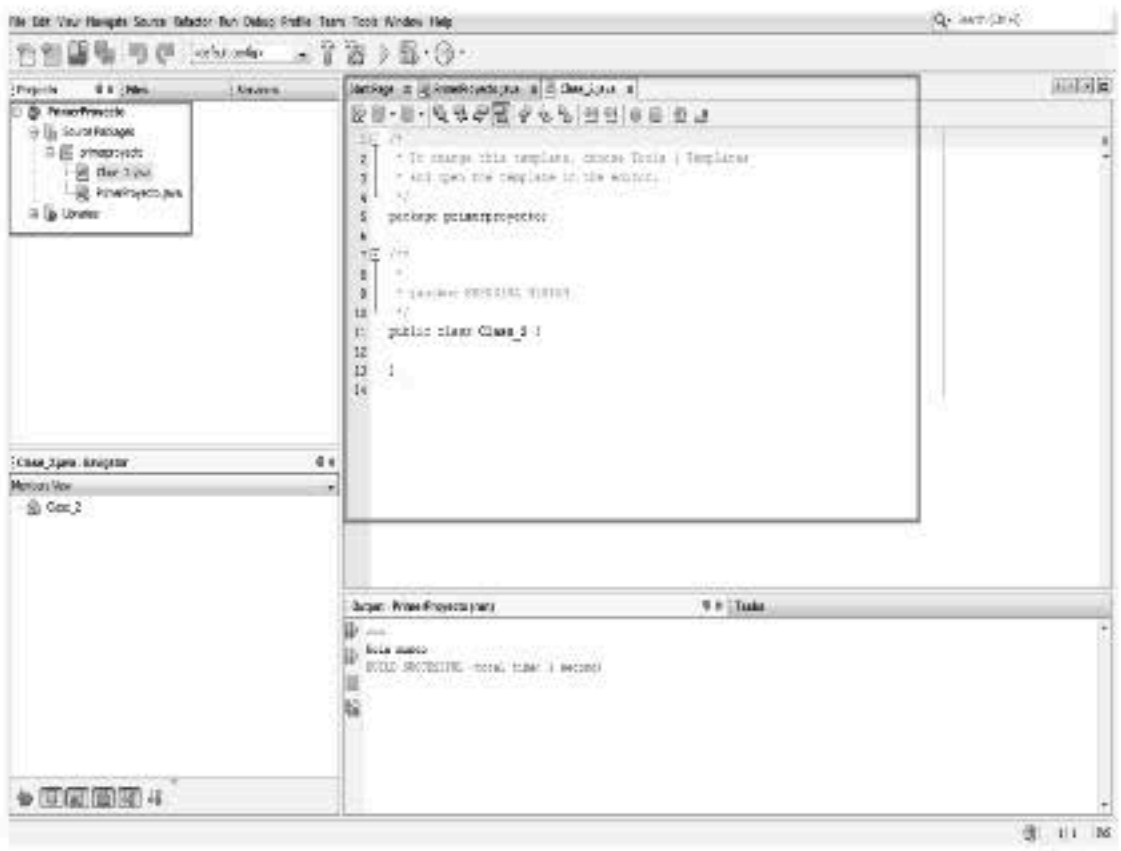


Figura 3.13. Visualización de la Nueva Clase.

3.6 Estructura básica de un programa

En Java la estructura básica de un programa es la siguiente:

```
public class Programa {
public static void main (String args []){
  instrucciones;
  instrucciones;
  }
}
```

La primera línea indica que la clase se llama `Programa` y que todo lo que esté dentro de la llave que abre "{" y que cierra "}" será parte de ella, de esta manera se puede observar que las llaves marcan el "inicio" y "fin" del programa.

Es importante mencionar que el nombre del archivo donde se guardará cierta clase deberá ser el mismo que el nombre de dicha clase; en el ejemplo anterior, la clase se llama `Programa` y el archivo en el que se guardará deberá llamarse `Programa.java`; como Java es sensible al uso de mayúsculas y minúsculas no es lo mismo "Ejemplo" que "ejemplo", por lo que se debe tomar en cuenta este aspecto al momento de programar.

La segunda línea del ejemplo corresponde a la instrucción de inicio de ejecución del programa, todo lo que esté dentro de las llaves que abren y cierran de la sentencia **`public static void main`** `(String args [])` será ejecutado por el compilador de Java.

En algunos casos es conveniente poner la instrucción **`package nombrePaquete;`** como primer línea en un programa, esta sentencia empaqueta o agrupa un conjunto de clases que tienen un objetivo en común; por lo cual, es opcional.

Finalmente es necesario mencionar que en Java se utiliza el símbolo `;` (punto y coma) para separar las instrucciones.

3.7 Elementos del lenguaje Java

3.7.1 Comentarios

Un comentario es un texto adicional que se añade al código para explicar su funcionalidad o para agregar una nota a manera de recordatorio o explicación. Los comentarios son ignorados por el compilador, por lo que no incrementan el tamaño del archivo ejecutable; de esta manera, se pueden añadir libremente al código para que pueda entenderse mejor.

Existen dos maneras de comentar en Java:

1. **Comentar una línea**, para comentar una línea se utiliza `//` y lo que vaya seguido de las diagonales es considerado por el compilador como comentario. Por ejemplo:

```
//Este es mi primer comentario en Java
```

2. **Comentar un bloque de líneas**, para comentar un bloque de texto, se encierra el comentario con la combinación de símbolos `/*` para iniciar el comentario y `*/` para finalizarlo. Por ejemplo:

```
/* Esta es la línea comentada número 1
   Esta es la línea comentada número 2
   Esta es la línea comentada número 3 */
```

3.7.2 Tipos primitivos de variables

Se llaman tipos primitivos de variables a aquellas variables que contienen los tipos de datos más habituales y que no requieren invocación para ser creados. Java tiene ocho tipos primitivos de variables, los cuales se describen en la **Tabla 3.1**.

Tabla 3.1. Tipos primitivos de variables en Java.

Nombre	Tipo	Espacio en memoria	Rango
`byte`	entero	1 byte	−128 y 127
`short`	entero	2 bytes	−32,768 y 32,767
`int`	entero	4 bytes	−2,147,483,648 y 2,147,483,647
`long`	entero	8 bytes	−9,223,372,036,854,775,808 y 9223372036854775807

(*Continúa*)

(*Continuación*)

Tabla 3.1. Tipos primitivos de variables en Java.

Nombre	Tipo	Espacio en memoria	Rango
`float`	decimal simple	4 bytes	−3.402823e38 a −1.401298e-45 y 1.401298e-45 a 3.402823e38
`double`	decimal doble	8 bytes	±1.79769313486232e±308
`char`	caracter simple	2 bytes	Código UNICODE (incluye ASCII)
`boolean`	valor true o false	1 byte	true o false

Las variables se deben definir de acuerdo al tipo de dato que se manejará con ellas. Por ejemplo si la variable `x` maneja cantidades enteras entre −10000 y 20000 no se puede definir dicha variable del tipo `byte`, porque las variables que se definen tipo `byte` solamente pueden manejar cantidades entre -128 y 127, pero tampoco tiene sentido definir a la variable `x` del tipo `int` o `long`, que aunque manejan también cantidades enteras, la cantidad requerida de bytes para ese tipo de datos es mayor, desperdiciando de esa manera memoria en una variable que no la requiere. De tal manera que se recomienda definir a la variable `x` de tipo entero corto (`short`) de la siguiente manera:

```
short x;
```

Es obvio que si `x` se hubiera definido como `int` o `long` también funciona perfectamente, porque estos tipos de dato permiten un rango mayor de información, en donde están incluidos los valores de `x` (-10000 y 20000), pero se recomienda definir las variables de acuerdo a la información que almacenarán. Lo que no es permitido es definir las variables de un tipo de dato que no pueda manejar la información requerida. Ejemplo: considerar que un programa tiene las siguientes líneas de código:

```
int w, x;        //Las dos variables se definen de tipo entero
w=2000000000;    //La variable w toma un valor válido en
                 //el rango de los enteros

x=5*w;           //La variable x rebasa el rango de los
                 //enteros porque 5*2000000000=10000000000
```

Se entiende que al definirse a las dos variables de tipo entero (`int w, x;`) sus valores no se pueden salir del rango de los enteros (-2,147,483,648 a 2,147,483,647). Aunque la variable `w` si tiene un valor en el rango de los enteros, la variable `x` se sale de él. Este error no lo detecta el compilador, porque no es un error de sintaxis, pero si causa problemas al momento de la ejecución del programa. Las líneas anteriores son solamente tres, pero en programas con muchas líneas de código, algunas veces los errores de mala definición de datos son difíciles de encontrar, ya que pueden arrojar resultados inesperados conocidos como basura o salirse del programa en forma inesperada y el programador no sabría la razón.

Es válido definir a las variables de un tipo de dato determinado y asignarles un valor al mismo tiempo que se realiza la declaración. A continuación se muestran algunos ejemplos de variables (`var_1`, `var_2`, `var_3`, `var_4`, `var_5`, `var_6`, `var_7` y `var_8`) declaradas con valores válidos

para los diferentes tipos de datos primitivos que existen en Java (**byte**, **short**, **int**, **long**, **float**, **double**, **char** y **boolean**):

```
byte var_1 = 127;
short var_2= -32768;
int var_3 = 2147483647;
long var_4 = 1206854758;
float var_5 = -8765.87f;
double var_6 = 8.9809938193018;
char var_7 = 'b';
boolean var_8 = true;
```

Es importante notar que para asignar valores de tipo **float** es necesario poner después del valor la letra `f` como en el caso de la variable **float** `var_5 = -8765.87f` sino marcará un error.

3.7.3 Cadenas de caracteres

Además de los ocho tipos primitivos de variables, las variables en Java pueden ser declaradas para guardar una instancia de una clase, como se verá en el capítulo de Introducción a la Programación Orientada a Objetos; de esta manera, en Java las cadenas son objetos de la clase `String` y se declara e inicializa de la siguiente manera:

```
String cadena = "cadena de ejemplo";
```

La variable se llama `cadena` y es de tipo **String**, además guarda el valor de `cadena de ejemplo.` Por lo general las variables tipo **String** se utilizan para manejar nombres de personas, calles, ciudades, animales, etc., ya que dicha cadena puede estar integrada por letras, números, espacios en blanco, entre otros caracteres.

3.7.4 Definición de variables

Ya se definieron anteriormente algunas variables, pero es importante mencionar que en Java es necesario definir todas las variables que se van a utilizar, para ello existen dos formas:

1. **Declarar las variables**. Para declarar una variable se debe definir primero su tipo y después su nombre, para lo cual se utiliza la siguiente sintaxis:

   tipo_variable nombre_variable;

   Un ejemplo de lo anterior es:

   ```
   int cont;
   ```

   Donde se declara una variable de tipo **int** (entero) llamada `cont`. También es posible declarar varias variables de un mismo tipo en una sola línea, para ello es necesario separarlas por comas, por ejemplo:

   ```
   int cont, num, prom;
   ```

   En este ejemplo se declararon tres variables de tipo entero que se llaman `cont`, `num` y `prom`.

2. **Declarar e inicializar las variables.** Para declarar una variable e inicializarla es necesario definir el tipo de la variable, seguida de su nombre y un signo de =(igual) para finalmente escribir el valor que guardará inicialmente la variable, la sintaxis es la siguiente:

   tipo_variable nombre_variable = valor_variable;

   Un ejemplo de lo anterior es:

```
float minimo = 23.5f;
```

Donde, se declara una variable de tipo **float** (decimal) llamada `minimo` y se le asigna inicialmente el valor de `23.5`. Al igual que cuando se declaran sólo variables, se pueden declarar e inicializar varias variables de un mismo tipo en una sola línea, para ello se separan con comas, tal como se muestra en el ejemplo siguiente:

```
float minimo = 23.5f, maximo = 56.78f, prom = 78.05f;
```

Cuando se definen los nombres de las variables se deben tomar en cuenta las siguientes reglas:

- El nombre debe ser único en el contexto del programa.
- Puede contener cualquier caracter UNICODE pero no puede comenzar con un número.
- No debe contener los símbolos que se utilicen como operadores (+,*,/,?, etc).
- Por convención los nombres de las variables comienzan con letra minúscula. Si un nombre consiste en más de una palabra, se escribirá sin espacios entre ellas y cada palabra (salvo la primera) comenzará con una letra mayúscula, por ejemplo: `estaBienEstaVariable`.
- Una palabra reservada por el lenguaje Java, no puede utilizarse como nombre de variable (véase **Tabla 3.2**).

Tabla 3.2. Palabras Reservadas en Java.

abstract	boolean	break	byte	case
catch	char	class	continue	default
do	double	else	extends	false
final	finally	float	for	goto
if	implements	import	instanceof	int
interface	long	native	new	null
package	private	protected	public	return
short	static	super	switch	synchronized
this	throw	throws	transient	true
try	void	volatile	while	

La tabla anterior contiene todas las palabras reservadas del lenguaje Java, lo cual implica que las variables, constantes, nombres de programas y nombres de métodos, no pueden tener el nombre de una de esas palabras reservadas, pero si cualquier otra palabra que respete las reglas mencionadas anteriormente.

3.7.5 Identificadores constantes

Las constantes son aquellas que una vez que se les asigna un valor, éste no puede ser modificado. Para definir constantes en Java sólo es necesario anteponer a la declaración la palabra **final**, por ejemplo:

```
final float PI = 3.1416f;
```

De esta manera, el identificador `PI` tendrá el valor constante de `3.1416` y no podrá modificarse. Por convención el nombre de las constantes se escribe en mayúsculas.

3.7.6 Operadores

Como su nombre lo indica, los operadores permiten realizar operaciones en el lenguaje de programación. En Java existen diferentes tipos de operadores, los cuales se describen a continuación.

Operadores aritméticos

Son operadores binarios (requieren siempre de dos operandos) que realizan las operaciones aritméticas habituales: suma (+), resta (-), multiplicación (*), división (/) y el módulo (%). En la **Tabla 3.3** se muestra una lista de los operadores aritméticos del lenguaje Java:

Tabla 3.3. Operadores aritméticos en Java.

Operador	Operación	Ejemplo	Significado de la operación
+	Suma	`c = a+b`	Suma el valor de `a` con `b` y asigna el resultado a `c`.
−	Resta	`c = a−b`	Al valor de `a` le resta el valor de `b` y el resultado de la operación se lo asigna a `c`.
*	Multiplicación	`c = a*b`	Multiplica `a` por `b` y asigna el resultado a `c`.
/	División	`c = a/b`	Divide el valor de la variable `a` entre el valor de `b` y asigna el resultado a la variable `c`.
/	Módulo residuo	`c = a % b`	El valor de `c` es el residuo de dividir `a` entre `b`. Ejemplos: 5 % 3 = 2; 13 % 8 = 5

Para agrupar términos en Java se emplean los paréntesis, por ejemplo:

```
a * (b - c)
```

En Java existen reglas que permiten realizar las operaciones en un orden correcto; es decir, existe una jerarquía de operadores, la cual se describe en la **Tabla 3.4**.

Tabla 3.4. Jerarquía de operadores en Java.

Operador	Operación
* / %	Los operadores de multiplicación, división y módulo son los que primero se evalúan; si hay varios de este tipo, se evalúan de izquierda a derecha.
+ −	Los operadores de suma y resta son los que se evalúan después; si hay varios de este tipo, se evalúan de izquierda a derecha.

Java no tiene un operador aritmético para la potenciación, en su lugar utiliza una función matemática para elevar una cantidad a una cierta potencia, que se citará posteriormente juntamente con las demás funciones matemáticas estándar de Java.

Operadores de asignación

La asignación consiste en trasferir un valor o expresión a una variable, para lo cual se utiliza el signo de igualdad "=" y su forma general es:

```
variable = valor o expresión
```

Ejemplo 3.1. Codificar en Java cada una de las siguientes expresiones matemáticas, usando paréntesis cuando sean requeridos para agrupar información. Considerar que cada una de las letras es una variable diferente a excepción de la palabra **mod** que representa la operación aritmética módulo.

a) $$y = \frac{(b-3a) \bmod c}{4-\frac{d}{c}} + 5d$$

b) $$y = (5a+7)d - \frac{c \bmod 3 - 6}{4b}$$

Respuestas:

a) y = ((b – 3 * a) % c) / (4 – d / c) + 5 * d;

a) y = (5 * a + 7) * d - (c % 3- 6)/(4 * b);

Ejemplo 3.2. ¿Cuál es el valor de la variable y? en cada una de las respuestas del ejemplo anterior, considerando que todas las variables se definen como enteras y con los valores que se muestran a continuación :

```
int  a=5, b=-4, c=2, d=3, y;
```

Respuestas:

```
a)  y = ((b-3*a)%c)/(4-d/c)+5*d;
    y = ((-4-3*5)%2)/(4-3/2)+5*3;  //Sustituyendo valores
    y = ((-4-15)%2)/(4-1)+15;
    y = (-19%2)/3+15;
    y = -1/3+15;
    y = 0+15;

a)  y = (5*a17)*d-(c%3-6)/(4*b);
    y = (5*517)*3-(2%3-6)/(4*-4);
    y = (2517)*3-(2-6)/(-16);
    y = 32*314/(-16);
    y = 9610;
    y = 96;
```

Es común tener líneas de código en donde se actualiza alguna de las variables

```
altura = 23.5;
contador = contador + 1;
```

En el caso de la segunda asignación puede emplearse una codificación que permite la compactación del código en expresiones simplificadas, en las que la variable que recibe el resultado participa a la vez como operando, las cuales pueden verse en la **Tabla 3.5.**

Tabla 3.5. Operadores simplificados en Java.

Operador	Operación	Ejemplo	Equivalente a	Significado	Si c = 13 el nuevo valor de c es
+=	Suma	c +=3	c = c + 3	Incrementa el valor de c en 3.	c = 16
−=	Resta	c -= 3	c = c − 3	Decrementa el valor de c en 3.	c = 10
*=	Multiplicación	c *= 3	c = c * 3	Multiplica el valor de c por 3	c = 39
/=	División	c /= 3	c = c / 3	Divide el valor de c entre 3	c = 4
%=	Módulo o residuo	c %=3	c = c % 3	El nuevo valor de c es el residuo de dividir c entre 3	c = 1

Las expresiones codificadas en Java a las que es posible aplicar la asignación compuesta tienen la siguiente forma:

```
variable   = variable operador expresión
```

Ejemplo 3.3. La codificación en Java de las líneas de código de cada uno de los incisos, usando para ello operadores de asignación compuestos es como se muestra a continuación:

	Expresión	Expresión compactada
a)	`y = y + 5`	`y +=5`
a)	`y = y − a/b`	`y −= a/b`
a)	`y = y *(3 −x/w)`	`y *= 3−x/w`
a)	`y = y / (7*x−3)`	`y /= 7*x−3`

También es común sumar o restar 1 a una variable; Java permite llevar a cabo también esta práctica tan frecuente de la siguiente manera:

```
variable + +
variable − −
```

Es decir:

Ejemplo	Descripción
`c++;`	Le suma un 1 a la variable `c`
`c--;`	Le resta un 1 a la variable `c`

3.8 Salida de datos

Todos los programas requieren de imprimir información, ya sea para desplegar un mensaje en pantalla, indicándole al usuario la entrada de datos o bien para imprimir textos y números obtenidos de bases de datos o cálculos que se realizaron en el transcurso de la ejecución del programa. El lenguaje Java tiene varias instrucciones para imprimir información, pero en este capítulo se abordarán solamente algunas en modo consola.

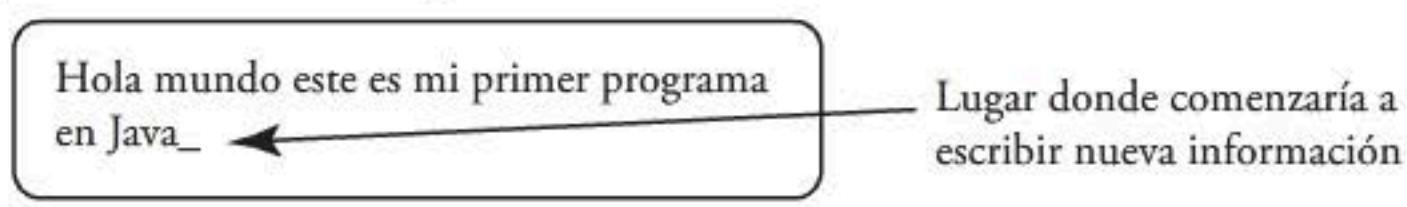
Los métodos más sencillos para desplegar información en java son: **`System.out.print( )`** y **`System.out.println( )`**. El hábil manejo de los parámetros en combinación con estas instrucciones, permite obtener mejores salidas.

Ejemplo 3.4. A continuación se muestra un programa que imprime dos líneas:

```
public class Hola_Mundo {
 public static void main (String args []){
 System.out.print("Hola mundo");
 System.out.println(" este es mi primer programa");
 System.out.print("en Java");
 }
}
```

Con la primera instrucción se imprime la información que se encuentra entre comillas y se queda el cursor en la misma línea, debido a que se desplegó la información con la instrucción **`print`**. Posteriormente imprime el texto entre comillas (este es mi primer programa) y el cursor salta a la siguiente línea, porque se desplegó la información con **`println`**. En general, si **`print`** termina con **`ln`** se imprime la información indicada y salta a la siguiente línea, sino termina con **`ln`**, se imprime la información y el cursor se queda en la misma línea.

El programa anterior tiene la siguiente salida en consola:



En lugar de imprimir la información anterior usando tres instrucciones, es posible obtener los mismos resultados con una sola instrucción usando "secuencias de escape" como se muestra:

```
System.out.print("Hola mundo este es mi primer programa\n en
Java");
```

Secuencia de escape. Es aquella que interrumpe la impresión lineal de la información para trasladar el cursor a una nueva posición de la pantalla y continuar a partir de ahí la impresión de la información restante, las secuencias de escape más comunes en Java son:

Tabla 3.6. Secuencias de Escape en Java.

Secuencia de escape	Descripción
`\n`	Salto de línea, coloca el cursor en la línea siguiente.
`\t`	Tabulador horizontal, coloca el cursor en la siguiente posición de tabulación.
`\r`	Retorno de carro, mueve el cursor al inicio de la línea actual; es decir, no avanza a la siguiente línea.
`\\`	Se emplea para imprimir la barra diagonal inversa.
`\"`	Se emplea para imprimir un carácter de doble comilla.

Otros ejemplos utilizando secuencias de escape son:

```
System.out.println("\"esto aparecerá entre comillas\"");
System.out.println("esto aparece en la primera línea\ny esto en la
segunda");
```

Es posible imprimir textos y valores usando el operador más (+) para concatenar textos y/o variables, como se muestra a continuación:

```
int a=3, b=7;
System.out.print (a + "×" + b + "=" + a*b);
```

$3 \times 7 = 21$

El método **String.format()** permite crear formatos especiales de impresión, de acuerdo a las necesidades del usuario y posteriormente imprimir dichos formatos usando las instrucciones: **System.out.println()** o **System.out.print( )**.

Ejemplo:

```
String nombre = "Juan", serie="Adios"
int edad = 18;
String serie=String.format("%s tiene %d años\n",nombre, edad);
System.out.println(serie);
```

Juan tiene 18 años
Adios

Al imprimir la información se sustituye el formato cadena %s por el nombre y el formato %d por la edad, posteriormente salta de línea e imprime la palabra "Adios".

El método **System.out.printf()** funciona de manera similar a la combinación de **String.format()** y **System.out.println()** usados en el ejemplo anterior, pero de una forma más compacta, ya que pasa directamente la cadena con los especificadores de formato para imprimirse. Ejemplo:

```
double ar = 43.542671;
int cant=13;
System.out.printf ("El valor del área  es = %.3f \n",ar);
System.out.printf("%d  en  decimal,  es  %x  en  hexadecimal
\n",cant,cant);
```

```
El valor del área es = 43.543
13 en decimal, es d en hexadecimal
```

Las instrucciones anteriores tienen los especificadores de formato: %.3f, %d y %x, los cuales tienen la finalidad de mostrar la información con ciertas características como se explica a continuación:

%.3f: Muestra el valor de la variable **ar** como una cantidad real (flotante) con 3 cifras decimales, en donde el último decimal se redondea.
%d: Muestra el valor de cant en forma decimal.
%x: Muestra el mismo valor de cant, pero ahora en hexadecimal.

Los especificadores de formato pueden tener los siguientes elementos:

%[posición$][indicador][ancho][.precisión]**especificador**

Los elementos entre corchetes son opcionales y cada uno de ellos permite ser más específicos en cuanto a la forma de visualizar la información en la pantalla. A continuación se tiene una explicación sencilla de cada uno de ellos.

[posición]. Indica el lugar en donde se desea dar el formato a un mismo dato. Los lugares se numeran de izquierda a derecha como 1$, 2$, etc.

[indicador]. Es el carácter que determina el formato de salida del dato. En la **Tabla 3.7** se tiene los indicadores más comunes usados en java.

Tabla 3.7. Indicadores de formato.

Indicador	Explicación
–	Alinea la información a la izquierda.
(	Encierra los números negativos entre paréntesis.
,	Muestra el separador decimal.
+	Muestra las cantidades positivas con signo.
0	Rellena de ceros los espacios sobrantes.

`[ancho]`. Indica el tamaño mínimo que debe ocupar el dato en pantalla.
`[.precisión]`. Indica el número de decimales que se imprimirán después del punto decimal.

Especificador. Carácter que indica la manera en que se debe formatear el dato. La siguiente tabla muestra una lista de algunos de los especificadores más utilizados en Java.

Tabla 3.8. Especificadores de formato.

Especificador	Explicación
d	Muestra la información como entero en decimal
f	Real (o flotante) con punto fijo.
e	Real usando notación científica.
g	Real, con opción a representarse en notación científica si la cantidad es muy grande o muy pequeña.
o	Entero en octal.
x	Entero en hexadecimal.
s	String o cadena
S	Cadena en mayúsculas
c	Carácter.
C	Carácter en mayúsculas.

En la **Tabla 3.9** se tienen algunos ejemplos usando especificadores de formato con una pequeña explicación de lo que se espera en la salida.

Tabla 3.9.

Bloque de instrucciones	Explicación
`double n=76.45781;` `System.out.printf ("%10.2f",n);`	El dato se imprime en 10 espacios con 2 decimales, redondeado y rellenando con blancos (b) los espacios que sobran bbbb76.46
`String nombre= "Pedro";` `System.out.printf ("%12s",nombre);`	Imprime la cadena en un espacio de 12 caracteres, justificado la información a la derecha y rellenando los espacios sobrantes con blancos. bbbbbbbPedro
`String nombre= "Pedro";` `System.out.printf("%-12s",nombre);`	Imprime la cadena en un espacio de 12 caracteres, justificado la información a la izquierda y rellenado los espacios sobrantes con blancos. Pedrobbbbbbbb

(*Continúa*)

(*Continuación*)

Tabla 3.9.	
Bloque de instrucciones	**Explicación**
`double w=17/3;` `System.out.printf("17/3= %8.3f",w);`	En un campo de 8 espacios, escribe el valor de w con tres decimales redondeados. En caso de ser necesario completa el campo con blancos. 17/3= ~~bbb~~5.667
`double w=13/4.0;` `System.out.printf("13/4=%08.3f",w);`	En un campo de 8 espacios, escribe el valor de w con tres decimales, en caso de ser necesario completa con ceros los espacios faltantes. 13/4= 0003.250
`double w=0.0000078924;` `System.out.printf("w = %9.2e",w);`	En un campo de 9 espacios, escribe el valor de w con dos decimales, en caso de ser necesario completa con blancos los espacios faltantes. w= ~~bb~~7.89e-6
`double w=-7.0/5.0;` `System.out.printf("w = %(8.2f",w);`	En un campo de 8 espacios, escribe el valor de w con dos decimales. Encierra el resultado entre paréntesis por ser negativo y rellena los espacios con blancos en caso de ser necesario. w= ~~bbbb~~ (1.40)

Ejemplo 3.5. En el siguiente programa con su respectivo salida es posible observar el uso de algunos especificadores de formato.

```
public class formas {
public static void main(String[] args){
double num = 357.8629;
String ciudad= "Morelia";
int a, b=3195;

System.out.printf("%d decimal es %o en octal y %x en hexadec\
n",b,b,b);
System.out.printf("%1$s minúsculas,  %1$2S mayúsculas\n",ciudad);
System.out.printf("Alineado a la izquierda: %-10s\n", ciudad);
```

lo que cambiaría de programa a programa es el tiempo de ejecución. A partir de ahora las salidas que estén encerradas por estos elementos, indicarán que es la salida fiel al correr el programa.

3.9 Lectura de datos

Todos los sistemas necesitan información, esa información puede venir de la pantalla, CD, base de datos, *mouse*, teclado, entre otras; pero seguramente cuando se trata de un programa la entrada es por medio de asignación o bien por medio del teclado usando para ello el modo consola.

Usando la clase **`java.util.Scanner`** es posible crear una instancia que permite la lectura de información en consola de la siguiente manera.

```
Scanner leer = new Scanner(System.in);
System.out.print ("Dame un número entero: ");
//la siguiente línea lee una cantidad entera y se la asigna
//a la variable num
int num = leer.nextInt();
```

Con las líneas anteriores primeramente se crea un nuevo objeto de la clase `Scanner` al cual se le llamó `leer` mismo que se puede aprovechar para leer cantidades enteras, flotantes, cadenas, booleanas, byte, double, etc; Poniendo el método indicado: `nextInt()`, `nextFloat()`, `next()`, `nextLine()`, `nextBoolean()`, `nextByte()` o `nextDouble()` respectivamente.

Ejemplo 3.6. El siguiente programa permite leer el nombre, edad y estatura de una persona y después imprime dicha información usando la instancia para leer información `leer`.

```
package lee_informacion;
import java.util.Scanner; //debe importarse la clase
public class lectura {
public static void main(String[] args){
Scanner leer = new Scanner(System.in);
String nombre;
int edad;
System.out.print("Cuál es tu nombre: ");
//lee una cadena y la asigna a la variable nombre
nombre=leer.next();

System.out.print("Cuál es tu edad: ");
//lee un entero y lo asigna a la variable edad
edad=leer.nextInt();

System.out.print("Qué estatura tienes en mts: ");
//la siguiente línea lee la altura y la asigna a la
//variable altura
```

```
float altura=leer.nextFloat ();
System.out.printf("%s tiene %d años de edad",nombre,edad);
System.out.printf(" y mide %.2f metros\n", altura);
```

```
run:
Cuál es tu nombre: Pedro
Cuál es tu edad: 18
Qué estatura tienes en mts: 1.73
Pedro tiene 18 años de edad y mide 1.73 metros
GENERACIÓN CORRECTA (total time: 13 seconds)
```

3.10 Conversión de tipo de datos

Cuando se programa es muy común tener que convertir de un tipo de dato a otro, en Java se pueden hacer las siguientes conversiones de tipos primitivos entre variables (véase **Tabla 3.10**).

Tabla 3.10. Conversiones de tipos primitivos válidos.

Tipo	Promoción válida
`double`	Ninguno.
`float`	`double`
`long`	`float, double`
`int`	`long, float, double`
`char`	`int, long, float, double`
`short`	`int, long, float, double`
`byte`	`short, int, long, float, doble`
`boolean`	Ninguno

Es posible realizar promociones no válidas (no incluidas en la tabla anterior), pero estas pueden producir errores de truncamiento o pérdida de información, como por ejemplo convertir de un **`float`** a un **`int`**.

Para realizar la conversión de los tipos primitivos es necesario anteponer entre paréntesis el tipo al cual se desea convertir una variable o valor:

```
(tipo) variable o valor
```

Ejemplos:

```
//El número entero 3 se convierte a double y se guarda en
//contador
double contador = (double) 3;
/* Se imprime el valor de 4 convertido a float; es decir: 4.0 */
```

```
System.out.println((float)4);
//El número 3.4 se convierte a int (se pierde la parte
//decimal) y se guarda en dato

int dato = (int) 3.4;

//Se imprime el valor de la variable dato; es decir: 3
System.out.println(dato);
```

Para convertir tipos primitivos a `String` es necesario utilizar el método `valueOf` de la clase `String`, la sintaxis es la siguiente:

`String.valueOf` `(variable o valor)`

Ejemplos:

```
/* se declara la variable peso y se inicializa con el valor double
de 3.56 */
double peso = 3.56;
/* se convierte a String el valor de peso y se guarda en cade-
na_peso */
String cadena_peso = String.valueOf(peso);
int x = 6;
String cadena = String.valueOf(x); //cadena = "6"
```

Si se desea convertir un `String` a tipo primitivo es necesario utilizar las clases y métodos de la **Tabla 3.11**.

Tabla 3.11. Clases y métodos para convertir de String a numérico

Convertir a	Clase y método
double	**Double.parseDouble** (cadena) ;
float	**Float.parseFloat** (cadena) ;
long	**Long.parseLong** (cadena) ;
int	**Integer.parseInt** (cadena) ;
char	**cadena.charAt**(posicion);
short	**Short.parseShort** (cadena) ;
byte	**Byte.parseByte** (cadena) ;
boolean	**Boolean.parseBoolean** (cadena) ;

Ejemplos:

```
String cadena = "3.4";
//covierte "3.4" a double y lo guarda en x
```

```
double x = Double.parseDouble(cadena);

//convierte 9 a boolean y lo guarda en val
boolean val = Boolean.parseBoolean("9");
/* imprime false, para que imprima true la cadena debe ser "true" */
System.out.println(val);

//en car guarda el caracter que está en la posición 0
char car = cadena.charAt(0);
//imprime 3
System.out.println(car);
```

Ejemplo 3.7. El siguiente programa permite leer un número de tipo **double** y después realiza algunas conversiones de tipos de datos.

```
import java.util.Scanner;

public class conversiones {
   public static void main (String args []){

//declaración de variables
double numero;
String cadena;
int entero;
float flotante;
char caracter;
short corto;
long largo;

//se crea un objeto de la clase Scanner
Scanner leer = new Scanner(System.in);

//se pide un número double, se lee y se guarda en número
System.out.print("Dame el número double: ");
numero = leer.nextDouble();

System.out.println("Conversiones:");
//se convierte número a int y se guarda en entero
entero = (int) numero;

//se convierte número a float y se guarda en flotante
flotante = (float) numero;

//se convierte número a char y se guarda en caracter
caracter = (char) entero;
```

```
//se convierte número a short y se guarda en corto
corto = (short) numero;

//se convierte número a long y se guarda en largo
largo = (long) numero;

//se imprimen los valores de las variables
System.out.println("int = "+entero);
System.out.println("float = "+flotante);
System.out.println("char = "+caracter);
System.out.println("short = "+corto);
System.out.println("long = "+largo);
 }
}
```

```
run:
Dame el número double: 87,9
Conversiones
int = −87
float = −87.9
char = 2
short = −87
long = −87
GENERACIÓN CORRECTA (total time: 23 seconds)
```

3.11 Clase Math

La clase Math proporciona una colección de métodos (véase **Tabla 3.12**) que permiten realizar cálculos matemáticos comunes. No es necesario importar la clase Math porque se importa automáticamente por el compilador.

Tabla 3.12. Métodos de la clase Math.

Método	Descripción	Tipos de valores (argumentos y retorno)	Ejemplo
`abs (n)`	Devuelve el valor absoluto de `n`	`double, float, int y long`	`Math.abs(-23.7)` es 23.7
`cos (n)`	Devuelve el valor del coseno de `n`	`double`	`Math.cos (8.97)` es -0.89835

(*Continúa*)

(*Continuación*)

Tabla 3.12. Métodos de la clase Math.

Método	Descripción	Tipos de valores (argumentos y retorno)	Ejemplo
`exp (n)`	Devuelve el valor de e^n	`double`	`Math.exp (3.78)` es 43.8160
`log (n)`	Devuelve el logaritmo natural de `n`	`double`	`Math.log (2.5)` es 0.912690
`max(n,m)`	Devuelve el valor más grande de `n` y `m`	`double, float, int y long`	`Math.max (-9.98,87.7)` es 87.7
`min(n,m)`	Devuelve el valor más pequeño de `n` y `m`	`double, float, int y long`	`Math.min(0,10)` es 0
`pow(n,m)`	Devuelve el valor de elevar `n` a la `m`	`double`	`Math.pow(-5.06,2)` es 25.6035
`random()`	Devuelve un número aleatorio entre 0 y 1	No recibe parámetros y regresa `double`	`Math.random()` es 0.583028798 o cualquier valor decimal entre 0 y 1
`sin (n)`	Devuelve el valor del seno de `n`	`double`	`Math.sin(0.87)` es 0.76432893
`sqrt (n)`	Devuelve el valor de la raíz cuadrada den	`double`	`Math.sqrt(9876)` es 99.378065
`tan (n)`	Devuelve el valor de la tangente de `n`	`double`	`Math.tan(-7.89)` es 27.751606

Para llamar un método se escribe la palabra `Math` seguida de un punto ".", además del nombre del método con los argumentos separados por comas que requiere el método entre paréntesis.

Ejemplos:

```
//eleva 3 a la potencia de 2; es decir, imprime 9.0
System.out.println(Math.pow (3, 2));

/* calcula la raíz cuadrada de 64; es decir guarda 8.0 en raíz */
double raiz = Math.sqrt (64);

/* calcula la raíz cúbica de 3; es decir guarda 4.0 en cubica */
double cubica = Math.pow (64,1/3.0);
```

En NetBeans IDE aparece una ventana desplegable que muestra los métodos de cierta clase, por lo que si se desea saber qué métodos tiene se debe escribir el nombre de la clase y poner el operador "." (punto) para que aparezca la ventana (véase **Figura 3.14**). En esa ventana se muestra del lado izquierdo el nombre del método y entre paréntesis el número y el tipo de argumentos que requiere; del lado derecho aparece el tipo de dato que devuelve cierto método.

De esta manera, no es necesario memorizar todos los métodos que posee una clase, ni cuántos argumentos recibe y qué tipo de dato recibe o regresa.

```
Math.
E                                          double
PI                                         double
IEEEremainder(double d, double d1)         double
abs(double d)                              double
abs(float f)                               float
abs(int i)                                 int
abs(long l)                                long
acos(double d)                             double
asin(double d)                             double
atan(double d)                             double
atan2(double d, double d1)                 double
cbrt(double d)                             double
ceil(double d)                             double
copySign(double d, double d1)              double
copySign(float f, float f1)                float
cos(double d)                              double
cosh(double d)                             double
```

Figura 3.14. Métodos de la clase Math.

Además NetBeans proporciona una descripción del método seleccionado, esto aparece en otra ventana (véase **Figura 3.15**).

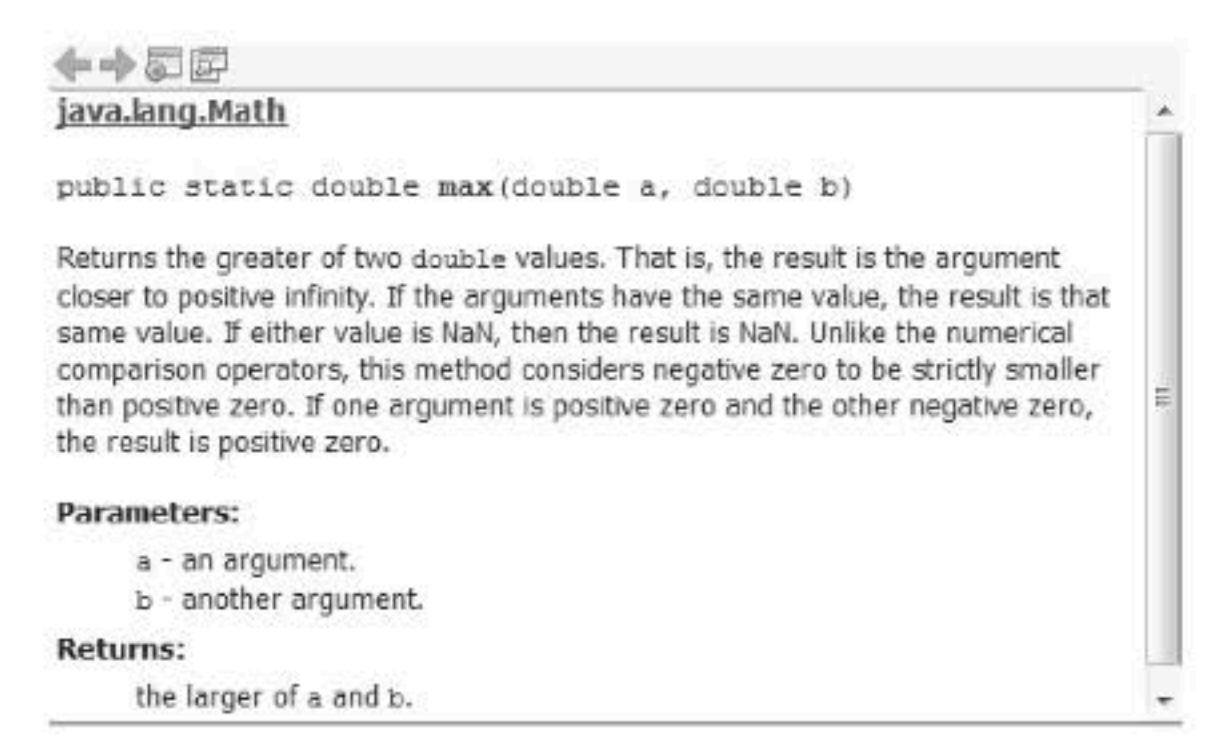


Figura 3.15. Descripción de los métodos de la clase Math.

Dentro de la clase Math están definidas dos constantes:

- `Math. PI.` Devuelve el valor de la proporción entre la circunferencia de un círculo y su diámetro, es 3.141592653589793
- `Math.E.` Devuelve el valor base para los logaritmos naturales, es 2.718281828459045

Ejemplos:

```
//imprime el valor de PI; es decir: 3.141592653589793
System.out.println(Math.PI);

//guarda en log_nat el valor de 2.718281828459045
double log_nat = Math.E;
```

Ejemplo 3.8. El siguiente programa permite leer dos números y calcula cuál es el mayor, el menor, la potenciación del primer número elevado al segundo, la raíz cuadrada del primero, la raíz cúbica del primero. Además genera un número aleatorio, imprime el valor de PI y de E.

```
import java.util.Scanner;
public class claseMath{
public static void main (String [] args){

//se crea el objeto leer
Scanner leer = new Scanner(System.in);
double primero, segundo; //se declaran dos variables double
System.out.println("\tPrograma que utiliza la clase Math");
System.out.print("Dame el primer número: ");
//se lee un double y se guarda en primero
primero = leer.nextDouble();

System.out.print("Dame el segundo número: ");
//lee otro double y se guarda en segundo
segundo = leer.nextDouble();

System.out.println ("Máximo= "+Math.max(primero, segundo));
System.out.println ("Mínimo= "+Math.min(primero, segundo));
System.out.println ("Potencia="+Math.pow(primero,segundo));
System.out.println ("Raíz cuadrada= "+Math.sqrt(primero));
System.out.println ("Raíz cúbica="+Math.pow(primero, 1/3.0));
System.out.println ("Número Aleatorio= "+Math.random());
System.out.println ("PI= "+Math.PI);
System.out.println ("E= "+Math.E);
 }
}
```

```
run:
Programa que utiliza la clase Math
Dame el primer número: 2.5
Dame el segundo número: 3
Máximo= 3.0
Mínimo= 2.5
Potencia= 15.625
Raíz cuadrada= 1.5811388300841898
Raíz cúbica= 1.0
Número Aleatorio= 0.5707071074560849
PI= 3.141592653589793
E= 2.718281828459045
GENERACIÓN CORRECTA (total time: 2 seconds)
```

Ejemplo 3.9. Codificar en Java cada una de las siguientes expresiones matemáticas, considerando que cada una de las letras es una variable y que la constante `PI = 3.1416`.

a) $y = 3x^2 - 4x + 8$

b) $y = \frac{4a-b}{7} + \sqrt[5]{6d-8}$

c) $y = \sqrt{\frac{5x^3 - w}{|7x-9|}} + \frac{(w-5x)^3}{48}$

d) $y = \frac{\text{sen}(30°) - x}{14 - x^2} + \sec(45°)$

Respuestas:

```
a) y = 3*Math.pow(x,2)-4*x + 18;
b) y = (4*a-b)/7+Math.pow(6*d-8,1/5.0);
c) y = Math.sqrt((5*Math.pow(x,3)-w)/Math.abs(7*x-9))+
       Math.pow(w-5*x,3)/48;
d) y = (Math.sin(30*PI/180)-x)/(14-Math.pow(x,2))+
       1/Math.cos(45*PI/180);
```

Se puede observar en el inciso (b) que la cantidad expresada en el radical se elevó a la `1/5.0` considerando que:

$$\sqrt[n]{x} = x^{\frac{1}{n}}$$

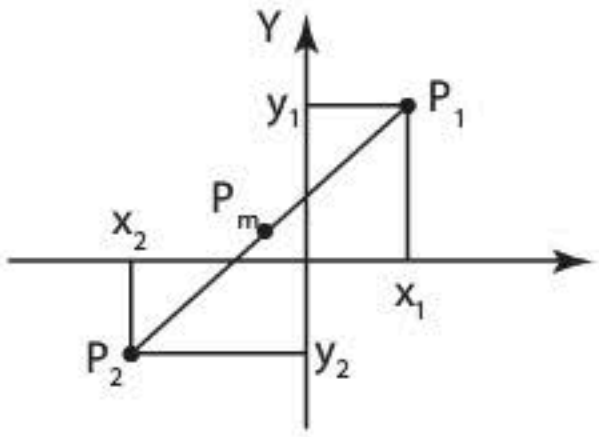
El `1/5.0` permite obtener raíz quinta con parte entera y decimal. En el inciso (d) se convierten los ángulos de 30º y 45º a radianes usando la equivalencia (π rad = 180º) considerando que los ángulos en Java deberán estar expresados en radianes. Se usó la constante PI=3.1416 que deberá ser definida en el lugar correspondiente de la siguiente manera: `final float PI = 3.1416;` aunque en lugar de PI se podría haber colocado directamente el valor de 3.1416. Ejemplo **y = tan 60º** se puede codificar de las siguientes formas:

```
//Siempre y cuando se defina PI=3.1416 como constante
y = Math.tan(60*PI/180);
```

```
//Sin necesidad de definir a PI como constante.
y= Math.tan(60*3.1416/180);
```

Por último, se usó la equivalencia trigonométrica: $\sec(x) = \frac{1}{\cos(x)}$ para encontrar la secante del ángulo de 45º, considerando que Java no tiene un método para encontrar directamente la secante de un ángulo.

Ejemplo 3.10. Las ecuaciones para calcular el punto medio (Pm), la distancia entre dos puntos (P_1P_2) y la pendiente (m) de una recta son:



$$Pm = \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2}\right) \qquad P_1P_2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$$m = \frac{y_2 + y_1}{x_2 + x_1}$$

Escribir un programa para leer las coordenadas de dos puntos de una recta y calcular el punto medio, la distancia entre dos puntos y la pendiente de esa recta:

Respuesta:

```
import java.util.Scanner;

public class partes_de_recta {
public static void main(String[] args){
 Scanner leer = new Scanner(System.in);

    double x1,y1,x2,y2,xm,ym,p1p2,m;

    System.out.println("Punto 1: ");
    System.out.print("x1: ");
    x1=leer.nextDouble();
    System.out.print("y1: ");
    y1=leer.nextDouble();

    System.out.println("Punto 2: ");
    System.out.print("x2: ");
```

```
    x2=leer.nextDouble();
    System.out.print("y2: ");
    y2=leer.nextDouble();

    //Coordenadas del punto medio
    xm=(x1+x2)/2.0;
    ym=(y1+y2)/2.0;
    System.out.printf("Punto medio (%.2f,%.2f)\n",xm,ym);

    //Distancia entre dos puntos
    p1p2=Math.sqrt(Math.pow(x2-x1,2)+Math.pow(y2-y1,2));
    System.out.printf("P1P2= %.4f\n",p1p2);

    //Pendiente
    m=(y2-y1)/(x2-x1);
    System.out.printf("m= %.4f\n",m);
}
}
```

```
run:
Punto 1:
x1: -3
y1: 4
Punto 2:
x2: 5
y2: 6
Punto medio (1.00,5.00)
P1P2= 8.2462
M= 0.2500
GENERACIÓN CORRECTA (total time: 1 minute 0 seconds
```

3.12 Errores en tiempo de ejecución

A continuación se describen algunas consideraciones para evitar errores comunes de programación:

- El nombre de la clase y el nombre del archivo deben tener el mismo nombre; es decir si el archivo se llama ejemplo.java la clase debe llamarse "ejemplo" para esto es necesario recordar que Java es sensible al uso de mayúsculas y minúsculas, por lo que no es lo mismo "Ejemplo" que "ejemplo".
- Dependiendo del idioma de NetBeans IDE, los números con decimales leídos de consola se deberán introducir con "," (coma) o "." (punto) para separar la parte entera de la decimal. Esto se debe a que en algunos países se emplea coma y en otros punto para separar la parte fraccionaria de un número.

Por ejemplo:

3.259

3,259

- Si se desea leer una línea después de un `nextDouble()`, `nextInt()` o `nextFloat()` es necesario el uso de dos `nextLine()`. El primero para capturar el retorno de carro (cursor) dejado por `nextDouble()`, `nextInt()`o `nextFloat()` y el segundo para leer la línea deseada.

Para gestionar errores de sintaxis, NetBeans IDE proporciona ayuda porque va interpretando cada una de las palabras que se escriben; de esta manera, si existe un error de sintaxis aparecerá del lado izquierdo de la línea que contiene el error, un ícono en color rojo como el que se muestra en la **Figura 3.16**, en la cual se describe el error si se posiciona el cursor encima de él.

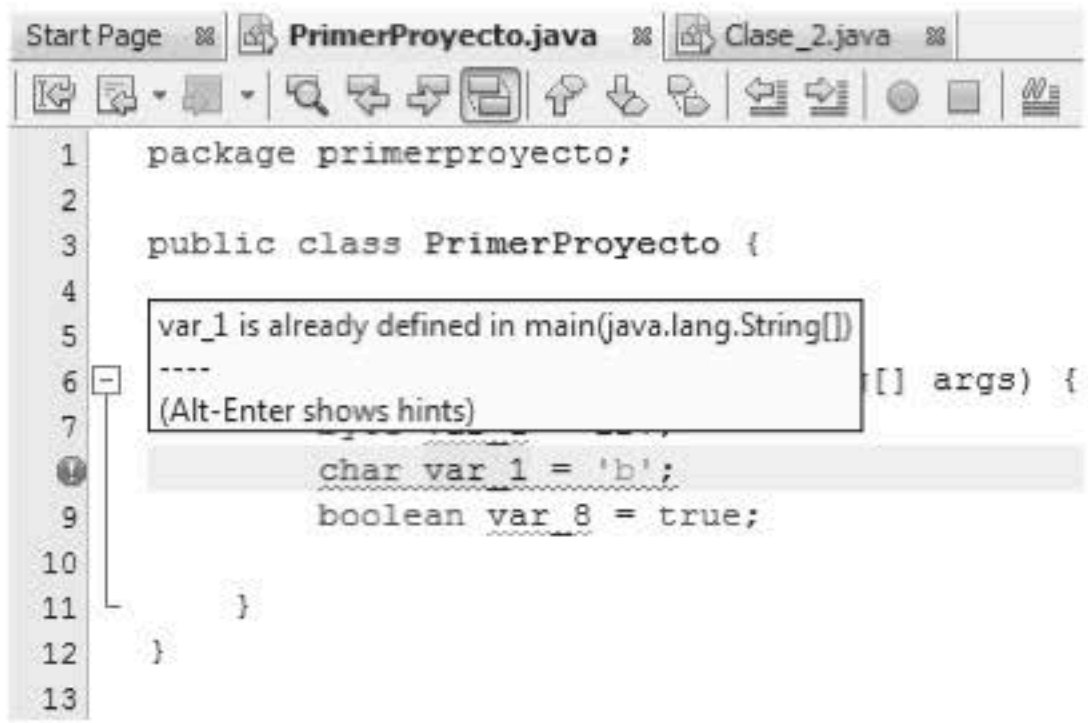


Figura 3.16. Errores de sintaxis identificados por NetBeans IDE.

En algunos casos, NetBeans IDE proporciona opciones para resolver un error, para esto aparecerá un ícono parecido a un "foquito" como el de la **Figura 3.17**, cuando se le da clic sobre él, aparecen posibles ideas para solucionar el problema para finalmente seleccionar (si así se desea) una de ellas para que NetBeans IDE la realice.

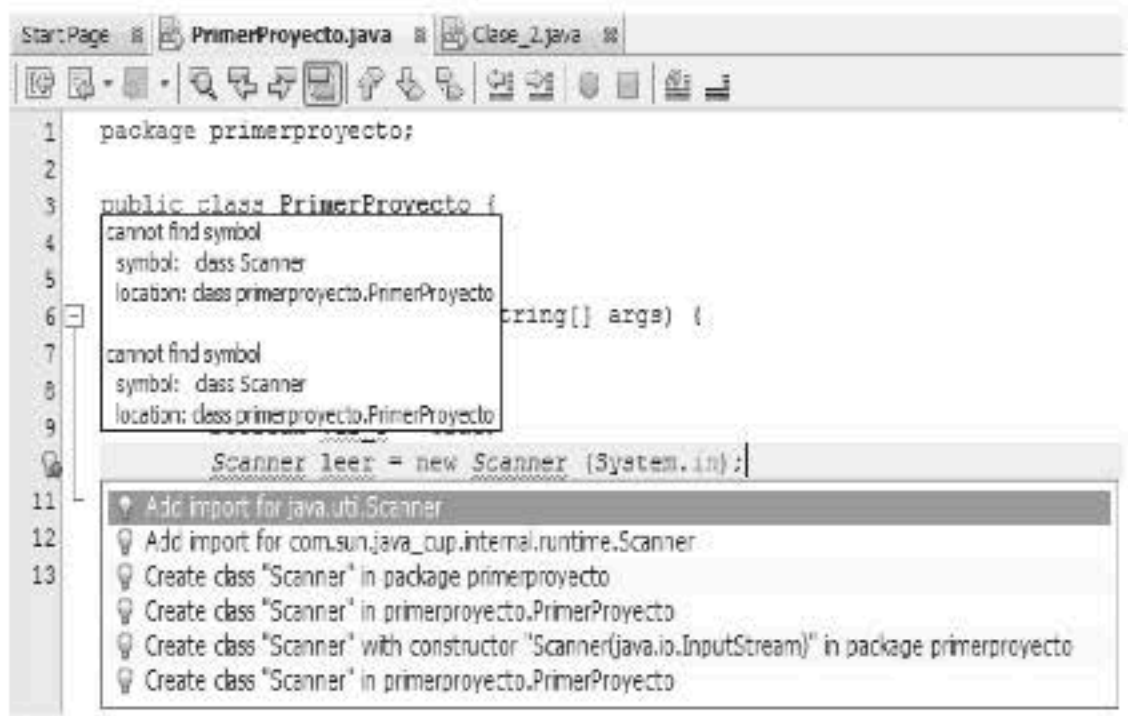


Figura 3.17. Propuestas de soluciones a errores de sintaxis de NetBeans IDE.

Java incorpora la gestión de excepciones, las excepciones indican que existe un problema o situación inesperada durante la ejecución de un programa. En muchos casos, el manejo de excepciones permite que el programa continúe su ejecución como si no hubiera habido error, esto permite crear programas tolerantes a fallos.

El proceso de gestión de excepciones consiste en atrapar y manejar los errores que ocurren durante la ejecución; por ejemplo, si se tiene un programa "corriendo" o ejecutándose y se introduce un dato no válido, se lanza una excepción, la cual debe atraparse y manejarse (véase **Figura 3.18**).

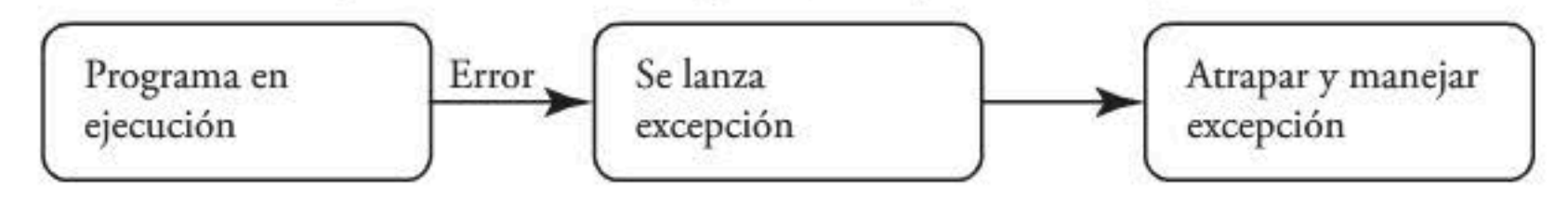


Figura 3.18. Manejo de Excepciones en Java.

Para realizar esto, sólo es necesario colocar dentro de un **`try`** el código que se desea "vigilar" para que si se produce un caso atípico o no esperado y se genera una excepción, el control pase al bloque que está dentro del **`catch`** para que se haga cargo de la situación.

Algunos de los tipos de excepciones más comunes son:

`ArithmeticException`

Esta excepción ocurre cuando se realiza una división entre cero; es decir, cuando el resultado produce un valor infinito.

`InputMismatchException`

Esta excepción ocurre cuando el usuario introduce datos no esperados para leerse por medio de los métodos de la clase Scanner. Se requiere importar la clase `java.util.InputMismatchException`.

`Exception`

Maneja cualquier tipo de excepción.

La sintaxis para manejar excepciones en Java es la siguiente:

```
try{
   código que se desea vigiliar
}
catch(tipoExcepcion objetoExcepcion){
   código para tratar la excepción
}
```

Ejemplo 3.11. El siguiente programa pide dos número enteros para ser leídos de teclado por consola, si no se introducen números enteros no termina con la ejecución porque la excepción que se genera se atrapa y se maneja (en este caso sólo se mandan mensajes) para después continuar con la ejecución normal.

```
//clase para leer de consola
import java.util.Scanner;
//clase para manejar excepciones de Scanner
```

```
import java.util.InputMismatchException;
public class ProgramaExcepcion1 {
    public static void main (String [] args){
        Scanner leer = new Scanner (System.in);
        int a=0, b=0;
        try{ //inicia código "vigilado"
            System.out.print ("Dame un número entero: ");
            a = leer.nextInt();
            System.out.print ("Dame otro número entero: ");
            b = leer.nextInt();
        }catch(InputMismatchException e){
/*si existe una excepción del tipo InputMismatchException
 realiza lo que esté dentro de este bloque */
            System.out.println("Leí datos inválidos");
            System.out.println("Se dio una excepción");
        }
        System.out.println("Se siguió con la ejecución");
    }
}
```

Ejecución sin excepciones	Ejecución con excepciones
run: Dame un número entero: 3 Dame otro número entero: 4 Se siguió con la ejecución GENERACIÓN CORRECTA (total time: 6 seconds)	run: Dame un número entero: 3 Dame otro número entero: esta es cadena Leí datos inválidos Se dio una excepción Se siguió con la ejecución GENERACIÓN CORRECTA (total time: 18 seconds)
En este caso el usuario sí introdujo valores válidos.	En este caso el usuario no introdujo valores válidos: se lazó una excepción, se atrapó y se manejó imprimiendo un mensaje. Después se imprimieron los demás mensajes como si no hubiera pasado nada.

Ejemplo 3.12. El siguiente programa pide dos números enteros para ser leídos de teclado por consola y después dividirlos, si no se introducen número enteros no termina la ejecución ni tampoco si al dividirse los dos números se obtiene un resultado infinito (ocurre cuando se divide entre 0) porque las excepciones que se pueden generar se atrapan y se manejan (en este caso sólo se mandan mensajes) para después continuar con la ejecución normal.

```
import java.util.Scanner;
import java.util.InputMismatchException;
public class ProgramaExcepcion2 {
    public static void main (String [] args){
        Scanner leer = new Scanner (System.in);
        int a=0, b=0, division;
        try{
            System.out.print ("Dame un número entero: ");
            a = leer.nextInt();
            System.out.print ("Dame otro número entero: ");
            b = leer.nextInt();
            division = a/b;
        }catch(InputMismatchException e){
/*si existe una excepción del tipo InputMismatchException
  realiza lo que esté dentro de este bloque */
  System.out.println("Leí datos inválidos");
  System.out.println("Se dio una excepción");
  }
  catch (ArithmeticException e){
/*si existe una excepción del tipo ArithmeticException
  realiza lo que esté dentro de este bloque */
  System.out.println("Resultado infinito");
  System.out.println("Se dio una excepción");
  }
    System.out.println("Se siguió con la ejecución");
    }
}
```

Ejecución sin excepciones	Ejecución con excepciones
run: Dame un número entero: 3 Dame otro número entero: 0 Resultado infinito Se dio una excepción Se siguió con la ejecución GENERACIÓN CORRECTA (total time: 6 seconds)	run: Dame un número entero: 3 Dame otro número entero: gato Leí datos inválidos Se dio una excepción Se siguió con la ejecución GENERACIÓN CORRECTA (total time: 12 seconds)
En este caso el usuario ingresó números que dan un resultado infinito; por lo que, se lanzó una excepción del tipo `ArithmeticException` que se atrapó en el `catch` y se manejó imprimiendo el mensaje "Resultado infinito".	En este caso el usuario ingresó un valor no válido (gato); por lo que se lanzó una excepción del tipo `InputMismatchException` que se atrapó en el `catch` y se manejó imprimiendo el mensaje "Leí datos inválidos".

Ejemplo 3.13. El siguiente programa es similar al del Ejemplo 3.12 con la diferencia de que no se especifica cuál es el tipo de excepción que puede manejar, al utilizarse `Exception` se atrapa cualquier tipo.

```
import java.util.Scanner;
public class ProgramaExcepcion3 {
 public static void main (String [] args){
 Scanner leer = new Scanner (System.in);
   int a=0, b=0, division;
   try{
      System.out.print ("Dame un número entero: ");
      a = leer.nextInt();
      System.out.print ("Dame otro número entero: ");
      b = leer.nextInt();
      division = a/b;
      }catch(Exception e){
   //se atrapara cualquier tipo de excepción
   System.out.println("Ocurrió un error en la ejecución");

   //se imprime el tipo de excepción ocurrida
   System.out.println(e.toString());
   }
     System.out.println("Se siguió con la ejecución");
     }
}
```

```
run:
Dame un número entero: 8
Dame otro número entero: 0
Ocurrió un error en la ejecución
java.lang.ArithmeticException: / by zero
Se siguió con la ejecución
GENERACIÓN CORRECTA
(total time: 13 seconds)
```

```
run:
Dame un número entero: 478
Dame otro número entero: cadena
Ocurrió un error en la ejecución
java.util.InputMismatchException
Se siguió con la ejecución
GENERACIÓN CORRECTA
(total time: 18 seconds)
```

3.13 Resumen

Java es un lenguaje de programación de alto nivel orientado a objetos que se creó con el patrocinio de Sun Microsystems en 1991 como parte del *Green Proyect* compuesto por trece personas y dirigido por James Goslin. La idea inicial era crear un lenguaje que permitiera a los programadores escribir el código una vez y ejecutarlo bajo cualquier plataforma. Se dice que Java debe su nombre a un tipo de café disponible en una cafetería cercana a Sun Microsystems, esa es la razón por la que su ícono es una taza de café caliente.

Java tiene sus bases en los lenguajes de programación C y C++ pero eliminando complejidades de estos lenguajes que propician errores al momento de programar. Una diferencia importante es que no usa apuntadores algo que es un problema para programadores menos experimentados en C y C++. En Java el programador no se debe preocupar por la administración de la memoria y la ubicación de la información, ya que cuando se desea crear un espacio para guardar información, simplemente hace la petición al lenguaje y Java le otorga el espacio y el control de la ubicación de los datos. Cuando se libera el espacio en memoria, Java tiene el método *Garbage Collector* (Recolector de Basura) para recuperar los espacios de memoria que no se utilizan, administrando eficazmente la memoria de la computadora. Java tampoco permite la herencia múltiple porque causa muchos problemas difíciles de predecir y opta solamente por la herencia simple, que usada adecuadamente suple la herencia múltiple de manera más fácil de aplicar y entender. Se dice que está basado en el paradigma orientado a objetos porque manejan las técnicas de herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento.

Java tiene muchas ventajas sobre los otros lenguajes de programación porque su uso es gratuito, es multiplataforma porque los programas en Java se pueden ejecutar en cualquier sistema operativo (Windows, Linux, Unix, Mac Os) sin realizar cambio alguno en los programas. Es *Open Source* porque el código fuente puede ser visto y modificado por cualquier usuario.

Un entorno de desarrollo es una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar los programas. Algunos de los entornos de desarrollo más conocidos son: JCreator, Eclipse, JGrasp y NetBeans IDE. En este libro se usará NetBeans IDE para editar los programas en Java, porque es muy didáctico y permite la identificación de errores en el momento de escribir el programa. Es posible descargar e instalar el *software* para programar en Java, directamente de la siguiente página:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Los programas en Java tienen tres fases fundamentales: *edición*, *compilación* y *ejecución*. En la edición el programador escribe su programa y guarda el archivo con una extensión **.java**. En la compilación se utiliza el java.exe para convertir el código fuente en código muy cercano al binario (baycode) y guarda el archivo con extensión **.class**. En la ejecución la JVM (*Java Virtual Machine*) carga los archivos .class en la memoria, verifica el código y lo convierte a código máquina para ejecutarlo finalmente. La JVM es la encargada de ejecutar los archivos escritos en Java en cualquier plataforma de trabajo.

La estructura básica de un programa en Java es la siguiente:

```
public class Programa {
public static void main (String args []){
  instrucciones;
  instrucciones;
  }
}
```

La primera línea indica el nombre de la clase (o programa). La segunda línea corresponde al programa principal que todos los lenguajes en Java deben tener. Tanto la clase como el programa principal enmarcan la información por medio de las llaves correspondientes. Es posible observar que dentro de la clase está el programa principal, y dentro del programa principal se encuentran todas las instrucciones del programa.

Todas las variables usadas en un programa de Java deben ser declaradas según el tipo de datos o información que manejarán. En Java se tienen los siguientes tipos de datos primitivos: **`byte`**, **`short`**, **`int`**, **`long`**, **`float`**, **`double`**, **`char`** y **`boolean`**.

Existen en Java operadores aritméticos que permiten llevar a cabo las operaciones básicas de suma, resta, multiplicación, división y módulo. Además se dispone de la librería `Math` que contiene métodos para calcular las principales funciones matemáticas como: seno, coseno, tangente, potenciación, valor absoluto, entre otras.

3.14 Problemas

1. Escribir en forma concreta al menos tres ventajas de Java sobre los demás lenguajes de programación.
2. Solamente diez de las veinte respuestas de la columna de la izquierda, tienen relación con los textos de la derecha. Colocar la letra que corresponda en el paréntesis de la derecha.

   Respuestas:

a) Orkad	Java fue creado en 1991 por la compañía	( )
b) James Goslin	Java tiene sus raices en los lenguajes	( )
c) Microsoft Co.	En honor a un viejo roble Java fue llamado inicialmente.	( )
d) NetBeans IDE		
e) C y C++	*Green Proyect* compuesto por trece personas diseñadores de Java estaba dirigido por.	( )
f) Jonh Bauer		
g) Multitarea.	Auxilia a Java para mejorar el rendimiento de la memoria.	( )
h) Prolog y Ada		
i) Sun Microsystems	En C++ esta característica da origen a muchas situaciones en donde es difícil predecir cuál será el resultado.	( )
j) Oak		
k) *Garbage Collector*	Herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas.	( )
l) Entorno de desarrollo		
m) POO	Está basado en varias técnicas, incluyendo herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento.	( )
n) JCreator		
ñ) Editor		
o) Herencia múltiple	Permite ejecutar el código binario generado por el compilador de Java en cualquier plataforma o sistema operativo.	( )
p) La propiedad .exe		
q) Programación estructurada	Nació en 1996 como un proyecto estudiantil (llamado *Xelfi*) en la Facultad de Matemáticas y Física en la Universidad Carolina en Praga	( )
r) Compilador		
s) La JVM		

3. Codificar en Java cada ua de las siguientes expresiones matemáticas, usando paréntesis cuando sean requeridos para agrupar información. Considerar que cada una de las letras es una variable diferente a excepción de la palabra "mod" que representa la operación aritmética módulo.

   a) $y = b \bmod (c - 5d) + \dfrac{32}{c \bmod 7}$

b) $y = \dfrac{\dfrac{8b - 7 \bmod 5}{ac - d}}{(4c - 3) \bmod a}$

c) $y = \dfrac{3 \bmod (4 + b \bmod 2) - 8}{9c - a} + d \bmod 3$

4. Codificar en Java las expresiones matemáticas de cada uno de los incisos. Considerar que cada una de las letras es una variable diferente a excepción de la palabra "mod" que representa la operación aritmética módulo.

   a) $y = (a \bmod bc - ad \bmod 3) \bmod 2$

   b) $y = \dfrac{\dfrac{(4d - b \bmod 5)6}{3(a - 7 \bmod b)}}{d \bmod c} - 7 \bmod ab$

   c) $y = \dfrac{a \bmod (c - db)}{b \bmod 5 + 2} + b \bmod 3 + 2$

5. Representar por edio de la expresión matemática correspondiente las líneas de código de cada uno de los incisos siguientes:

   a) y = (3%b−5)/(4*(7%b+6)+a*c)+c%6;

   b) y = (c*d+5%b)%(7*a+8)+3%c/a;

   c) y = (4%a-3*((b%5+8)/(c+b%5)))/(a*c);

6. Representar por medio de la expresión matemática correspondiente las líneas de código de cada uno de los incisos siguientes:

   a) y = ((a-(c*d+2)%5)*6)/(7*a+b%4)+7/(a%3);

   b) y = (d%(a+3)−7%c)/(4*d%5)+(b-9*a)/(7%c);

   c) y = (5*c+9%c−7)/((4%(c+2*b))/(b+c*d))+(b-3*a)/2;

7. Evaluar paso a paso las líneas de código de cada uno de los incisos del problema 5, considerando que las variables se definen de tipo entero de la siguiente manera:

```
int a=2, b=4, c=−3, d=8, y;
```

8. Evaluar paso a paso las líneas de código de cada uno de los incisos del problema 6, considerando que las variables se definen de tipo entero de la siguiente manera:

```
int a=−1, b=3, c=7, d=−4, y;
```

9. Codificar en Java cada una de las siguientes expresiones matemáticas, considerando que cada una de las letras es una variable a excepción de las funciones trigonométricas (cos y csc).

a) $y = \dfrac{4e-7}{3a(5-b)^{2/3}} + \dfrac{w}{4-d} - 1$

b) $y = \dfrac{\sqrt{3x-2}}{\cos(40^\circ)} - \dfrac{1}{5x+4}$

c) $y = \sqrt[3]{(7a-b)^2}\ \dfrac{\csc(70^\circ)}{b-4e}$

10. Codificar en Java cada una de las siguientes expresiones matemáticas, considerando que cada una de las letras es una variable a excepción de las funciones trigonométricas (tan y cos) y que los ángulos de las funciones trigonométricas se encuentran en grados.

a) $y = \dfrac{\dfrac{6x-2}{a^2}}{\left|4x + \dfrac{3}{8d}\right|} + x(5a-7)$

b) $y = \dfrac{7a(8b^3-4)}{\dfrac{6\tan(30^\circ + x)}{2a}} + \dfrac{3}{5b}$

c) $y = \sqrt[4]{\dfrac{\sqrt{3x-2}}{\cos(5+x)}} - \dfrac{8b-a}{7}$

11. ¿Cuál es la expresión matemática equivalente de cada una de las líneas de código expresadas a continuación?.

a) y = 3*a*(b−4*(c−2*d))+(5*e−8)/(7*a);

b) y = ((4***Math.pow**(x,2)−8)/7)/(6*(2*x−14))+(5-**Math.pow**(x,3))/8;

c) y = **Math.cos**(60*3.1416/180)−**Math.sqrt**(**Math.pow**(x,2)−4)/**Math.abs**(7*x-8);

12. ¿Cuál es la expresión matemática equivalente de cada una de las líneas de código expresadas a continuación?.

a) y = (6*a-7*(b+3))/**Math.sqrt**(**Math.pow**(a,2)-5*c)+(4*b)/(c−2);

b) y = ((3***Math.pow**(a,4)-5)/**Math.pow**(70*3.1416/180))/(b−4*d)+ **Math.pow**((6*d-7)/9,1/3.0);

c) y = (1/**Math.tan**((x+20)*3.1416/180))/(**Math.sin**((3*x)*3.1416/180)−7*(**Math.pow**(x,2)+1));

13. Escribir un programa para leer los valores de los coeficientes de la ecuación de segundo grado (a, b y c) cuya forma es la siguiente:

$$ax^2 + bx + c = 0$$

Posteriormente encontrar sus raíces usando para ello la fórmula general.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Considerar que se tiene la ecuación: $3x^2 + 7x - 6 = 0$, entonces la salida es:

```
a: 3
b: 7
c: −6
Resultado
x1 = 0.6667
x2 = −3.0000
```

14. Escribir un programa para leer el radio de una circunferencia (r) y posteriormente calcular el área y perímetro de la circunferencia, sabiendo que:

$$\text{Área} = \pi r^2 \qquad \text{Perímetro} = 2 \pi r$$

```
radio: 5
Área = 78.54
Perímetro = 31.416
```

15. Escribir un programa para evaluar el siguiente polinomio con un valor de x cualquiera.

$$y = \frac{\sqrt[3]{5x^4 - 6x^2 + 8}}{9}$$

```
x: −3.5
y = 4.2375
```

16. Escribir un programa para leer de teclado una cantidad expresada en Kilogramos (Kg) y convertir dicha cantidad a Toneladas (Ton), gramos (gr), onzas (oz) y libras (lb) considerando las siguientes equivalencias de peso.

1 Ton = 1000 Kg 1 Kg = 1000 gr 1 oz =28.35 gr 1 lb = 454 gr

```
Kilogramos: 678
678 Kilogramos es equivalente a:
a) 0.6780 Ton
b) 678000.00 gr
c) 23915.3439 oz
d) 1493.3921 lb
```

17. Escribir un programa para leer un ángulo (en grados) y calcular las funciones trigonométricas (seno, coseno, tangente, cotangente, secante y cosecante).

```
Ángulo (en grados): 30
sen (30) = 0.5000
cos (30) = 0.8660
tan (30) = 0.5774
cot (30) = 1.7320
sec (39) = 1.1547
csc (39) = 2.0000
```

18. Escribir un programa para leer dos números enteros (x, y) de teclado y realizar las siguientes operaciones con ellos:

   a) Multiplicarlos (x*y).

   b) Dividir el primero entre el segundo (x/y).

   c) Elevar el primero a la potencia del segundo (x^y).

   d) Encontrar la raíz del primero con respecto al segundo $\left(\sqrt[y]{x}\right)$.

   e) Encontrar el módulo del primero con respecto al segundo (x mod y).

```
x: 2
y: 3
2*3 = 6.00
2/3 = 0.67
2^3 = 8.00
Raíz 3 de 2 = 1.26
2 mod  3 = 2
```

CONTROL DE FLUJO 4

Primero entiende el problema, luego escribe el código.

John Johnson

Competencia de la unidad

Construir programas utilizando estructuras condicionales y repetitivas para aumentar su funcionalidad.

- Diseñar programas donde se utilicen las estructuras de repetición y selección.
- Construir programas con funciones y/o métodos.
- Escribir programas en donde se utilicen algunas funciones tipo cadena.
- Escribir programas usando funciones recursivas.

Control de flujo

Contenido

4.1. Introducción.
4.2. Estructuras selectivas: simple, doble y múltiple.
4.3. Estructuras iterativas: mientras, hacer–mientras, desde.
4.4. Ciclos anidados.
4.5. Manipulación de cadenas.
4.6. Diseño e implementación de funciones y métodos.
4.7. Resumen.
4.8. Problemas.

4.1 Introducción

El control de flujo permite encausar a la computadora sobre la ruta que debe seguir al momento de la ejecución de un programa, para ello se apoya en las estructuras de control que le permitirán tomar decisiones, repetir la ejecución de un grupo de instrucciones, subdividir los programas grandes en pequeños programas que realizan una actividad determinada.

En esta unidad se tratarán las estructuras selectivas simples (**`if`**), dobles (**`if-else`**) y múltiples (**`switch`**) aplicadas a problemas en donde sean necesarias las bifurcaciones. Las estructuras iterativas mientras (**`while`**), hacer—mientras (**`do-while`**) y desde (**`for`**) para ejecutar un bloque de sentencias mientras cierta proposición sea verdadera. Se resolverán problemas que requieran ciclos anidados e instrucciones selectivas anidadas, buscando con ello la potenciación de las estructuras de control.

También se verá la segmentación y control de programas en subprogramas más pequeños llamados métodos y/o funciones, como una manera de preparar el terreno para atacar sistemas de programación más complejos en el momento en que se requiera. Se abordarán métodos iterativos y recursivos con la finalidad de que se tengan los elementos para resolver problemas que presentan una naturaleza recursiva.

4.2 Estructuras selectivas: simple, doble y múltiple

La complejidad de los problemas hacen que difícilmente los programas sean solamente una estructura secuencial, muchas veces es necesario ejecutar una instrucción o bloque de instrucciones dependiendo de si se cumple o no una condición. Las preguntas se plantean por medio de condiciones estructuradas en forma lógica y el resultado será verdadero o falso pero no ambos a la vez, en función del resultado será el bloque de acciones a ejecutar. Las instrucciones selectivas se clasifican en: simples, dobles y múltiples.

4.2.1 Instrucción selectiva simple (if)

Se dice que la instrucción selectiva es simple si al cumplirse la condición se ejecuta una instrucción o bloque de instrucciones y en caso de ser falsa no se ejecuta acción alguna. El formato general de una instrucción selectiva simple es:

```
if (condición)
  {
     .........
     instrucciones.
     .........
  }
```

La condición de la instrucción selectiva es una proposición lógica integrada por variables, operadores relacionales y/o operadores lógicos. Las instrucciones que se ejecutan en caso de que la proposición sea verdadera, pueden ser cualquiera que la computadora pueda llevar a cabo. Ejemplos típicos de la instrucción selectiva simple se muestran a continuación.

```
a)                                          b)
if (promedio >= 70){                        if (promedio >=70)
 System.out.println ("Felicidades ");       n++;
 n++;
 }
```

En el primer caso se utilizan llaves para encerrar el bloque de instrucciones que se deben ejecutar si la condición se cumple. En el segundo caso no son necesarias esas llaves, porque es solamente una instrucción la que se debe realizar si se cumple la condición. La regla es que si son más de una sentencia las que se deben ejecutar se deben encerrar entre llaves, pero si solamente se trata de una instrucción, podrían no ser necesarias las llaves; aunque si se ponen igual funciona perfectamente bien.

Ejemplo 4.1. Escribir un programa para leer el nombre y precio de un artículo. Si el artículo tiene un precio mayor o igual a 100 pesos, imprimir el nombre del artículo y el mensaje "es muy caro" y el precio del artículo. Posteriormente en otra línea desplegar el mensaje "máximo puedo pagar por este artículo $100.00". Si el artículo tiene un precio menor a $100, que no se imprima nada.

```
import java.util.Scanner;
public class ifsimple {
public static void main(String[] args){
    String arti;
    double precio;

    Scanner leer= new Scanner(System.in);

     System.out.print ("Nombre del artículo: ");
     arti= leer.next();
     System.out.print("Precio (pesos):");
     precio= leer.nextDouble();

    if (precio>=100) {
    System.out.printf ("%s es muy caro, vale $%.2f\n",arti,precio);
       System.out.println ("máximo puedo pagar por un "+arti+"
$100.00");
     }
   }
  }
```

```
run:
Nombre del artículo: Martillo
Precio (pesos):143
Martillo es muy caro, vale $143.00
máximo puedo pagar por un Martillo $100.00
GENERACIÓN CORRECTA (total time: 11 seconds)
```

4.2.2. Instrucción selectiva doble (if - else)

La estructura selectiva doble se utiliza cuando se tienen dos opciones de acción. Con la bifurcación doble se ejecuta un bloque de instrucciones A si se cumple la condición o bien se ejecuta el bloque de instruc-

ciones B en caso de que no se cumpla, debido a que son mutuamente excluyentes. La estructura selectiva doble tiene el siguiente formato:

```
if (condición) {
     ……
     Instrucciones A
     ……
    }
else {
           ……
         Instrucciones B
         ……
      }
```

Nuevamente el uso de las llaves es importante para agrupar el bloque de instrucciones a ejecutarse si se cumple o no la condición. A continuación se tienen dos casos típicos de la doble condición.

```
if (promedio >= 80){
System.out.
println("Felicidades ");
n++;
}
else
System.out.print("Buen
promedio ");
```

```
if (promedio < 70){
 System.out.
println("Lastima ");
 n++;
 }
 else {
 System.out.println("Bajo
promedio ");
 System.out.
println("Estudia más");
 }
```

En el primer caso se usaron las llaves para agrupar las instrucciones que se ejecutarán si se cumple la condición, pero si no se cumple no fue necesario usar llaves porque es solamente una instrucción la que se debe ejecutar. En el segundo caso se utilizaron llaves para agrupar las instrucciones que se deben ejecutar si se cumple o no la condición, porque en ambos casos es más de una instrucción.

Ejemplo 4.2. Escribir un programa para leer el tiempo de duración de una llamada telefónica y determinar la cantidad a pagar, de acuerdo a lo siguiente:

a) Toda llamada que dure 3 minutos o menos tiene un costo de $2.50.
b) Cada minuto adicional cuesta $1.50

```
import java.util.Scanner;

public class llamada {
public static void main(String[] args){
```

```
    double tiempo, costo_minimo=2.50, costo;
    Scanner leer = new Scanner(System.in);

    System.out.print ("Tiempo (minutos): ");
     tiempo=leer.nextDouble();

    if (tiempo>3){
      costo=costo_minimo+(tiempo-3)*1.50;
      System.out.printf ("Costo= $%.2f\n",costo);
        }
    else
      System.out.printf ("Costo= $%.2f\n",costo_minimo);
    }
  }
```

```
run:
Tiempo (minutos): 5.3
Costo= $5.95
GENERACIÓN CORRECTA (total time: 8 seconds)
```

En el programa anterior se lee el tiempo en minutos de la llamada, si la duración es de más de tres minutos el precio de la llamada resulta de sumar el costo mínimo de $2.50 para los primeros 3 minutos, más los minutos restantes multiplicados por $1.50. En caso de que la duración sea menor o igual a 3 minutos, solamente se cobrará el costo mínimo.

Operadores relacionales. Con ellos es posible llevar a cabo la comparación de información. La comparación permite determinar si la proposición de una instrucción selectiva es falsa o verdadera. Los operadores relacionales usados en el lenguaje Java se muestran en la siguiente tabla.

Operador	Significado	Ejemplo
>	Mayor que	**if** (x > y)
<	Menor que	**if** (x < y)
>=	Mayor o igual que	**if** (x >= y)
<=	Menor o igual que	**if** (x <= y)
==	Igual a	**if** (x == y)
!=	Diferente de	**if** (x != y)

Operadores lógicos. Adicionalmente a los operadores relacionales es posible utilizar en caso de ser requeridos los operadores lógicos, para determinar si son falsas o verdaderas más de una proposición. A continuación se tienen los operadores lógicos más utilizados en Java.

Operador	Significado	Operador en java	Ejemplo
and	y	&&	`if (x > y && m == n)`
or	o	||	`if (x > y || (m == n))`
not	no	!	`if !(m == n)`

El operador || se puede obtener presionando la tecla ALT y simultáneamente teclear el número 124 (ALT+124). O bien en algunos teclados pulsando ALT GR+1.

Una proposición puede estar integrada por signos aritméticos, operadores relacionales, paréntesis, operadores lógicos e incluso el signo de igualdad. De tal manera que es conveniente saber qué operación tiene mayor jerarquía de operación con el objeto de evitar errores al momento de estructurar la proposición. A continuación se tienen los operadores y su jerarquía de operación.

Operador	Jerarquía de operación
( )	1a.
*, /, %	2a.
+, −	3a.
>, >=, <, <=	4a.
= =, !=	5a.
=	6a.
!	7a.
&&	8a.
||	9a.

Para aquellos operadores que tienen la misma jerarquía, se evalúa primero el operador que esté más a la izquierda.

Ejemplo 4.3. Si a=3, b=−1, c=7, letra='w'. ¿Cuál es el mensaje que se imprime en la siguiente instrucción selectiva?:

```
if (a>=b*c && !(letra!='w' || c==a+5) || 2<c%a)
        System.out.println ("Hola manzana ");
  else
        System.out.println ("Adiós pera ");
```

Respuesta. La computadora lo hace rápidamente pero aquí se llevará a cabo la evaluación paso a paso para observar el proceso.

Operador	Explicación
`if (3>=(-1)*7 && !('w'!='w' || 7==3+5) || 2<7%3)`	Sustituyendo valores.
`if (3>=(-1)*7 && !('w'!='w' || 7==8) || 2<7%3)`	Suma del paréntesis interno.
`if (3>=(-1)*7 && !(falso || 7==8) || 2<7%3)`	Dentro del paréntesis está (!=) y (==) que tienen la misma jerarquía, pero se evalúa el que está más a la izquierda (!=).
`if (3>=(-1)*7 && !(falso || falso) || 2<7%3)`	Ahora el (==).
`if (3>=(-1)*7 && !(falso) || 2<7%3)`	El || del paréntesis.
`if (3>=-7 && !(falso) || 2<7%3)`	La multiplicación.
`if (3>=-7 && !(falso) || 2<1)`	El modulo (%).
`if (verdadero && !(falso) || 2<1)`	El (>=).
`if (verdadero && !(falso) || falso)`	El (<).
`if (verdadero && verdadero || falso)`	La negación not (!).
`if (verdadero || falso)`	El and (&&).
`if (verdadero)`	Finalmente el or (||).

Lo cual implica que el mensaje que se imprime es "`Hola manzana`".

Muchos programas utilizan instrucciones en donde una sentencia selectiva está dentro de otra, y dentro de ésta última; está otra instrucción selectiva. En estos casos se dice que el programa tiene **`ifs`** anidados.

Ejemplo 4.4. A los alumnos de nuevo ingreso del Tecnológico de Morelia se les entregarán recomendaciones para becas. La recomendación se basa en lo siguiente.

a) promedio ≥ 90	Muy buena recomendación
b) 80 ≤ promedio <90	Buena recomendación
c) 70 ≤ promedio <80	Endeble recomendación

Escribir un programa que lea el nombre y promedio del alumno y que con esa información indique el tipo de recomendación para ese alumno.

```
import java.util.Scanner;

public class beca {
  public static void main(String[] args){
   int promedio;
   String nombre;
```

```
   Scanner leer = new Scanner(System.in);

System.out.print ("Nombre: ");
nombre=leer.nextLine();

System.out.print ("Promedio: ");
promedio= leer.nextInt();

 if (promedio>=90)
   System.out.println (nombre+", Tiene MUY BUENA recomendación");
  else
   if (promedio>=80 && promedio<90)
     System.out.println (nombre+", Tiene BUENA recomendación");
   else
     if (promedio>=70 && promedio<80)
      System.out.println (nombre+", Tiene ENDEBLE recomendación");
    else
         System.out.println (nombre+", Checa tu promedio por fa-
vor");
    }
   }
```

```
run:
Nombre: Carlos Alberto
Promedio: 87
Carlos Alberto, Tiene BUENA recomendación
GENERACIÓN CORRECTA (total time: 16 seconds)
```

Observar que si se cumple la condición: **`if`** `(promedio>=90)`, imprime el nombre del alumno y el mensaje de: `MUY BUENA recomendación`, Sí la condición es falsa, ejecuta la instrucción selectiva: **`if`** `(promedio>=80 && promedio<90)`, si la condición de este nuevo **`if`** se cumple imprime el mensaje correspondiente y en caso de ser falsa ejecuta una nueva instrucción condicional. A esta forma de colocar un **`if`** dentro de otro **`if`**, se conoce en computación como **`ifs`** anidados.

4.2.3 Instrucción condicional múltiple (switch)

Existen programas en donde la cantidad de instrucciones condicionales anidadas requeridas es considerable, y por lo tanto se tiene que escribir mucha información en cada línea, porque las instrucciones selectivas anidadas ocupan mucho espacio, además de que no son ilustrativos los programas con muchos **`ifs`** anidados, ya que causan confusión al momento de tratar de entenderlos. Java tiene la instrucción de selección múltiple "`switch`" que permite sustituir estos **`ifs`** anidados por una estructura fácil de entender y usar. El formato general de la instrucción `switch` es:

```
switch (variable)
  {
   case valor_1:
       ...
      Instrucciones A
      break;
   case valor_2:


      ......
      Instrucciones B
      break;
      ..............................
      ..............................
   case valor_n:
      .........
      Instrucciones N
      break;
     default:
      .........
      Instrucciones
      break;
  }
```

Donde:
variable: Es una variable o expresión cuyo valor es un dato: entero, byte, short o char.

valor_1, valor_2,..,valor_n: Son valores de la variable para cada uno de los casos posibles (case).

break: Instrucción que tiene la finalidad de romper la ejecución del flujo.

default: En caso de que no se ejecuten ninguno de los casos previstos se ejecutarán las instrucciones por default.

Si la variable tiene el `valor_1` entonces se ejecutarán el bloque de instrucciones para el caso `valor_1`, si el valor de la variable es `valor_2` se ejecuta el bloque de instrucciones del caso `valor_2`. Si la variable tiene un valor no considerado en los casos, se ejecutará el bloque por `default`. Default es opcional; lo cual significa que si el valor no está considerado en los casos y no se desea ejecutar instrucción alguna, entonces no se pone el `default`.

Observar que la estructura de **`switch`** se debe encerrar entre llaves para indicar el alcance de la instrucción selectiva múltiple. También se puede observar que en cada caso se tiene un *`break`* para limitar las acciones a ejecutar para cada opción seleccionada. Los valores de la variable de selección, solamente pueden ser enteros (*`int`*), entero corto (*`short`*), *`byte`* o caracter (*`char`*).

Ejemplo 4.5. Escribir un programa para desplegar en la pantalla una lista de 4 países, cada uno de ellos con su correspondiente código. Se deberá leer el código del país y la computadora indicará cuál es la capital, continente, idioma y moneda oficiales de ese país.

```
import java.util.Scanner;
public class capitales {
public static void main(String[] args){

  Scanner leer = new Scanner(System.in);
  int cod;
```

```
System.out.println ("Lista de países ");
System.out.println ("Código \t\t País");
System.out.println ("   1 \t\t México");
System.out.println ("   2 \t\t Italia");
System.out.println ("   3 \t\t Surinam");
System.out.println ("   4 \t\t Suecia");

System.out.print ("  Código: ");
cod= leer.nextInt();

switch (cod)
 {
  case 1:
    System.out.println("País: México \t Capital: Distrito Federal
\n");
    System.out.println("Continente:Americano Idioma:Español Mone-
da: Peso\n");
    break;
   case 2:
    System.out.println("País: Italia \t Capital: Roma \n");
    System.out.println("Continente:Europeo Idioma:Italiano  Mone-
da: Euro\n");
    break;
   case 3:
   System.out.println("País: Surinam \t Capital: Paramaribo \n");
    System.out.println("Continente:Africano Idioma:Holandés  Mo-
neda: Guinea\n");
    break;
   case 4:
    System.out.println("País: Suecia \t Capital: Estocolmo \n");
            System.out.println("Continente:Europeo   Idioma:Sueco
Moneda:Corona Sueca\n");
    break;
   default:
    System.out.println("Código Inexistente\n");
    break;
    }
   }
  }
```

```
run:
Lista de países
Código          País
   1            México
   2            Italia
   3            Surinam
   4            Suecia
Código: 3
País: Surinam     Capital: Paramaribo
Continente: Africano Idioma: Holandés  Moneda: Guinea
GENERACIÓN CORRECTA (total time: 6 seconds)
```

En el programa anterior se lee el código de un país, dependiendo del valor del código se desplegará en la pantalla la capital, el continente y la moneda de dicho país. Solamente funciona para cuatro países, pero podría funcionar para todos los países existentes, agregándole los `case` correspondientes.

4.3. Estructuras iterativas: mientras, hacer-mientras, desde

Una estructura iterativa permite ejecutar una instrucción o un bloque de instrucciones varias veces. En el lenguaje Java se tienen fundamentalmente las siguientes tres estructuras iterativas:

a) Mientras (**`while`**).
b) Hacer-mientras (**`do-while`**).
c) Desde (**`for`**).

Las tres permiten ejecutar cierto número de veces un bloque de instrucciones, pero tienen pequeñas diferencias que es conveniente analizar.

4.3.1. Mientras (`while`)

La estructura iterativa **`while`** es la más utilizada en los diferentes lenguajes de programación debido a la versatilidad que tiene. Con **`while`** se ejecuta el bloque de instrucciones encerradas entre llaves, mientras se cumpla la condición establecida. Su formato general es el siguiente:

```
while (condición) {
   ……..
   Instrucciones;
   ……
   }
```

Una característica de **`while`** es que si en la primera vez no se cumple la condición no se ejecutará ninguna vez el bloque de instrucciones encerradas entre llaves, debido a que la condición está al principio del bloque.

Ejemplo. 4.6. Escribir un programa para leer n números enteros de teclado y contar cuántos de ellos son pares y cuántos son impares.

```
import java.util.Scanner;
public class cuenta_pares {
public static void main(String[] args){
    Scanner lectura = new Scanner(System.in);

int n, i, num, par=0;

System.out.print("Cuántos números son: ");
n=lectura.nextInt();

System.out.println ("Dame los "+n+" números: ");
i=0;
while (i<n){
  num=lectura.nextInt();
  if (num % 2==0)
      par+=1;
  i++;
  }
System.out.printf("Números pares= %d, Números impares= %d\n",par,
(n-par));
}
}
```

```
run:
Cuantos números son: 5
Dame los 5 números:
3
8
-12
14
7
Números pares= 3,  Números impares= 2
GENERACIÓN CORRECTA (total time: 25 seconds)
```

Si al leer el valor de n se le diera cero, no hubiera ejecutado ninguna vez el bloque de instrucciones que están entre llaves después de **`while`**, porque la condición planteada no se cumple.

Otra característica de **`while`** es que no es necesario saber de antemano el número de veces que se ejecutará el bloque de instrucciones.

Ejemplo. 4.7. Escribir un programa para leer números enteros de teclado hasta que el número leído sea 99. Contar cuántos números se leyeron en total sin considerar el 99, cuántos de ellos son divisibles exclusivamente entre 3, cuántos son divisibles exclusivamente entre 7, cuántos son divisibles entre 3 y 7 a la vez.

```
import java.util.Scanner;
public class divisibles_de_3_y_7 {
    public static void main(String[] args){
    Scanner leer = new Scanner(System.in);

int x, div3, div7, tot, ambos;
div3=0; div7=0; tot=0; ambos=0;
System.out.println ("Dame números enteros (finalizar con 99): ");
x=leer.nextInt();
tot=0;
while (x!=99){
   if (x%3==0 && x%7==0)
      ambos+=1;
  else {
         if (x % 3==0)
            div3+=1;
         if (x%7==0)
            div7+=1;
        }
  x=leer.nextInt();
  tot++;
  }
System.out.printf ("Total leídos=%d \n",tot);
System.out.printf ("Divisibles entre 3 (exclusivamente) = %d\n",div3);
System.out.printf("Divisibles entre 7 (exclusivamente) =%d\n",div7);
System.out.printf("Divisibles entre 3 y 7=%d \n",ambos);
}
}
```

```
run:
Dame números enteros (finalizar con 99):
5
12
21
14
63
28
99
Total  leídos=6
Divisibles entre 3 (exclusivamente)= 1
Divisibles entre 7 (exclusivamente)=2
Divisibles entre 3 y 7=2
GENERACIÓN CORRECTA (total time: 34 seconds)
```

En el programa anterior no se sabe cuántas veces se ejecutarán las instrucciones que están dentro de **while** porque no se sabe hasta cuándo el usuario teclea el número 99.

Ejemplo 4.8. Un número *cuadrado perfecto* es aquel que se obtiene al elevar al cuadrado un número natural, de la misma forma que un *cubo perfecto* resulta de elevar al cubo un número natural. Los números 4 y 9 son cuadrados perfecto porque $2^2=4$ y $3^2=9$, así como 125 y 729 son cubos perfectos porque $125=5^3$ y $729=9^3$.

Escribir un programa para imprimir todos los números cuadrados y/o cubos perfectos comprendidos en el rango m y n. Indicar cuáles y cuántos son cuadrados o cubos perfectos.

```
import java.util.Scanner;
public class cuadrado_y_cubo {
public static void main(String[] args){

  Scanner leer = new Scanner(System.in);

int n, m, cuadrados=0, cubos=0;
double a;

System.out.print("m: ");
m= leer.nextInt();
System.out.print("n: ");
n= leer.nextInt();
System.out.println();

while (m <= n){
  if (m==(Math.pow((int)Math.pow(m,1/2.0),2))){
   System.out.println(m+" cuadrado ");
   cuadrados++;
   }
 if (m==(Math.pow((int)Math.pow(m,1/3.0),3))){
   System.out.println(m+" cubo ");
   cubos++;
   }
 m++;
 }
System.out.println("Total cuadrados = "+cuadrados);
System.out.println("Total cubos = "+cubos);
}
}
```

```
run:
m: 7
n: 30

8 cubo
9 cuadrado
16 cuadrado
25 cuadrado
27 cubo
Total cuadrados = 3
Total cubos = 2
GENERACIÓN CORRECTA (total time: 10 seconds)
```

Para encontrar la raíz cuadrada de un número se eleva a la un medio (`1/2.0`) dicha cantidad utilizando la función `pow` de la librería `Math`. Por ejemplo:

```
int x=9, y=23;
double c;

int a=(int)Math.pow(x,1/2.0);   //El resultado es a=3
int b=(int)Math.pow(y,1/2.0);   //El resultado es b=4
c = Math.pow(y,1/2.0);      //El resultado es c=4.795831523
```

Observar que las variables x, y, a y b se definieron de tipo entero, pero el resultado de `Math.pow(x,1/2.0)` es una cantidad `double` y no puede ser asignado a una variable que es entera, por lo tanto se hizo un "cast" con la palabra `(int)`, para obligar a que el resultado sea entero. Sin embrago para la variable c no se hizo el cast y el resultado tiene la parte entera y la fraccionaria.

En Java, al hacer operaciones con números enteros el resultado es entero, pero si alguno de ellos es `double` el resultado es `double`. Ejemplo: si se dividen dos números enteros el resultado es un entero (1/2 = 0), pero si uno de ellos es `double` el resultado ya no es entero (1/2.0 = 0.5), es por esa razón que se le agrega (0) al denominador para que al dividirse no sea un cero, porque en caso contrario no se estaría sacando la raíz cuadrada de `m` sino elevándola a la potencia cero.

En el programa se generan los números desde `m` hasta `n`. Donde `m` se incrementa en uno (`m++;`) en cada iteración, mientras la condición que está en **`while`** `(m <= n);` sea verdadera. Para cada valor de `m` se obtiene la raíz cuadrada, pero se le trunca la parte fraccionaria con el cast, después se eleva al cuadrado el entero de la raíz cuadrada de `m` y si es igual al `m` inicial, entonces `m` es cuadrado perfecto. Lo mismo se hace para el cubo perfecto, se eleva al cubo el entero de la raíz cubica de `m`, para determinar si es un cubo perfecto. En caso de que sean cuadrados o cubos perfectos se incrementan las variables correspondientes: `cuadrados++;` o `cubos++;`

4.3.2. Hacer-mientras (do-while)

Con el ciclo **`do-while`** se ejecuta el bloque de instrucciones que se encuentran encerradas entre las llaves de la palabra *do* y la palabra `while` con la condición correspondiente. El formato general es el siguiente:

```
do {
  ……
  Instrucciones;
  ……
}while (condición);
```

Ejemplo 4. 9. Escribir un programa para simular n lanzamientos de una moneda al aíre. Los resultados se generarán aleatoriamente. Imprimir cada uno de los resultados e imprimir los resultados obtenidos al final del evento.

```
import java.util.Scanner;
import java.util.Random;

public class lanzamiento {
public static void main(String[] args){
 Scanner leer = new Scanner(System.in);
 Random aleatorio=new Random();

int n, soles, x, aguilas, c;

System.out.print("Número de lanzamientos: ");
n=leer.nextInt();

soles=0; aguilas=0; c=0;
System.out.println ("Resultados: ");

do {
     //genera número natural aleatorio menor a 2
     x=aleatorio.nextInt(2);
     if (x==1){
      System.out.println ("Sol");
      soles+=1;
      }
    else {
      System.out.println ("Águila");
      aguilas+=1;
      }
      c++;
    } while (c<n);
   System.out.printf ("%d Soles y %d Águilas \n",soles, aguilas);
  }
 }
```

```
run:
Número de lanzamientos: 6
Resultados:
Águila
Águila
Sol
Águila
Águila
Sol
2 Soles y 4 Águilas
GENERACIÓN CORRECTA (total time: 3 seconds)
```

En el programa anterior además de importar la clase **`import java.util.Scanner;`** se importó la clase **`import java.util.Random;`** que permite generar números aleatorios.

La variable c se encarga de contar el número de veces que se ejecutarán las instrucciones que se encuentran entre las palabras **`do`** y **`while`**. El bloque de instrucciones se ejecutará mientras se cumpla la condición planteada en **`while`**.

Con la línea: **`Random`** `aleatorio=`**`new Random();`** se define también un objeto llamado "aleatorio" que permite generar un valor entero aleatorio y que se asigna a la variable x por medio de la siguiente instrucción: `x=aleatorio.`**`nextInt`**`(2);`. El 2 entre paréntesis indica que el número aleatorio será menor que 2 (0 ó 1). La distribución es "**uniforme**", lo cual significa que tienen la misma probabilidad de ser generado tanto el 0 como el 1, esto hace que los resultados de la salida cambien cada vez que se ejecuta el programa, simulando de esa manera el lanzamiento de una moneda al aire.

Es posible generar números aleatorios en un rango determinado. Por ejemplo; si se desea que los números aleatorios estén entre 50 y 150 la expresión siguiente puede ser de utilidad.

```
x=aleatorio.nextInt(101)+50;
```

Porque los números que genera están entre 0 y 100, pero al sumarle 50 realmente están en el rango de 50 y 150.

En el programa anterior se considera que si el valor generado es 1, entonces el resultado del lanzamiento es "Sol" y si es 0 es "Águila".

La diferencia principal entre **`while`** y **`do-while`**, es que en ésta última la condición está al final, lo que permite ejecutar al menos una vez el bloque de instrucciones contenidos en `do-while`.

Ejemplo 4.10. Una lavandería compró equipo que se va a depreciar en cierto lapso de tiempo por el método de "Saldo de doble declinación"; es decir; la depreciación de cada año está dada por la siguiente fórmula:

$$d = \frac{2 * precio}{vida}$$

Siendo "*precio*" el valor del equipo al comenzar el año, y "*vida*" los años en los cuales se deprecia el equipo. Escribir un programa para calcular la depreciación y el valor del equipo en cada año.

```
import java.util.Scanner;

public class depreciacion {
public static void main (String[] args){
  Scanner leer = new Scanner(System.in);

int a;
float d, precio, vida;

System.out.print ("Valor del equipo: ");
precio=leer.nextInt();
System.out.print ("Vida esperada (años): ");
vida=leer.nextInt();

a=0; d=0;
System.out.println ("Año   Depreciación   Valor ");
 do {
     System.out.printf ("%2d    %10.2f   %9.2f \n",a,d,precio);
     d=2*precio/vida;
     precio-=d;
     a++;
    } while (a<=vida);
   }
 }
```

```
run:
Valor del equipo: 7000
Vida esperada (años): 5
Año      Depreciación      Valor
 0              0.00     7000.00
 1           2800.00     4200.00
 2           1680.00     2520.00
 3           1008.00     1512.00
 4            604.80      907.20
 5            362.88      544.32
GENERACIÓN CORRECTA (total time: 8 seconds)
```

4.3.3. Desde (`for`).

Es posible elaborar cualquier programa donde se requieran ciclos con las estructuras iterativas `while` y `do-while`, pero la estructura repetitiva *`for`* es más compacta y sencilla de utilizar bajo ciertas condiciones. La forma general de `for` es:

```
for (inicio; condición; actualización) {
    ........
    Instrucciones;
    ........
}
```

La instrucción **for** requiere de una variable que tiene la finalidad de controlar el número de veces que se ejecuta el bloque de instrucciones. En "**inicio**" se coloca el valor con el cual inicia la variable. En "**condición**" se indica la proposición que deberá ser verdadera para que el ciclo continúe, en caso de ser falsa saldrá de dicho ciclo y finalmente en "**actualización**" se coloca una instrucción que tendrá la finalidad de modificar el valor de la variable con la finalidad de que después de ejecutar varias veces el bloque de instrucciones que están encerradas entre llaves, la condición sea falsa.

Ejemplo 4.11. Escribir un programa para leer un número entero positivo y encontrar el factorial de ese número.

```
import java.util.Scanner;
public class factorial {
public static void main(String[] args){
    Scanner leer = new Scanner(System.in);

int n;

System.out.print ("Valor de n: ");
n=leer.nextInt();
int fac=1;

for (int x=1; x<=n; x++)
    fac=fac*x;

 System.out.printf ("El factorial de %d es = %d \n",n,fac);
}
}
```

```
run:
Valor de n: 5
El factorial de 5 es = 120
GENERACIÓN CORRECTA (total time: 4 seconds)
```

En el programa anterior la variable que hace las veces de contador es x, su inicio es (x=1), la condición (x<=n) permite ejecutar el bloque de instrucciones que se encuentran entre llaves, mientras la condición sea verdadera, y el incremento de dicha variable en cada iteración es de uno en uno (x++).

Lo que ocurre es que al llegar a la instrucción **for**; cuando todavía no se ejecutan ninguna vez las instrucciones que se encuentran dentro del ciclo, la variable x toma el valor inicial (x=1), inmediatamente verifica si la condición es verdadera, en caso de que así sea, se ejecuta el bloque de instrucciones que están entre llaves después de la instrucción **for** (en este caso no fueron necesarias las llaves, porque es solamente una instrucción la que se ejecuta varias veces). Después de ejecutar las instrucciones que se encuentran dentro del ciclo, regresa a incrementar la variable (x++) y acto seguido verifica si con este nuevo valor de x sigue siendo verdadera la condición, en caso de ser así ejecuta nuevamente la instrucción o bloque de instrucciones que están dentro del ciclo, posteriormente regresa a incrementar el valor de x, verifica si la condición sigue siendo verdadera, en caso de ser así ejecuta el bloque de instrucciones y así sucesivamente: incrementa, verifica, ejecuta. Cuando la condición ya no se cumple se sale del ciclo para continuar con la ejecución del resto del programa.

Ejemplo 4.12. Se dice que un número es primo si solamente es divisible entre él mismo y la unidad. Escribir un programa para determinar si un número es primo o no, contando los divisores entre 1 y el mismo número. Imprimir los divisores que tiene e indicar si es primo o no.

```
import java.util.Scanner;
public class determinar {
public static void main(String[] args){
    Scanner leer = new Scanner(System.in);

int n, divisores, w;

System.out.print(" n: ");
n=leer.nextInt();

divisores = 0;
System.out.print ("Sus divisores son: ");

 for (w=1; w<n; w++){
   if (n % w==0){
      System.out.printf ("%2d,",w);
      divisores+=1;
    }
  }

System.out.println (n);
 if (divisores <=2)
   System.out.printf ("El %d es primo \n",n);
 else
   System.out.printf ("El %d no es primo \n",n);
 }
}
```

```
run:
 n: 20
Sus divisores son:  1, 2, 4, 5,10,20
El 20 no es primo
GENERACIÓN CORRECTA (total time: 3 seconds)
```

En el programa anterior se determinan e imprimen los divisores dentro del ciclo en forma horizontal a excepción del último divisor que se imprime fuera y se aprovecha la instrucción `System.out.println (n);` para saltar de línea.

Se recomienda usar el `for` cuando se sabe el número de veces que se ejecutarán las instrucciones que están dentro de la instrucción iterativa ya que ocupa menos espacio que `while` y `do-while`.

4.4 Ciclos anidados

En un programa es posible tener una instrucción dentro de otra y por lo tanto puede haber también instrucciones iterativas dentro de otra instrucción iterativa, esto permite crear programas con mayor potencia.

Ejemplo 4.13. El astrónomo y geógrafo del siglo XI a.C. Eratóstenes, estableció que para determinar si un número x es primo, no es necesario dividir x entre todos los divisores comprendidos entre 1 y x, que es suficiente probar los divisores que están entre 1 y $\sqrt{x}$. Si solamente hay un divisor en ese rango, entonces x es primo. Escribir un algoritmo para imprimir todos los números primos comprendidos entre 1 y n usando la afirmación de Eratóstenes.

```
import java.util.Scanner;

public class primos {
public static void main(String[] args){
    Scanner leer = new Scanner(System.in);

int n, divis, w, x;

System.out.print("n: ");
n=leer.nextInt();

x=1;
System.out.println("Números primos entre 1 y "+n);
while (x<=n){
 divis=0;
 for (w=1; w<=Math.pow(x, 1/2.0); w++){
    if (x%w==0)
      divis+=1;
    }
    if (divis<=1)
```

```
        System.out.println (x);
      x++;
    }
  }
}
```

```
run:
n: 18
Números primos entre 1 y 18
1
2
3
5
7
11
13
17
GENERACIÓN CORRECTA (total time: 3 seconds)
```

En el programa anterior se puede observar que dentro del ciclo **while** se encuentra el ciclo **for**. En la estructura iterativa **while** se generan los números enteros del 1 a n y en **for** se encuentran los divisores de cada uno de esos números, los que tengan 1 o menos divisores son primos. También se puede ver que ya no prueba todos los divisores de cada uno de los números, sino que solamente los que son menores o iguales a la raíz cuadrada de ese número. La raíz cuadrada en este caso se encuentra elevando a la un medio (1/2.0) el valor de x, usando el método `pow` que pertenece a la clase `Math`. Como se muestra en la línea de código:

```
for (w=1; w<=Math.pow(x, 1/2.0); w++)
```

Ejemplo 4.14. Escribir un programa para desplegar en la pantalla el menú del restaurant de comida Mexicana "Mexican food" que se muestra a continuación

Código	Descripción	Precio
1	Enchiladas	$30.00
2	Pozole	$20.00
3	Tamales	$21.00
4	Atole	$12.00
5	Café	$8.00

El programa deberá leer el código del producto y la cantidad de órdenes o productos consumidos por un cliente. Leerá información hasta que el código sea 999 y al final indicará la cantidad a pagar por el cliente. Preguntará si se desea leer otra cuenta, en caso afirmativo leerá los datos de ella, de lo contrario terminará.

```
import java.util.Scanner;
public class mexicana {
public static void main(String[] args){
Scanner leer = new Scanner(System.in);
  int cod, cant;
  double subtot=0, tot=0;
  char continua;

System.out.println ("Mexican food ");
System.out.println ("Código \t\t Descripción\t\tPrecio");
System.out.println ("   1 \t\t Enchiladas\t\t$30.00");
System.out.println ("   2 \t\t Pozole\t\t\t$20.00");
System.out.println ("   3 \t\t Tamales\t\t$21.00");
System.out.println ("   4 \t\t Atole\t\t\t$12.00");
System.out.println ("   5 \t\t Café\t\t\t $8.00");

cod=0;
do{
 while(cod!=999){
 System.out.print ("Código: ");
   cod= leer.nextInt();

 if (cod!=999){
  System.out.print ("Cantidad: ");
     cant= leer.nextInt();

  switch (cod)
  {
  case 1:
   subtot=cant*30;
   System.out.printf ("%d Enchiladas = %.2f\n",cant,subtot);
   break;
  case 2:
   subtot=cant*20;
   System.out.printf ("%d Pozoles = %.2f\n",cant,subtot);
   break;
  case 3:
   subtot=cant*21;
   System.out.printf("%d Tamales = %.2f\n",cant,subtot);
    break;
   case 4:
    subtot=cant*12;
    System.out.printf ("%d Atoles = %.2f\n",cant,subtot);
    break;
```

```
    case 5:
     subtot=cant*8;
     System.out.printf ("%d Cafés = %.2f\n",cant,subtot);
     break;
    default:
     System.out.println ("Código Inexistente\n");
     break;
    }
 tot+=subtot;
 }
}
  System.out.printf ("Total= %.2f\n",tot);
  System.out.print ("Otra cuenta s/n?:");
  String cadena= leer.next();  //lee una cadena
  //toma el primer caracter de la cadena
  continua=cadena.charAt(0);
  cod=0;
  tot=0;
 }while(continua!='n');
 }
}
```

```
run:
Mexican food
Código          Descripción        Precio
   1            Enchiladas         $30.00
   2            Pozole             $20.00
   3            Tamales            $21.00
   4            Atole              $12.00
   5            Café               $8.00
Código: 3
Cantidad: 4
4 Tamales = 84.00
Código: 2
Cantidad: 5
5 Pozoles = 100.00
Código: 999
Total= 184.00
Otra cuenta s/n?:n
GENERACIÓN CORRECTA (total time: 37 seconds)
```

Observar que el programa termina con la letra 'n' porque ya no hay más cuentas de clientes que leer, pero con la clase *import java.util.Scanner*; no es posible leer caracteres de teclado, de tal manera que se lee una cadena y posteriormente se asigna a una variable el primer caracter de esa cadena, con las siguientes instrucciones:

```
String cadena= leer.next();  //lee una cadena
continua=cadena.charAt(0);   //toma el primer caracter de
                             //la cadena
```

Se entiende que el programa termina siempre y cuando el primer caracter de la cadena sea 'n' en caso que se teclee por ejemplo un espacio y después la letra 'n' seguirá leyendo información. Los casos de excepción que se verán posteriormente.

Las cadenas (`String`) son una secuencia de caracteres o también se conocen como arreglos de caracteres, en donde cada caracter ocupa una posición. El primer caracter de la cadena está en la posición cero, el segundo en la posición uno y así sucesivamente como se muestra a continuación:

Cadena:	E	l		c	o	c	h	e		r	o	j	o
Posición:	0	1	2	3	4	5	6	7	8	9	10	11	12

Es posible conocer la longitud de una cadena usando la función `length()` de la siguiente manera:

```
int  longitud=cadena.length();
```

Ejemplo 4.15. Escribir un programa que lea una frase de teclado y que cuente e imprima el número de vocales que tiene dicha frase. Considerar también el conteo de letras minúsculas y mayúsculas.

```
import java.util.Scanner;
public class vocales {
    public static void main(String[] args){
    Scanner leer = new Scanner(System.in);

String cadena;
int longitud, pos=0, as=0, es=0,is=0,os=0, us=0;
char letra;

System.out.print("Frase: ");
cadena= leer.nextLine();
longitud=cadena.length();

while (pos<longitud){
 letra=cadena.charAt(pos);

switch (letra) {
    case 'a':
    case 'A':
             as++;
             break;
    case 'e':
    case 'E':
```

```
            es++;
            break;
    case 'i':
    case 'I':
            is++;
            break;
   case 'o':
   case 'O':
            os++;
            break;
   case 'u':
   case 'U':
            us++;
            break;
    }
    pos++;
   }
System.out.printf("Letras a=%d Letras e= %d Letras i= %d\n",as,es,
is);
System.out.printf ("Letras o= %d  Letras u = %d\n",os,us);
 }
}
```

```
run:
Frase: El elefante es blanco
Letras a= 2  Letras e = 5  Letras i = 0
Letras o= 1  Letras u = 0
GENERACIÓN CORRECTA (total time: 10 seconds)
```

En el programa anterior se encuentra la longitud de la cadena y posteriormente se visitan cada una de las localidades de los caracteres de la cadena, si es una vocal se cuenta independientemente si es mayúscula o minúscula. Al final se imprimen los resultados. Observar que funciona para letras mayúsculas y minúsculas ya que la instrucción `switch` solamente tiene un `break` para ambas letras. También se puede ver que las cadenas se encierran entre comillas y los caracteres entre apóstrofes.

4.5. Manipulación de cadenas

Es posible llevar a cabo varias operaciones con los métodos de la clase `String` (cadena). A continuación se tienen algunos de ellos, que pueden ser de utilidad. Pero el lenguaje Java tiene muchos más métodos que permiten la manipulación de cadenas. En la tabla siguiente considerar que cad1, cad2 y cad3 son datos tipo `String`

Instrucciones	Explicación
cad1="El perro ladra"; **int** longi=cad1.**length**();	Calcula la longitud de cad1. longi=14
cad1="El león no es como lo pintan"; **char** letra=cad1.**charAt**(5);	Extrae un copia del carácter que está en la posición 5 y se lo asigna a la variable letra='ó'
cad1="Pera madura"; **StringBuilder** invierte=**new StringBuilder**(cad1); cad2=invierte.**reverse**().**toString**();	Se crea el objeto invierte de la clase `StringBuilder` que permite invertir cad1. cad2="arudam areP"
cad1="Juan Colorado"; cad2=cad1.**substring**(2,7);	Copia los caracteres de cad1, de la posición 2 a la 7. cad2="an Co"
cad1="josé"; cad2="José"; **if** (cad1.**equals**(cad2))	Compara si cad1 y cad2 son iguales, el resultado es booleano (true o false). En este caso es **false** (falso), porque aunque es el mismo nombre en uno se escribió exclusivamente con letras minúsculas y en el otro la primer letra es mayúscula.
cad1="josé"; cad2="José"; **if** (cad1.**equalsIgnoreCase**(cad2))	Compara nombres, pero en este caso es **true** (verdadero) porque considera iguales la misma letra mayúscula o minúscula.
cad1="Hasta"; cad2="luego"; cad3=cad1.**concat**(cad2);	Concatena al final de cad1 la cadena cad2. cad3="Hastaluego"
cad1=" 478"; cad1=cad1.**trim**();	Elimina los espacios que están al principio o al final de la cadena. cad1="478"
cad1=" 478"; **int** n=**Integer.parseInt**(cad1.**trim**());	Elimina los espacios en blanco y convierte la cadena en un entero. n=478
cad1="3.697"; **double** n; n=**Double.valueOf**(cad1).**doubleValue**();	Convierte la cadena a un dato tipo `double`. n=3.697
int n=347; **String** cad1=**String.valueOf**(n);	Convierte un número en cadena. cad1="347"

Ejemplo 4.16. Escribir un programa para leer una cadena de teclado, invertir dicha cadena, comparar la cadena inicial y la cadena invertida con la finalidad de determinar si son iguales, copiar un rango de caracteres de una cadena, comparar si dos cadenas son iguales con o sin letras mayúsculas, concatenar dos cadenas y convertir una cadena a número.

```
import java.util.Scanner;
public class variascadenas {
public static void main(String[] args){
 Scanner leer = new Scanner(System.in);

 String cad1, cad2, cad3;

System.out.print("Dame una cadena: ");
cad1=leer.nextLine();

//invierte cad1 y se la pasa a cad2
StringBuilder invierte=new StringBuilder(cad1);
cad2=invierte.reverse().toString();
System.out.println ("Primera cadena (cad1)= "+cad1);
System.out.println ("Cadena invertida (cad2)= "+cad2);

//copia una parte de cad1 a cad3
cad3=cad1.substring(3,9);
System.out.println ("Del caracter 3 al 9 de cad1= "+cad3);

//compara haciendo diferencia entre mayúsculas y minúsculas
if (cad1.equals(cad2))
    System.out.println ("Son iguales: "+cad1+" y "+cad2);

// compara ignorando mayúsculas
if (cad1.equalsIgnoreCase(cad2))
System.out.println ("Son iguales ignorando mayúsculas: "+cad1+" y
"+cad2);

//Concatena cad1 y cad2 y las asigna a cad3
String cad4=cad1.concat(cad2);
System.out.println ("Concatenadas cad1 y cad2: "+cad4);

//convierte una cadena en numero
String dato1=" 14 ";
String dato2="135.72";

int num1=Integer.parseInt(dato1.trim());
double num2=Double.valueOf(dato2).doubleValue();
System.out.println ("Resultado de multiplicarlos= "+num1*num2);
 }
}
```

```
run:
Dame una cadena: Hola m aloh
Primera cadena (cad1) = Hola m aloh
Cadena invertida (cad2) = hola m aloH
Del caracter 3 al 9 de cad1= a m al
Son iguales ignorando mayúsculas: Hola m aloh y hola m aloH
Concatenadas cad1 y cad2: Hola m alohhola m aloH
Resultado de multiplicarlos= 1900.08
GENERACIÓN CORRECTA (total time: 7 seconds)
```

En el programa anterior se convirtió una cadena a número, pero se da por hecho que los caracteres de la cadena son números, si esto no fuera así; por ejemplo que tuviera letras o espacios intermedios. Es conveniente utilizar adicionalmente otros métodos que permitan validar la cadena antes de hacer la conversión a cantidad numérica, para asegurar que los caracteres de la cadena son caracteres válidos de una cantidad numérica.

Ejemplo 4.17. Escribir un programa para leer dos cadenas y escribirlas en orden alfabético.

```
import java.util.Scanner;

public class comparanombres {
public static void main(String[] args){
    Scanner leer = new Scanner(System.in);

String cad1, cad2, cad3;

System.out.print ("Primer nombre: ");
cad1=leer.nextLine();
System.out.print ("Segundo nombre: ");
cad2=leer.nextLine();

System.out.println ("En orden alfabético: ");
 if ( cad1.compareTo(cad2)<=0)
      System.out.println (cad1+"\n"+cad2);
   else
     System.out.println (cad2+"\n"+cad1);
  }
}
```

```
Primer nombre: Martínez Sánchez Juan
Segundo nombre: Bueno Zarate Pedro
En orden alfabético:
Bueno Zarate Pedro
Martínez Sánchez Juan
GENERACIÓN CORRECTA (total time: 41 seconds)
```

En el programa anterior, con la instrucción siguiente se determina alfabéticamente la posición de las cadenas: `cad1` y `cad2`.

```
if ( cad1.compareTo(cad2)<=0)
```

Se están comparando las cadenas `cad1` con `cad2` y el resultado puede ser una cantidad negativa, positiva o cero. Si `cad1` es alfabéticamente menor que `cad2`, la cantidad es negativa, si `cad1=cad2` la cantidad es cero y si `cad1>cad2` la cantidad es positiva. Ejemplo:

```
String cad1="Juan", cad2="Ana María", cad3="Carlos", cad4="Juan";
int n;
n=cad1.compareTo(cad2);        //n es positiva porque
                               //alfabéticamente "Juan" > "Ana María"
n=cad2.compareTo(cad3);        //n es negativa "Ana María" < "Carlos".
n=cad1.compareTo(cad4);        //n=0 porque alfabéticamente "Juan"="Juan".
```

4.6. Diseño e implementación de funciones y métodos

Un método es un bloque de instrucciones que tiene un nombre, que opcionalmente puede recibir parámetros o argumentos, que se utiliza para realizar algo y que opcionalmente puede devolver un valor de algún tipo de dato conocido. La sintaxis de los métodos es:

```
modificador tipo nombre(tipo par1, tipo par2,…, tipo parn)
{
   .....
   Instrucciones
   .....
 }
```

Donde:

`modificador`: Llamado comúnmente modificador de acceso, es el modo en que podrá ser visto el módulo dentro del programa. Los modificadores de acceso más comunes son: `public`, `static`, `private`, `protected` y `default`.

`tipo`: Es el tipo de dato (`int`, `float`, `double`, `boolean`, etc) que regresa el método después de ejecutar las instrucciones que están dentro de él. Cuando el método no regresa ningún valor, en lugar del tipo de dato, se le pondrá la palabra reservada *`void`*, la cual indica que no se devolverá ningún valor.

`nombre`: Es el nombre asignado al módulo o función.

`par1, par2,.., parn`: parámetros para recibir los valores de los argumentos al momento de invocar el método. Estos parámetros deberán ser definidos de algún tipo de dato (`int`, `double`, `String`, `char`, etc.).

4.6.1. Funciones

Cuando un método recibe uno o varios argumentos y devuelve un solo valor recibe el nombre de función.

Las funciones tienen el mismo formato que los métodos, porque son métodos con ciertas características, pero adicionalmente deben tener la línea `return` para regresar el valor, como se muestra en la siguiente función "`resta`".

```
static float resta(float a, float b) {
   float c;

   c=a-b;
   return c;
  }
```

La función tiene el modificador `static` lo cual permite que la función sea reconocida en todos los métodos y clases del programa sin necesitar de un objeto, pero no en otros programas diferentes. Los valores que reciben `a` y `b` son del tipo `float` y el valor que regresa `c` también es `float` . Se puede observar que lo que hace esta función, es restar `b` al valor de `a`. Una llamada a esta función podría ser:

```
float x=8, y=5, z;
z=resta(x,y);
```

Al invocar la función con `z=resta(x,y);` una copia del valor de la variable "`x`" se manda al primer parámetro de la función "`a`", de la misma manera que el parámetro "`b`" recibe el valor de la variable "`y`" . Se realiza la resta y regresa el resultado por medio de la instrucción `return c;` por lo tanto la variable "`z`" recibe el resultado de la resta.

Después de ejecutar la función se destruyen los valores de los parámetros de la función `a` y `b`, además del valor de la variable local `c`.

Es importante mencionar que el número de parámetros de la función debe coincidir con la posición, número y tipo de dato de las variables al llamar la función. Ejemplo:

```
String nombre="Juan";
int edad=20, r;

r= azul(nombre,edad);

static int azul(String w, int edad) {
   int x;
    x= w.length()*edad;
   return x;
 }
```

En el ejemplo anterior la función `azul`, calcula la longitud del `nombre` que previamente se le mandó a la variable `w`, y multiplica el resultado por la `edad` para posteriormente devolver el resultado de

la multiplicación por medio de `return x,` mismo que se asigna finalmente a la variable `r`. Observar que `nombre` y `w` son de tipo `String` porque manejan la misma información y la variable `edad` es del mismo tipo del parámetro `edad`, con lo cual se muestra que pueden tener el mismo nombre los parámetros y las variables, pero no ocupan el mismo espacio en memoria ya que al salir de la función, se destruyen los espacios de memoria de los parámetros `w` y `edad`, además de las variables locales de la función, que en este caso solamente se tiene a `x`.

Todos los programas tienen funciones estándar para realizar operaciones muy comunes. En Java la librería *`Math`* tiene varias funciones, entre las cuales se pueden citar a:

`sin` `(x)`	Encuentra el seno de `x`.
`max` `(x,y)`	Valor más grande entre `x` y `y`.
`pow` `(x,y)`	Eleva `x` a la potencia `y` (x^y).
`sqrt` `(x)`	Raíz cuadrada de `x`.

Al invocar las funciones anteriores, se deberá anteponer el nombre de la librería *`Math`* como se muestra a continuación.

```
y=Math.sqrt (x);  //Calcula la raíz cuadrada de x y el resultado se asigna a la
                  //variable y.
```

Observar que cuando se invoca una función, generalmente el valor que devuelve dicha función se asigna a una variable o se imprime directamente, pero cuando se llama una función siempre se hace algo con el valor que regresa. Esto implica que por lo general las funciones son parte de una instrucción.

Es posible programar funciones que realicen actividades que no llevan a cabo las funciones estándar, permitiendo de esa manera, enriquecer el lenguaje de programación.

Ejemplo 4.18. Escribir un programa para leer tres valores enteros de teclado y determinar por medio de una función el mayor de ellos.

```
import java.util.Scanner;
public class determinamayor {
public static void main(String[] args){
    Scanner leer = new Scanner(System.in);

    int a, b, c, resultado;

    System.out.print("a: "); a=leer.nextInt();
    System.out.print("b: "); b=leer.nextInt();
    System.out.print("a: "); c=leer.nextInt();

    resultado=mayor(a,b,c);    //Se invoca la función mayor

    System.out.printf("El mayor es = %d\n",resultado);
    }  // Fin de programa principal
```

```
    //Función para encontrar el mayor de tres cantidades
    static int mayor(int x, int y, int z){
     int m=x;

     if (x>=y && x>=z)
         m=x;
     else if (y>=x && y>=z)
             m=y;
     else if (z>=x && z>=y)
             m=z;

    return m; //Devuelve el valor encontrado por la función
    } //Fin de función mayor
  }  // Fin de programa
```

```
run:
a: 31
b: 72
a: 8
El mayor es = 72
GENERACIÓN CORRECTA (total time: 19 seconds)
```

Los métodos y/o funciones se colocan después del programa principal (`main`) y antes de cerrar el programa.

En el programa principal se leen de teclado los tres números `a, b` y `c`. Al llamar la función por medio de la instrucción `resultado=mayor(a,b,c);` se pasan los valores de estas variables a los parámetros `x, y, z.` En la función `mayor,` el más grande de los tres datos se asigna a la variable local `m`, para finalmente enviar dicho valor por medio de **`return`** `m;` al lugar donde fue invocada la función y asignarlo a la variable `resultado`.

Ejemplo 4.19. Escribir un programa para calcular el número de combinaciones en que se pueden seleccionar r objetos, de un universo de n objetos diferentes, sabiendo que el número de combinaciones está dada por:

$$\binom{n}{r} = \frac{n!}{r!(n-r)}$$

Hacerlo:

a) Sin usar funciones.
b) Usando una función para calcular el factorial.

```
//Solución sin usar funciones.
import java.util.Scanner;
```

```
public class numerodecombinaciones {
public static void main(String[] args){
    Scanner leer = new Scanner(System.in);

int n,r, fact, i, nfact, rfact, combina;

System.out.print ("Número de objetos n: ");
n=leer.nextInt();
System.out.print("Tamaño de los grupos r: ");
r=leer.nextInt();

//Calcula el factorial de n
fact=1;
for (i=1; i<=n; i++){
    fact*=i;
    }
nfact=fact;

//Encuentra el factorial de r
fact=1;
for (i=1; i<=r; i++){
    fact*=i;
    }
rfact=fact;

combina=nfact/(rfact*(n-r));
System.out.printf("Número de combinaciones= %d\n",combina);
  }
 }
```

```
run:
Número de objetos n: 5
Tamaño de los grupos r: 3
Número de combinaciones= 10
GENERACIÓN CORRECTA (total time: 8 seconds)
```

En el programa anterior se utilizó un bloque de instrucciones para calcular el factorial de n y después otro bloque diferente para encontrar el factorial de r. Una vez que se tiene el factorial de ambos se encuentra el número de combinaciones. En este programa toda la información está en el programa principal.

```
/*Solución usando una función para encontrar el factorial de n y
r */
```

```
import java.util.Scanner;
public class numerodecombinaciones {
public static void main(String[] args){
    Scanner leer = new Scanner(System.in);

int n, r, combina;
System.out.print ("Número de objetos n: ");
n=leer.nextInt();
System.out.print ("Tamaño de los grupos r: ");
r=leer.nextInt();

combina=fact(n)/(fact(r)*(n-r));

System.out.printf ("Número de combinaciones= %d\n",combina);
}     //Finaliza programa principal

//Función para calcular el valor de la función de n y r
  static int fact(int x){
   int i, f=1;
    for (i=1; i<=x; i++){
       f*=i;
    }
    return f;
    }    // finaliza función fact
}   //Finaliza clase numerodecombinaciones
```

La salida de este programa es exactamente la misma que la del programa anterior, con la diferencia que utiliza una sola función para encontrar el factorial de n y también el de r. No fue necesario guardar el factorial de n y de r en una variable como se hizo en el programa anterior ya que directamente se calcularon el número de combinaciones por medio de la instrucción:

```
combina=fact(n)/(fact(r)*(n-r));
```

Al hacer la llamada de `fact(n)` y `fact(r)` se encuentran los factoriales de n y r respectivamente, mismos que se utilizan para encontrar el número de combinaciones. Esto hace que el programa sea más compacto, ya que se aprovecha el código de la función para encontrar el factorial de dos cantidades diferentes. Lo mismo ocurre con las funciones de la librería `.Math` para encontrar `sin(x)`, `cos(x)`, `pow(x,y)`, etc. Las cuales permiten encontrar el valor de funciones trigonométricas de cualquier valor "x", o elevar a la potencia "y" cualquier valor de "x" (x^y).

4.6.2. Métodos

Se dijo anteriormente que un método que recibe valores y devuelve un solo resultado recibe el nombre de función. Pero cuando no recibe datos, regresa más de un valor o simplemente no regresa valor alguno, se le llama simplemente **método**.

Los métodos se usan en la programación para dividir un programa grande en pequeños subprogramas con la finalidad de hacer más claros y sencillos los sistemas, aprovechando lo que realiza un método en el momento y lugar del programa cuando sea requerido.

Posiblemente se pregunten, ¿por qué usar métodos si los programas en la materia son pequeños? Recordar que en la práctica la mayoría de los sistemas computacionales son grandes y complejos, que muchas veces no son desarrollados por una sola persona, sino que requieren de un grupo de programadores, que se dividen las actividades del sistema y posteriormente se integra el trabajo de todos, en un solo sistema. Si el sistema falla, no es necesario revisar todo el programa sino solamente la parte que está funcionando mal, por ejemplo, si el sistema no elabora correctamente una factura, solamente se revisa el método que se encarga de la elaboración de facturas, pero no se revisan los métodos que se encargan de dar altas o bajas de productos, impresión de nómina, contabilidad, etc.

Ejemplo 4.20. Escribir un programa que lea un nombre en el programa principal, que imprima el nombre leído y posteriormente que llame un método para imprimir el mensaje "es un nombre bonito", enseguida llamar otro método que imprima el mensaje "Me gustaría llamarme " y finalmente imprimir nuevamente el nombre que se ha leído con anterioridad.

```
import java.util.Scanner;
public class sin_parametros {
public static void main(String[] args){
    Scanner leer = new Scanner(System.in);

    String nombre;

    System.out.print ("¿Cómo te llamas?: ");
    nombre=leer.nextLine();

    System.out.print (nombre);
    primermetodo();            //Llama al método 1
    segundometodo();           // Llama el otro método
    System.out.println (nombre);
    }

    //Método1
    static void primermetodo(){
     System.out.println(" es un nombre bonito");
    }

    //Método 2
    static void segundometodo(){
     System.out.print ("Me gustaría llamarme ");
    }
  }
```

```
run:
¿Cómo te llamas?: Alicia
Alicia es un nombre bonito
Me gustaría llamarme Alicia
GENERACIÓN CORRECTA (total time: 16 seconds)
```

Observa que los métodos del programa anterior no tienen parámetros, sin embargo es necesario el paréntesis aunque esté vacío, porque es parte del lenguaje. Es posible ver que no regresan ningún valor, por lo tanto no tienen una instrucción `return` y en lugar del tipo de dato del método, se colocó la palabra `void` indicando que se trata de un método que no regresa valor alguno.

Ejemplo 4.21. Escriba un programa que realice lo mismo del programa anterior, pero ahora usando métodos que tengan como parámetro la variable `nombre`.

```
import java.util.Scanner;
public class conparametros {
public static void main(String[] args){
    Scanner leer = new Scanner(System.in);

    String nombre;

    System.out.print("¿Cómo te llamas?: ");
    nombre=leer.nextLine();

    primermetodo(nombre);  //Llama al método 1
    segundometodo(nombre);  // Llama al otro método
    }

    //Método 1
    static void primermetodo(String nombre){
     System.out.println (nombre+" es un nombre bonito");
    }
    //Método 2
    static void segundometodo (String nombre){
     System.out.println ("Me gustaría llamarme "+nombre);
    }
}
```

La salida de este programa es exactamente la misma del programa anterior, pero ahora enviando la variable `nombre` como argumento, para que se imprima juntamente con el mensaje correspondiente en cada uno de los métodos.

Ejemplo 4.22. Escribir un programa para leer un número entero positivo n. Posteriormente desplegar en la pantalla en forma horizontal n veces dicho número, en la línea siguiente desplegar horizontalmente (n−1) veces el número (n−1) y así sucesivamente hasta llegar a una vez el número 1. Después de

imprimir dos veces el 2, tres veces el 3 y así sucesivamente hasta imprimir horizontalmente en una línea n veces n. Hacer dicho programa:

a) Sin usar métodos.

b) Usando métodos con parámetros.

c) Usando métodos sin parámetros.

```
import java.util.Scanner;
public class imprime_alas {
public static void main(String[] args){
    Scanner leer = new Scanner(System.in);

int n, i, digito;

System.out.print ("Valor de n: ");
n=leer.nextInt();

//Imprime las filas del n al 1
digito=n;
while (digito>=1){
    i=1;
    while (i<=digito){
      System.out.print (digito);
      i++;
      }
    System.out.println();  //Salta de línea
    digito--;
    }
//Imprime las filas del 2 al n
digito=2;
while (digito<=n){
    i=1;
    while (i<=digito){
      System.out.print (digito);
      i++;
      }
    System.out.println();  //Salta de línea
    digito++;
    }
   }
 }
```

```
run:
Valor de n: 4
4444
333
22
1
22
333
4444
GENERACIÓN CORRECTA (total time: 4 seconds)
```

En el programa anterior todo el código está en el programa principal, de tal manera que no se utilizaron métodos para segmentar el programa. La impresión es en dos partes, por medio de ciclos anidados ya que se encuentra un `while` dentro de otro `while`. Los dígitos se imprimen en forma horizontal por medio de la instrucción:

```
System.out.print (digito);
```

Pero se cambia de línea después de haber impreso todos los dígitos correspondientes a ese renglón por medio de la instrucción:

```
System.out.println();
```

```
/*Solución usando métodos para imprimir la información con méto-
dos*/
import java.util.Scanner;
public class imprimealas {
    public static void main(String[] args){
    Scanner leer = new Scanner(System.in);

int n;

System.out.print ("Valor de n: ");
n=leer.nextInt();

 haciabajo(n);   //Imprime líneas del n al 1
 haciarriba(n);  //Imprime líneas del 2 al n

 }  //fin de programa principal

//Método para imprimir las filas de n a 1
static voidhaciabajo (int n){
int i, digito=n;
```

```
while (digito>=1){
  i=1;
  while (i<=digito){
    System.out.print (digito);
    i++;
    }
  System.out.println();  //Salta de línea
  digito--;
  }
}

  //Imprime las filas del 2 al n
  static voidhaciarriba(int x){
    int i, digito=2;
    while (digito<=x){
        i=1;
        while (i<=digito){
          System.out.print (digito);
          i++;
          }
      System.out.println();  //Salta de línea
      digito++;
      }
    }
  }
```

Observar en la solución anterior que la variable n se define en el programa principal (`main`), pero también se define como parámetro en el método `haciabajo (int n)`, en el otro método al parámetro se le llama *x* para mostrar que no es necesario que el parámetro se llame igual que la variable con la que se envía la información, pero sí debe coincidir en la posición y tipo de dato. Las variables **`i`** y **`digito`** se definen en cada uno de los métodos, porque son variables locales conocidas solamente en ese método, una vez que la computadora sale del método se olvida de ellas.

```
//Solución usando métodos sin parámetros.
import java.util.Scanner;

public class sinparametros {
public static int n;   //Variable conocida en toda la clase
public static void main(String[] args){
    Scanner leer = new Scanner(System.in);

System.out.print("Valor de n: ");
n=leer.nextInt();
```

```
 haciabajo();   //Imprime líneas del n al 1
 haciarriba();  //Imprime líneas del 2 al n

 }  //fin de programa principal

//Método para imprimir las filas del n al 1
static void haciabajo(){
int digito=n;
while (digito>=1){
   int i=1;
    while (i<=digito){
      System.out.print (digito);
      i++;
     }
  System.out.println();  //Salta de línea
  digito--;
  }
}

 //Imprime las filas del 2 al n
  static void haciarriba(){
    int digito=2;
    while (digito<=n){
      for (int i=1; i<=digito; i++){
        System.out.print (digito);
        }
      System.out.println();  //Salta de línea
      digito++;
      }
    }
   }
```

Observar que con la línea: `public static int n;` se define la variable `n` para que sea conocida en toda la clase, tanto en el programa principal `main` como en todos los métodos que pertenezcan a dicha clase, esto permite usarla en cualquier parte de la clase.

Se aprovecha también este programa para observar que es posible definir las variables en cualquier parte del programa considerando el siguiente alcance para su definición:

a) Si una variable se define antes del programa principal, dicha variable será conocida en el programa principal y todas las clases y métodos de ese programa. Por ejemplo. La línea que está antes del programa principal en el programa anterior.

```
public static int n;
```

b) Los parámetros definidos en los métodos, son conocidos en todo el cuerpo del método. Por ejemplo los parámetros `p` y `q` en el siguiente método son únicamente conocidos en ese método.

```
Static void metodoA(int p, String q){
  ……
  Instrucciones
  ……
  }
```

c) Las variables locales de un método son conocidas a partir de la posición donde se definen. Por ejemplo las variables locales `tot` y `res` son conocidas del lugar donde están definidas hasta el resto del método.

```
Static void metodoB(int x, float w){

  int tot;
  Instrucciones
  float res;
  ……
  }
```

c) Las variables que se definen en el encabezado de una instrucción **`for`** son conocidas en el cuerpo de la misma estructura iterativa **`for`**. Lo mismo ocurre con las demás instrucciones iterativas **`while`** y **`do-while`**.

```
for (int i=1; i<=digito; i++){
 System.out.print (digito);
 }
```

Ejemplo 4.23. Escribir un programa para leer el nombre y fecha de nacimiento de una persona, encontrar con esa información el Registro Federal de Causantes (RFC). Considerar que primero se leen los apellidos y después el nombre en una sola cadena y que el año (aaaa) mes (mm) y día (dd) son datos tipo cadena. Recordar que el RFC está integrado por las dos primeras letras del primer apellido, la primera del segundo y la primera del nombre. Considerar también que si la persona tiene más de un nombre, se deberá tomar la primera letra del primer nombre siempre y cuando ese primer nombre no sea "José" o "María" ya que en tal caso se tomará la primer letra del segundo nombre. Ejemplo: si el nombre leído es: TAPIA MENDES JOSE SANTIAGO, Día: 23, Mes: 3 Año: 1989 su RFC es como se muestra a continuación.

RFC: TAMS890323

Utilizar métodos para separar los apellidos y nombre(s). Usar también métodos para concatenar en el RFC los caracteres requeridos.

```
import java.util.Scanner;
public class reg_fed_de_causantes {
```

```
public static String rfc;
public static void main(String[] args){
     Scanner leer = new Scanner(System.in);
     String nombre, dia, mes, anio;

      System.out.print ("Nombre: ");
      nombre=leer.nextLine();
      System.out.print ("Día: ");
      dia=leer.next();
      System.out.print ("Mes: ");
      mes=leer.next();
      System.out.print ("Año: ");
      anio=leer.next();

      //Convierte a mayúsculas el nombre
      nombre=nombre.toUpperCase();

      separacadenas(nombre);
      digitosanio(anio);
      digitomes(mes);
      digitodia(dia);
      System.out.println ("RFC: "+rfc);
}   //Fin del programa principal

/*Método para concatenar las primeras dos letras del primer ape-
llido */
    static void delpaterno(String x){
     rfc=x.substring(0,2);
     }

/*Método para concatenar la primer letra del segundo apellido*/
    static void delmaterno(String x){
      rfc=rfc.concat(x.substring(0,1));
      }

/*Método para concatenar la letra del primer o segundo nombre*/
   static void letranombre(String x, String y){
       if (y.equals("")){
       rfc=rfc.concat(x.substring(0,1));
       }
       else{
         if (x.equals("JOSE") || x.equals("MARIA"))
         rfc=rfc.concat(y.substring(0,1));
        else
```

```
        rfc=rfc.concat(x.substring(0,1));
         }
       }

  //Método para separar apellidos y nombres
   static void separacadenas(String x){
      int pos, esp=0, longitud;
      int lim1=0, lim2=0;
      String paterno="", materno="", nom1="", nom2="";
      char letra;

     //cuenta los espacios para separar cadenas
      pos=0;
      longitud=x.length();
      while (pos<longitud){
          letra=x.charAt(pos);
           if (letra== ' ' ){
            esp++;
            if (esp==1){
                paterno=x.substring(0,pos);
                lim1=pos;
                }
            else if (esp==2){
                 materno=x.substring(lim1+1, pos);
                 lim2=pos;
                 }
            else if (esp==3){
                nom1=x.substring(lim2+1, pos);
                nom2=x.substring(pos+1,longitud);
                }
            }
        pos++;
      }

      if (esp==2){
       nom1=x.substring (lim2+1,longitud);
       }
      delpaterno(paterno);            //Llama método
      delmaterno(materno);            //Invoca método
      letranombre(nom1,nom2);         //Invoca método
      }

//Método para concatenar los dos últimos caracteres del año
static void digitosanio(String a){
```

```
      rfc=rfc.concat(a.substring(2,4));
      }

  //Método para concatenar los dígitos del mes
    static void digitomes(String m){
    int longitud;
     longitud=m.length();

     if (longitud<2){
     rfc=rfc.concat("0"); //Concatena el 0
     rfc=rfc.concat(m); //Concatena el número de mes
     }
     else{
      //Concatena los dos dígitos del mes
      rfc=rfc.concat(m);       }
     }

    //Método para concatenar dígitos del día
    static void digitodia(String d){
     int longitud;
     longitud=d.length();

     if (longitud<2){
     rfc=rfc.concat("0"); //Concatena el 0
     rfc=rfc.concat(d); //Concatena el número de día
     }
     else{
      rfc=rfc.concat(d); //Concatena el número de día
      }
    }
 }
```

```
run:
Nombre: Carranza Pérez Juan Antonio
Día: 15
Mes: 8
Año: 1992
RFC: CAPJ920815
GENERACIÓN CORRECTA (total time:  6
```

En el programa anterior primeramente se separan los apellidos y nombre(s) contando los espacios que existen entre ellos. Observar que es posible llamar de un método a otro método diferente como ocurre en: `separacadenas(String x)`, en donde se invoca a los métodos `delpaterno(paterno)`,

`delmaterno(materno)` y `letranombre(nom1,nom2)`, para concatenar al RFC las dos primeras letras del apellido paterno, la primera del apellido materno y la primer letra del primer nombre si su nombre no es "José" o "María" o bien la letra del segundo nombre en caso de la persona tenga dos nombres y el primero sea "José" o "María". Finalmente se invoca a los métodos: `digitosanio(String a)`, `digitomes(String m)` y `digitodia(String d)`, para concatenar al RFC los dígitos del año, mes y día respectivamente.

Ejemplo 4.24. Gran cantidad de funciones han sido definidas usando series infinitas. En todos los casos la aproximación aumenta a medida que se toman en cuenta más términos de la serie. Escribir un programa para calcular el valor real de e^x. Aproximar ese valor usando la serie que se muestra a continuación. El programa leerá de teclado el valor de x y el error permitido. Terminará cuando la diferencia entre la función e^x real y la aproximada sea menor o igual al error leído de teclado. Imprimir en cada paso la iteración, el término de la sumatoria, el valor aproximado de e^x y la diferencia que existe con respecto al valor real, hasta llegar a la exactitud indicada.

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots.$$

Usar una función para calcular el factorial y otra para calcular cada uno de los términos que se van sumando.

```
import java.util.Scanner;
public class con_funciones {
public static void main(String[] args){
    Scanner leer = new Scanner(System.in);

    double x, err, resul;

System.out.print ("Valor de x: ");
x=leer.nextFloat();
System.out.print ("Error permitido: ");
err=leer.nextFloat();

resul=Math.exp(x);
System.out.printf("ex real= %7.4f\n",resul);
tabula(resul,x,err);
}   //Fin de programa principal

/*Método para tabular la tabla. Hace llamadas a la función
term(x, n)para encontrar cada término de la sumatoria*/

static void tabula(double resul, double x, double err){
  int n=1;
  double t, sum, e;
  sum=0;
  System.out.println ("  n \tTérmino \tex(Aprox) \tDiferencia");
  t=1;
```

```
  sum=sum+t;
  e=Math.abs(resul-sum);
  while (e>=err){
    System.out.printf ("%4d \t%7.4f \t%7.4f \t%7.4f\n",n, t,sum,e);
      t=term(x,n);
      sum=sum+t;
      e=Math.abs(resul-sum);
      n++;
      }
  }

//Función para encontrar cada término de la sumatoria
  static double term(double x, int n){
  return Math.pow(x, n)/fact(n);
  }

//Función para encontrar el factorial de n
  static double fact(int n){
  double f=1;

  for (int x=1; x<=n; x++)
    f=f*x;
  return f;
  }
}
```

```
run:
Valor de x: 3.4
Error permitido: 0.01
ex real= 29.9641
   n      Término      ex(Aprox)      Diferencia
   1      1.0000        1.0000        28.9641
   2      3.4000        4.4000        25.5641
   3      5.7800       10.1800        19.7841
   4      6.5507       16.7307        13.2334
   5      5.5681       22.2987         7.6654
   6      3.7863       26.0850         3.8791
   7      2.1456       28.2306         1.7335
   8      1.0421       29.2727         0.6914
   9      0.4429       29.7156         0.2485
  10      0.1673       29.8829         0.0812
  11      0.0569       29.9398         0.0243
GENERACIÓN CORRECTA (total time: 8 seconds)
```

En el programa anterior se tiene el método: `tabula(double resul, double x, double err)` y las funciones: `term(double x, int n)` y `fact(int n)`. El método `tabula` se encarga de la tabulación de la tabla y llama a la función `term` para encontrar cada uno de los términos de la sumatoria. A su vez la función `term` llama a la función `fact` para encontrar el factorial de cada uno de los términos.

4.6.3. Recursividad

La recursividad es una técnica de programación en donde un método se llama a sí mismo desde su propio código.

La recursividad y la iteración están muy relacionadas, ya que un programa recursivo se puede escribir en forma iterativa. Los programas iterativos utilizan las instrucciones repetitivas: **`while`**, **`do-while`** y **`for`** para ejecutar varias veces un bloque de sentencias y los programas recursivos se invocan a sí mismos apoyándose en una instrucción selectiva (**`if-else`**) y una pila para guardar expresiones que no se pueden evaluar de manera inmediata, hasta que se encuentra un valor con el que sí se pueda encontrar el resultado de la expresión. Al primer punto en donde es posible evaluar la expresión se le conoce como "***punto de retorno***", porque a partir de ese momento se comienzan a sacar de la pila las expresiones que estaban pendientes de calcular.

Se toma la expresión que está en la cima de la pila y se evalúa, después se retira de la pila la siguiente expresión que está más arriba para encontrar su resultado, y así sucesivamente se van sacando y evaluando todas las expresiones que se guardaron en la pila, hasta que la pila queda completamente vacía.

En computación la estructura de datos conocida como "***pila***" tiene el mismo principio de una pila de platos, en donde se toma el plato que se encuentra encima de la pila cuando se desea usar.

Se debe tener cuidado al programar métodos recursivos ya que al ejecutarlos podrían prolongarse indefinidamente y no terminar nunca.

Todos los métodos recursivos se pueden programar de forma iterativa, aunque algunas veces la solución es más complicada, sin embargo no todos los métodos iterativos se pueden resolver en forma recursiva ya que deben reunir las siguientes características:

a) Deben tener uno o más puntos base o retorno.

a) Tienen un caso recursivo que permite llamarse a sí mismo.

El punto de retorno es donde se especifica en forma explícita la solución del problema para el caso más simple y también donde ya no se hace un nuevo llamado al método recursivo. El caso recursivo representa la solución al resto del problema y se define en términos de su solución previamente ya definida.

Ejemplo 4.25. Encontrar el punto de retorno y caso recursivo de cada uno de los siguientes incisos:

a) Sumatoria: $1 + 2 + 3 + \ldots\ldots + n$

b) Sumatoria: $-1 + 2 - 3 + 4 - 5 + \ldots\ldots + (-1)^n n$

c) Factorial: $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n, \quad 0! = 1$

d) Sumatoria: $1 + 4 + 9 + \ldots + n^2$

Respuestas:

a)

$1 + 2 + 3 + \ldots + n$

Punto de retorno:
1 para $n=1$

Punto recursivo:
$suma(n-1) + n$ para $n>1$

b)

$-1 + 2 - 3 + \ldots + (-1)^n n$

Punto de retorno:
1 para $n=1$

Punto recursivo:
$suma(n-1) + (-1)^{n*} n$ para $n>1$

c)

$1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n$
$0!=1$

Punto de retorno:
1 para $n=0$, $n=1$

Punto recursivo:
$factorial(n-1)^* n$ para $n>1$

d)

$1 + 4 + 9 + \ldots + n^2$

Punto de retorno:
1 para $n=1$

Punto recursivo:
$Suma(n-1)+n^2$ para $n>1$

Analizando las soluciones a los incisos anteriores, es posible observar que el punto de retorno es la solución del problema para el primer o primeros valores de n, es la solución definida para valores determinados de n (por lo general son los valores más pequeños de n aunque no siempre es así, ya que depende del problema).

El punto recursivo es la solución del problema representado en forma abstracta, con la llamada al procedimiento en términos de n y tomando como base el punto de retorno, en donde están involucrados los últimos elementos.

Es muy importante que tenga un punto de retorno en donde ya no existe una llamada al método recursivo, de lo contrario el programa tendría llamadas infinitas y no terminaría nunca.

Los métodos recursivos tienen la desventaja de ocupar más memoria que los métodos iterativos, porque cuando el compilador llama el método recursivo guarda todas las variables locales hasta que se vacía completamente la pila, pero tienen la ventaja de que las soluciones son claras, elegantes y simples.

Se debe usar recursividad solamente en aquellos problemas complejos que poseen naturaleza recursiva y que por lo tanto son fáciles de implementar de esa manera.

Ejemplo 4.26. Escribir un programa para encontrar el resultado de n términos de la siguiente sumatoria:

$$2 + 4 + 6 + 8 + \ldots + 2n$$

Usar una función recursiva para tabular la iteración y la suma de los términos para esa iteración. Encontrar el punto de retorno y punto recursivo.

```
import java.util.Scanner;
public class recursiva {
public static void main(String[] args){
    Scanner leer = new Scanner(System.in);

int n;
System.out.print("Valor de n: ");
n=leer.nextInt();

System.out.println ("   n          Sumatoria");
System.out.println("Resultado= "+sumatoria(n));
}

  //Función recursiva
  static int sumatoria(int n){
      int c;
      if (n>1){
        c=sumatoria(n-1)+2*n;
        System.out.printf ("%4d   %12d\n",n,c);
        }
      else{
       c=2;
       System.out.printf ("%4d   %12d\n",n,c);
       }
  return c;
  }
 }
```

```
run:
Valor de n: 6
    n        Sumatoria
    1                2
    2                6
    3               12
    4               20
    5               30
    6               42
Resultado= 42
GENERACIÓN CORRECTA (total time: 3 seconds)
```

Observar cómo el punto de retorno es el resultado de la sumatoria para el primer valor de `n` y que el punto recursivo es la solución general, en donde están involucrados el n−ésimo término (`2*n`) y el resultado de la suma antes del último término a sumar `suma(n-1)`.

La función del programa anterior suma n términos de la sumatoria y regresa el resultado con la instrucción `return c;`, pero se agregaron líneas para imprimir cada uno de los valores parciales que salen de la pila, con la finalidad de obtener una tabla.

Recordar que las funciones recursivas van guardando valores no determinados en la pila, hasta obtener un valor conocido y en ese momento regresa sobre su propia trayectoria, sacando cada elemento de la pila y determinando su valor, hasta llegar a la primera expresión que entró a la pila, que permitirá calcular el valor buscado. En la siguiente tabla se muestran los valores no determinados para cada una de las iteraciones para los distintos valores de n.

n	c	
1	2	= sumatoria(1−1) +2*1
2	?	= sumatoria(2−1) +2*2
3	?	= sumatoria(3−1) +2*3
4	?	= sumatoria(4−1) +2*4
5	?	= sumatoria(5−1) +2*5
6	?	= sumatoria(6−1) +2*6

Cuando `n=6` no es posible determinar el valor de `c=sumatoria(n-1)+2*n;` por tal razón lo guarda en la pila. Lo mismo sucede para los valores de (`n>1`). El primer resultado que encuentra es cuando `n=1`, porque para ese caso sí está determinado el valor de `c=2`, por eso es que el primer valor que imprime es cuando la condicional es falsa. Después se retira la expresión que está en la cima de la pila y la evalúa, posteriormente retira la expresión siguiente, sustituye el valor de `c` en `sumatoria(n-1)` y le suma `2*n`, retira la siguiente expresión de la pila y la evalúa apoyándose del valor anterior, hasta que la pila quede completamente vacía. Observar que en los algoritmos recursivos no es necesaria una instrucción iterativa como (`while o for`) para llevar a cabo varias iteraciones, porque la computadora en forma natural guarda en la pila las expresiones que no es posible evaluar.

No solamente las funciones pueden ser recursivas, también los métodos pueden tener esta característica, si dentro de ese método se invoca al mismo método.

Ejemplo 4.27. Una cantidad en decimal se puede convertir a binario dividiendo dicha cantidad entre dos y quedándose con los residuos (o módulo) de la división. Ejemplo si la cantidad decimal es n=13, la conversión a binario es como sigue:

	Residuo	
13 / 2 = 6	1	Lo dígitos se toman de abajo hacia arriba lo cual implica que:
6 / 2 = 3	0	
3 / 2 = 1	1	
1 / 2 = 0	1	13 decimal es = 1101 en binario

Escribir un programa para leer un valor entero y que lo imprima en binario utilizando para ello un método recursivo.

```
import java.util.Scanner;
public class convierte_a_binario {
public static void main(String[] args){
    Scanner leer = new Scanner(System.in);

int n;
System.out.print("Valor de n: ");
n=leer.nextInt();

System.out.print ("Equivalente en binario: ");
binario(n);  //Ejecuta el procedimiento

//Salta de línea después de imprimir resultado
System.out.println ();
}

  //Método recursivo para realizar la conversión
  static void binario(int n){
      if (n>=2){
        binario(n/2);
        //Imprime los dígitos del módulo
      System.out.print (n%2);
      }
      else
       System.out.print (n);
   }
 }
```

```
Valor de n: 147
Equivalente en binario: 10010011
GENERACIÓN CORRECTA (total time: 3 seconds)
```

En el programa anterior se lee una cantidad en decimal, posteriormente se escribe el valor equivalente en binario utilizando el método recursivo `binario(n)`. El método se invoca varias veces él mismo dentro de su propio código y cada vez se divide a `n` entre dos (`n/2`) hasta que el cociente entero de dicha división es igual a uno (`n=1`). Esto implica que el primer resultado que se imprime es cuando deja de cumplirse la condición (`n>=2`), a partir de ese momento comienza a sacar las expresiones no determinadas de la pila, las evalúa e imprime el resultado hasta que la pila queda completamente vacía.

4.7 Resumen

La mayoría de problemas de computación requieren de estructuras selectivas que permiten seleccionar las operaciones a realizar si se cumple una condición o grupo de condiciones. Las instrucciones que ayudan a elegir el grupo de sentencias a ejecutar son las estructuras selectivas, entre las cuales se pueden mencionar a la *estructura condicional simple* (**`if`**), que permite ejecutar un bloque de instrucciones si se cumple una condición, la *estructura selectiva doble* (**`if-else`**) con la cual se puede elegir el bloque de instrucciones que se deben ejecutar si la proposición es verdadera o bien el conjunto de operaciones que se deben realizar si la condición es falsa, pero además se tiene la *estructura selectiva múltiple* (**`switch`**) que ayuda a darle claridad a los programas donde el número de condicionales es grande, ya que hace la misma operación de varias estructuras selectivas dobles anidadas.

Las estructuras iterativas en programación más comunes son *mientras* (**`while`**) que ejecuta un bloque de instrucciones encerradas entre llaves, mientras la proposición establecida sea verdadera, la estructura iterativa *hacer-mientras* (**`do-while`**) que se utiliza para ejecutar varias veces el bloque de sentencias que están entre las palabras **`do`** y **`while`** mientras sea verdadera la condición planteada después de la palabra **`while`**, y la estructura repetitiva *desde* (**for**) que ejecuta un bloque de sentencias controlada por una variable que se inicializa con un cierto valor, que tiene la condición y actualización dentro de la propia instrucción **`for`**.

Existen muchos programas que requieren de una estructura iterativa dentro de otra, lo que comúnmente se conoce como *ciclos anidados*, mismos que permiten potenciar la programación al resolver problemas complejos usando pocas líneas de código.

En las estructuras selectivas e iterativas se deben encerrar entre llaves el bloque de sentencias {`instrucciones..`} que se desean ejecutar, ya que esto permite extender o limitar el alcance. Un buen manejo de las llaves permite agrupar las sentencias que se realizan bajo ciertas condiciones.

Cuando se requiera programar algo, es conveniente primeramente revisar si el lenguaje cuenta con algún método que realiza esa actividad. Java tiene varios métodos que ayudarán a obtener mejores soluciones a los problemas que se presenten, sin necesidad de programar nuevamente rutinas, que realizan lo mismo que hace un método del propio lenguaje. Un ejemplo de ello son los métodos de la clase **`String`** que permiten realizar varias operaciones entre cadenas y las funciones de la librería `Math` que ayudan a encontrar la mayoría de los parámetros matemáticos y estadísticos conocidos.

Cuando un programa es muy grande es recomendable dividirlo en subprogramas o programas más pequeños con la finalidad de simplificar su solución. A estos pequeños programas se les llama en algunos lenguajes funciones, procedimientos o subrutinas, dependiendo del lenguaje y las características propias de los subprogramas. En Java se les llama métodos, aunque para respetar la definición de función, si un método recibe uno o varios parámetros y regresa un solo valor, recibe el nombre de función, pero si no cumple estas características, se le llama simplemente método.

Los métodos y las funciones se ubican generalmente después del programa principal, aunque podrían colocarse antes de dicho programa principal e incluso en una clase distinta. Se pueden invocar desde cualquier parte del programa, función e incluso desde el propio método.

Los métodos recursivos se caracterizan por invocarse a si mismos, tienen un punto de retorno en el cual la solución al problema es conocida y un punto recursivo donde la solución del problema se plantea en forma abstracta involucrando las soluciones anteriores. La recursividad es un método poderoso aplicado en la computación, su poder radica en que la solución de los problemas se puede expresar en forma simple. Tiene aplicaciones en la prueba de teoremas, juegos, acertijos, combinaciones, estructuras de datos e inteligencia artificial entre otras ramas de la computación. Se recomienda usar recursividad solamente

en aquellos problemas que por su naturaleza, tengan una solución recursiva ya que por lo general requieren de mayor memoria que los programas iterativos al guardar la información de soluciones parciales no conocidas.

4.8 Problemas

1. Escriba un programa para leer los tres lados de un triángulo (a, b, c). Suponiendo que c es el lado mayor determinar qué tipo de triángulo es de acuerdo a lo siguiente:

   - Si $c \geq a + b$. No se trata de triángulo.
   - Si $c^2 = a^2 + b^2$. Es un triángulo rectángulo.
   - Si $c^2 > a^2 + b^2$. Se forma un triángulo obtusángulo.
   - Si $c^2 < a^2 + b^2$. Se forma un triángulo acutángulo.

```
a: 4
b: 3
c: 6
Es un triángulo obtusángulo
```

2. Suponer m=7 y n=13. Colocar en el paréntesis de cada uno de los incisos una "V" si la proposición es verdadera o "F" si es falsa

   a) 2*m < = n ( )
   b) 3m− n < n ( )
   c) 2m−n > n−m ( )
   d) m>0 && (n<34 && !(m<47)) ( )
   e) ((n % m) + 1) >= m ( )

3. Un negocio de fotocopiado tiene la siguiente lista de precios.

Cantidad	Precio por copia
100 o menos	0.50
101 a 200	0.35
201 a 500	0.25
Más de 500	0.20

Si una persona ordena 720 copias; las primeras 100 se le cobran a 0.50, las segundas 100 a 0.35, las siguientes 300 a 0.25 y las restantes a 0.20. Escribir un programa que lea el número de copias y que calcule el costo total

Número de copias: 823
Importe = $224.60

4. Una agencia de alquiler de automóviles tiene la siguiente tarifa:

Recorrido (Kms)	Tarifa
300 o menos	1200.00 (Fija)
301 a 1000	4.00 c/u
Más de 1000	2.00 c/u

Escribir un programa para leer el kilometraje inicial y final de un auto y calcular el pago de la renta.

K.Inicial: 438
K.Final: 1672
Costo=$4468.00

$$Costo = 1200 + 700 * 4 + (1672 - 438) - 1000) * 2$$

5. En un comercio se realizan descuentos en función del monto total de la compra, como se muestra a continuación:

Importe	Descuento
500 o menos	0%
Entre 501 y 1000	5%
Entre 1001 y 2000	8%
Más de 2000	10%

Los descuentos solamente se hacen a las cantidades que se encuentran dentro de ese rango; por ejemplo, si una persona realiza una compra de $2345.00 el importe a pagar será el siguiente:

$$\text{Pago} = 500+500*95\%+1000*92\%+345*90\% = 2205.50$$

Escribir un programa para leer el monto de compra y calcular el descuento y cantidad a pagar.

Monto de compra: 3498
Descuento = $254.80
Pago = $3243.20

6. Expresar las líneas de código de cada uno de los incisos siguientes, usando solamente condiciones simples:

```
a) if ((a<b && c!=d) && (b>d || b==q))
      System.out.println ("Hola Pera");
   else
      System.out.println ("Adiós manzana");

b) if ((a*d>=c || c!=d) && (b-c>a || d%c==a))
      System.out.println ("Hola pera");
   else
      System.out.println ("Adiós manzana");
```

7. Si `a=1, b=2, c=3, d=4`. ¿Cuál es el resultado de la salida en cada uno de los incisos del problema anterior.

8. Los obreros de una fábrica de jabones pueden laborar en tres turnos: matutino, vespertino y nocturno, además podrían trabajar horas extras en caso de ser necesario. El pago por hora de cada una de las opciones se muestra en la siguiente tabla:

Turno	Pago por hora
Matutino	$20.00
Vespertino	$28.00
Nocturno	$35.00
Extra (Cualquier turno)	$50.00

Los trabajadores laboran 40 o más horas a la semana, las que excedan a esa cantidad se consideran horas extras. Se hará un descuento de 18%, por concepto de impuesto sobre la renta (ISR).

Escribir un programa para leer el turno y número de horas laboradas por el trabajador y calcular el salario semanal. Considerar que el código de los turnos es: 1 = matutino, 2= vespertino, 3 = nocturno.

```
Turno: 2
Horas: 47
Salario= $1470.00
Descuento=$264.60
Salario neto esta semana: $1205.40
```

9. El código de los meses del año es: 1: Enero, 2: Febrero,…,12: Diciembre. Escribir un programa para leer el año y código del mes. Con esa información desplegar en la pantalla el número de días que tiene dicho mes. El programa deberá seguir leyendo información hasta que el código del mes sea 999. Considerar también los años bisiestos; se sabe que un año es bisiesto si es múltiplo de 4, es decir si el resto de dividir el año entre 4 es cero.

```
Mes: 6
Año: 1987
Junio tiene 30 días
Mes: 2
Año: 2008
Febrero tiene 29 días
Mes: 999
```

10. Escribir un programa para elaborar una tabla de conversiones entre las unidades metros(m), kilómetros(km) y millas(mi) tomando en cuenta las siguientes equivalencias:

$$1 \text{ km} = 1000 \text{ m} \qquad 1\text{mi} = 1609 \text{ m}$$

Se leerá el inicio, final e incremento en metros de la tabla de conversión.

```
Inicio: 0
Final: 1000
Incremento: 260
 Metros     Kilómetros     Millas
      0        0.0000      0.0000
    260        0.2600      0.1616
    520        0.5200      0.3232
    780        0.7800      0.4848
   1040        1.0400      0.6464
```

11. Escribir un programa para imprimir todos los pares de números que suman una cantidad determinada c, en un intervalo dado m y n.

```
De: −5
A: 8
Suma: 13
Pares cuya suma es 13
5  8
6  7
7  6
8  5
Total de pares = 4
```

12. Escribir un programa para realizar una división de dos cantidades enteras, exclusivamente con restas. Las cantidades pueden ser del mismo signo o de signo contrario. Obtener el cociente y el residuo.

```
Dividendo: 25
Divisor: 7
Cociente = 3    Residuo = 4
```

```
Dividendo: -14
Divisor: 3
Cociente = −4  Residuo = −2
```

```
Dividendo: −17
Divisor: −6
Cociente = 2  Residuo = −5
```

13. Escribir un programa para imprimir las tablas de multiplicar del 1 al 10 de la siguiente manera:

Tabla del 1	**Tabla del 2**	**Tabla del 3**	**Tabla del 4**	**Tabla del 5**
$1 \times 1 = 1$	$2 \times 1 = 2$	$3 \times 1 = 3$	$4 \times 1 = 4$	$5 \times 1 = 5$
$1 \times 2 = 2$	$2 \times 2 = 4$	$3 \times 2 = 6$	$4 \times 2 = 8$	$5 \times 2 = 10$
.....	.....	.....	.....	.....
$1 \times 10 = 10$	$2 \times 10 = 20$	$3 \times 10 = 30$	$4 \times 10 = 40$	$5 \times 10 = 50$

Tabla del 6	**Tabla del 7**	**Tabla del 8**	**Tabla del 9**	**Tabla del 10**
$6 \times 1 = 6$	$7 \times 1 = 7$	$8 \times 1 = 8$	$9 \times 1 = 9$	$10 \times 1 = 10$
$6 \times 2 = 12$	$7 \times 2 = 14$	$8 \times 2 = 16$	$9 \times 2 = 18$	$10 \times 2 = 20$
.....	.....	.....	.....	.....
$6 \times 10 = 60$	$7 \times 10 = 70$	$8 \times 10 = 80$	$9 \times 10 = 90$	$10 \times 10 = 100$

a) Hacer el programa primeramente sin usar métodos.
b) Hacerlo usando un método para desplegar las tablas.

14. Escribir un programa para imprimir los números enteros positivos entre 1 y n, en orden sucesivo por renglón, según lo siguiente:

a) En el primer renglón va solamente el 1
b) A partir del segundo renglón se escribe un número más que en el renglón anterior.

```
Dn: 18

1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
16 17 18
```

15. La amistad es difícil de encontrar. Se dice que dos números A y B son amigos, si la suma de los divisores de A (sin considerar al número A) es igual B, y si la suma de los divisores de B es igual a A. Ejemplo.

Los divisores de A=28 son: suma1=1+2+4+7+14=28 y los divisores de B=28 son suma2=1+2+4+7+14=28. Por lo tanto A y B son amigos.

Escribir un programa para encontrar todos los números amigos comprendidos entre 1 y n. Escribir el programa:

a) Sin utilizar métodos.
b) Utilizando una función para sumar los divisores de los números.

```
Valor de n: 1000
Números amigos:.
    6      6
   28     28
  220    284
  ....    ...
```

16. Se dice que un número es "caprichoso" si la suma de sus dígitos elevada a cierta potencia n es igual al mismo número. Los números siguientes son caprichosos si la suma de sus dígitos se eleva a la potencia 3.

$4913 = (4+9+1+3)^3 = 4913$

$5832 = (5+8+3+2)^3 = 5832$

Escribir un programa para imprimir y contar los números caprichosos entre a y b, para una potencia de n.

```
a : 1
b: 100
n: 2
Lista de números caprichosos:
1
81
Total  = 2
```

17. Se dice que un número x es "raro" si se cumple que: 3m + s = x. Donde m resulta de escribir el número x en orden inverso, y s es la suma de las cifras de x. Ejemplo:

Para x=51; m=15; s=6;

Se cumple que: 3(15) + 6 = 51

Por lo tanto 51 es un número raro. Escribir un programa para listar todos los números raros entre 1 y n.

```
n: 6000
Números raros entre 1 y 6000
51
441
882
```

18. Suponer que el precio de todos los productos en una tienda departamental es una cantidad entera. Escribir un programa que lea de teclado el precio de cada uno de los productos que un cliente compra en dicha tienda, deja de leer precios de productos cuando el precio sea igual a −1. Después de leer los precios de los productos se debe leer también la cantidad entera con que paga el cliente. Finalmente deberá imprimir el cambio en billetes (o monedas) de 500, 200, 100, 50, 20, 10, 5, 2 y 1 pesos. Se requiere que la computadora indique el cambio con los billetes o monedas de mayor denominación.

```
Precio productos
15
36
107
13
−1
Total a pagar = $171
Cantidad recibida $: 500
Cambio = $329
Con      billetes o monedas de
  1              200
  1              100
  1               20
  1                5
  2                2
```

19. El *mínimo común múltiplo* (*mcm*) de dos números ***a*** y ***b*** es el menor número natural que es múltiplo de ellos. El *mcm*(*a, b*) se puede encontrar al multiplicar los factores primos comunes y no comunes elevados a su mayor potencia.

El *máximo común divisor* (*mcd*) de dos números ***a*** y ***b*** es el mayor número que los divide. El *mcd*(*a, b*), se puede encontrar al multiplicar los factores comunes con su menor exponente.

Para encontrar el *mcm*(50, 12) y el *mcd*(50, 12) se descomponen en sus factores primos cada uno de ellos:

	factores		factores
50	2	12	2
25	5	6	2
5	5	3	3
1		1	

De tal manera que $a = 50 = 2.5^2$ y $b = 12 = 2^2.3$.

El $mcm(50, 12) = 5^2.2^2.3 = 300$. Ya que se tomaron los factores no comunes 5^2 y 3, además del factor común de mayor exponente 2^2.

El $mcd(50, 12) = 2$, que es el único factor que es común a ambos números con el menor exponente.

Escribir un programa para leer dos números enteros positivos ***a*** y ***b***. Descomponer dichos números en sus factores. Encontrar el *mcm*(*a, b*) y el *mcd*(*a, b*), usando el método anteriormente descrito.

```
a : 450
b: 840
Factores
450 = 2*3*3*5*5
840 = 2*2*2*3*5*7
mcm(450, 840)=2*2*2*3*3*5*5*7 = 12600
mcd(450,840)= 2*3*5 = 30
```

20. Existe relación entre el *mcm* y el *mcd* de los números ***a*** y ***b***, ya que conociendo el *mcd(a,b)* es posible conocer el *mcm(a,b)*, aplicando la siguiente fórmula.

$$mcm(a,b) = \frac{a * b}{mcd(a,b)}$$

El matemático Griego Euclides, diseñó un algoritmo para encontrar el *mcd(a, b)*. El algoritmo consiste en llevar a cabo divisiones Euclidianas sucesiva. En la primera división, se toma como dividendo el mayor de los números y como divisor el otro. Luego el divisor y el resto se utilizan respectivamente como dividendo y divisor en la siguiente división. Se deja de iterar hasta que se obtiene como resultado un resto de cero y el *mcd(a, b)* es el penúltimo resto obtenido.

Ejemplo. Para encontrar el *mcd*(840, 450) y el *mcm*(840, 450) usando el algoritmo de Euclides se procede de la siguiente manera.

Iteración	Dividendo	Divisor	Resto
1	840	450	390
2	450	390	60
3	390	60	30
4	60	30	0

Por lo tanto *mcd*(840,450) = 30, ya que fue el penúltimo resto encontrado. El *mcm*(840,450) es:

$$mcm(840,450) = \frac{840 * 450}{mcd(840,450)} = \frac{378000}{30} = 12600$$

Escribir un programa para leer dos números enteros positivos ***a*** y ***b***. Encontrar el *mcm*(*a, b*) y el *mcd*(*a, b*), usando el algoritmo de Euclides.

```
a : 4200
b: 234
Iteración   dividendo   divisor   Resto
        1        4200       234     222
        2         234       222      12
        3         222        12       6
        4          12         6       0
mcd(4200,234) = 6
mcm(4200,234) = 163800
```

21. Se considera que una frase palíndroma es aquella que se escribe igual al derecho y al revés ignorando los espacios. Ejemplo de una frase palíndroma es "**hola nena an enaloh**". Escriba un programa con un método que acepte una frase de teclado y que determine si se trata de una frase palíndroma o no. Leer varias frases hasta que la frase sea "finaliza".

```
Frase: esto es in teresante
etnaseret ni se etse => No es frase palíndroma
Frase: el arbo les selo brale
elarb oles sel obra le => Es frase palíndroma
Frase: finaliza
```

22. Escribir un programa con un método que acepte una frase como parámetro y que regrese al punto donde fue llamado el método, para imprimir el número de palabras de la frase y el promedio de letras de las palabras. Considerar que la frase puede tener palabras separadas por uno o más espacios.

```
Frase: el león no es como lo pintan
La frase tiene 7 palabras
Promedio de letras de las palabras = 3
```

23. Escribir un programa para leer tres números (a, b y c) de teclado y encontrar el promedio aritmético de ellos por medio de una función:

```
A: 9
B: 5
C: 8
Promedio = 7.33
```

24. Escribir un programa para leer el número de puntos a tabular "n", los valores de las variables "x" y "*y*", con su correspondiente incremento. Con esta información desplegar en una tabla los valores de las variables y el valor de la función, usando la siguiente expresión matemática:

$$f(x, y) = \frac{x^2 - y^2}{x^2 + y^2}$$

```
n: 4
x: 2
Incremento de x: 0.76
y: 19
Incremento de y: −4.38
   x        y        f(x,y)
 2.00    19.00     −0.9781
 2.76    14.62     −0.9312
 3.52    10.24     −0.7886
 4.28     5.86     −0.3043
```

25. Escribir un programa para contar los puntos con coordenadas enteras que se encuentran en la siguiente elipse.

$$\frac{x^2}{16}+\frac{y^2}{25}=1$$

- Considerar que los puntos sobre la elipse están dentro de ella.
- El intervalo de coordenadas está limitado por los ejes mayor y menor de la elipse. En este caso $-4 \leq x \leq 4$ y $-5 \leq y \leq 5$
- Imprimir las parejas de puntos dentro de la elipse cuya suma sea la indicada por teclado.

```
Imprimir puntos donde (x+y) sea: 8
Puntos dentro de la elipse cuya suma es x+y=8
(3,5)
(4,4)
Total dentro de la elipse = 95
```

26. Escribir un programa para tabular n puntos de la parábola $y=3x^2-4x+5$. Leer los valores de n, x y el incremento de x. El programa deberá funcionar:

a) Sin utilizar funciones.
a) Utilizando una función para encontrar el valor del polinomio.

```
In: 6
x: 3.23
Incremento de x: .75
 Punto        x        y=3x^2-4x+5
   1        3.23          23.3787
   2        3.98          36.6012
   3        4.73          53.1987
   4        5.48          73.1712
   5        6.23          96.5187
   6        6.98         123.2412
```

27. Escribir un programa para calcular el valor real de cos(x). Leer de teclado el valor de x y el error permitido. Aproximar cos(x) al valor real usando la serie que se muestra a continuación. El programa terminará cuando el error entre la función real y la aproximada sea menor o igual al error permitido. Tabular los resultados de cada iteración hasta lograr la exactitud indicada.

$$\cos(x)=1+\sum_{n=1}^{\infty}(-1)^n\frac{x^{2n}}{(2n)!}=1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\ldots..$$

Usar una función para calcular el factorial y otra para calcular e imprimir cada uno de los términos que se van sumando. Leer el ángulo en grados, pero considerar que los ángulos de las funciones trigonométricas se deben expresar en radianes en la mayoría de los lenguajes de programación. (π rad=180º)

```
x: 60
Error: 0.001
cos(x) real = 0.500000
 n        cos(x) Aprox.        Diferencia
 0                   1          0.500002
 1          0.45168608          0.048312
 2        0.501794106          0.001796
 3        0.499962444          0.000035
```

28. Un número es palíndromo si es el mismo al derecho y al revés. Escribir un programa para leer un número entero y determinar si el número leído es palíndromo. Hacerlo de las siguientes maneras diferentes:

a) Apoyándose en las funciones cadena de Java.
b) Sin utilizar las cadenas de Java.

```
Número: 1704071
Con apoyo de funciones cadena de Java.
1704071 es palíndromo
Tratándolo como número
Es palíndromo 1704071
```

29. Escribir un programa para un juego en donde se realiza lo siguiente:

a) Se lee el nombre del jugador.
b) Se genera un número al azar mayor que 13
c) El jugador resta una cantidad entre 1 y 3. Posteriormente la computadora también restará una cantidad entre 1 y 3.

El juego terminará cuando el número se reduce a 0. Ganará la computadora (o el usuario) si se apoderan del 0. Hay números clave que se deben de tomar, asegurarse que la computadora gane, si se llega a apoderar antes que el usuario de alguno de esos números clave.

```
run:
Nombre:  Juan
Número generado: 15
Juan resta: 3
Número= 12
Compu. resta: 2
Número= 10
Juan resta: 3
Número= 7
Compu. resta: 3
Número= 4
Juan resta: 1
Número= 3
Compu. resta: 3
Número= 0
GANÓ la computadora
GENERACIÓN CORRECTA (total time: 26 seconds)
```

30. El juego de dados es muy común en ferias y casinos. En dicho juego hay una banca representada por la persona que atiende, el jugador, una mesa con el número 7 al centro de dicha mesa, los números menores que 7 (2, 3, 4, 5, 6) a la izquierda de la mesa y los mayores (8, 9, 10, 11, 12) a la derecha. El jugador coloca la cantidad apostada en el número 7, después se lanzan por primera vez los dados. Si el resultado de este primer lanzamiento sumando los dos dados es:

2, 3 ó 12	La banca gana
11 ó 7	El jugador gana

Si no es ninguno de los resultados anteriores, la cantidad apostada por el jugador se pasa al número que salió. Supóngase que salió el 8 entonces la cantidad se pasa al número 8. El jugador deberá seguir lanzando los dados hasta obtener el número de la primera tirada (en este caso el 8) o bien obtener el número 7. Ganará:

a) El jugador; si obtiene nuevamente el número que lanzó en la primera tirada.
b) La banca; si el número que sale es el 7.

Después de la primera tirada pueden salir el 2, 3, 12 ó 11 varias veces y nadie gana, ya que se deberán seguir haciendo lanzamientos hasta obtener el número lanzado en la primera tirada o el 7.

Escribir un programa para simular el juego de dados. Considerar que la banca es la computadora y el usuario es el jugador. Primeramente preguntará el monto de la apuesta, deberán aparecer los resultados de los lanzamientos en la pantalla, indicar quien ganó y preguntar si se desea seguir jugando. Al final informar las pérdidas o ganancias del jugador.

Los números que se generan aleatoriamente en un lenguaje de programación tienen una distribución uniforme, esto significa que tienen la misma probabilidad de ser elegidos. Tomando en cuenta esto; para la simulación del juego de dados se deben hacer los ajustes correspondientes, teniendo en cuenta que la probabilidad de los diferentes resultados de los números que se obtienen a lanzar los dados (2=1/36, 3=2/36, 4=3/36, 5=4/36, 6=5/36, 7=6/36, 8=5/36, ...12=1/36).

```
Monto de la apuesta: 1000
Primer lanzamiento.
2    5
Gana jugador
¿Seguir jugando (s/n)?: s
Monto de la apuesta: 1800
Primer lanzamiento.
4    2
Lanzamientos siguientes
6    4
3    5
6    6
3    4
Gana la banca
¿Seguir jugando (s/n)?: n
El jugador perdió = 800
```

31. Escribir un programa que lea el valor de n y encuentre el resultado en forma recursiva de cada una de las series de los siguientes incisos:

a) Sumatoria: $1 + 2 + 3 + \ldots\ldots + n$
b) Sumatoria: $-1 + 2 - 3 + 4 - 5 + \ldots\ldots + (-1)^n n$
c) Factorial: $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n$
d) Sumatoria: $1 + 4 + 9 + \ldots + n^2$

32. Escribir un programa para encontrar el n−esimo término de la serie de Fibonacci en forma recursiva.

```
Valor de n: 6
El elemento 6 de Fibonacci es = 8
```

ARREGLOS

5

Controlar la complejidad es la esencia de la programación.

Brian Kernigan

Competencia de la unidad

Construir programas utilizando arreglos de diferentes dimensiones para crear soluciones más apegadas a la realidad.

- Entender y utilizar los conceptos básicos para el manejo de arreglos.
- Diseñar y escribir programas incluyendo arreglos de una dimensión.
- Diseñar y escribir programas incluyendo arreglos de más de una dimensión.

Control de flujo

Contenido

5.1. Introducción.
5.2. Arreglos unidimensionales: conceptos básicos, operaciones y aplicaciones.
5.3. Arreglos bidimensionales: conceptos básicos, operaciones y aplicaciones.
5.3.1. Arreglos irregulares.
5.3.2. Arreglos como argumentos y parámetros.
5.4. Arreglos multidimensionales: conceptos básicos, operaciones y aplicaciones.
5.5. Resumen.
5.6. Problemas.

5.1 Introducción

Una variable es capaz de almacenar un valor a la vez (y solo uno). Cuando existe la necesidad de almacenar más valores entonces se deben crear más variables, una por cada valor que se quiere almacenar, considerando que los nombres de las variables deben ser diferentes.

Si se desea almacenar el número de calorías consumidas por una persona en una semana determinada entonces deben crearse siete variables diferentes. Si es necesario llevar un registro del número de quejas recibidas por una compañía telefónica en el mes de diciembre, entonces se hace necesaria la creación de 31 variables, una por cada día del mes. En las dos situaciones anteriores resultaría impráctico declarar tantas variables diferentes para almacenar cada valor, dado que en cada situación los valores tienen el mismo significado, solo que para días diferentes. Otro inconveniente sería cuidar que los identificadores de las variables no se repitieran. Para estos casos la utilización de arreglos es muy útil.

Un arreglo es un conjunto de datos o elementos de un mismo tipo y con un mismo identificador. De esta manera se puede tener un arreglo identificado como *calorías* y almacenar en él los siete valores del número de calorías consumidas en cada día de una semana; de la misma manera podría crearse un arreglo de 31 elementos para almacenar el número de quejas de cada día del mes de diciembre.

Aunque todos los elementos de un mismo arreglo se identifican con el mismo nombre es posible diferenciarlos a través del lugar o posición que ocupan. Esta diferenciación se hace a través de un índice. Así pues, si tenemos el arreglo `quejas`, entonces `quejas[0]` estaría almacenando el número de quejas que se dieron en el primer día de diciembre, `quejas[1]` contendría el número de quejas en el segundo día de diciembre, y así sucesivamente. En la mayoría de los lenguajes de programación, como Java, los arreglos se rigen por una indexación basada en cero, esto quiere decir que los elementos de un arreglo comienzan a numerarse a partir de cero, así que el primer elemento del arreglo es el elemento cero, el segundo elemento es el elemento uno, el tercer elemento el dos, etc.

El arreglo `calorías`, considerando que una persona ha quemado 2040 calorías el lunes, 2500 el martes, 1890 el miércoles, 2000 el jueves, 2200 el viernes, 2560 el sábado y 2800 el domingo podría representarse así:

`calorías`

valor almacenado	2040	2500	1890	2000	2200	2560	2800
índice	[0]	[1]	[2]	[3]	[4]	[5]	[6]

Es importante notar que el arreglo es de 7 elementos; el primer elemento tiene un índice 0 y el último tiene un índice 6. Entonces un arreglo de `n` elementos utilizará índices desde cero hasta `n-1`.

Los *vectores* son arreglos unidimensionales. Los vectores necesitan de un solo índice para diferenciar a cada elemento. Los arreglos bidimensionales reciben el nombre de *matrices*. Las matrices hacen uso de dos índices; uno para indicar el número de renglones y otro para indicar el número de columnas. Aunque menos comunes, también existen los arreglos *multidimensionales* de más de dos dimensiones, que consideran un índice por cada una de sus dimensiones.

5.2 Arreglos unidimensionales: conceptos básicos, operaciones y aplicaciones

Los arreglos unidimensionales, tienen un solo índice para diferenciar sus elementos. Para utilizar un arreglo, es necesario declararlo y crearlo previamente.

Declaración y creación de un arreglo unidimensional. Para crear un arreglo en Java, es necesario declararlo primero. De manera general un arreglo unidimensional en Java se declara de la siguiente manera:

```
tipo identificador [];
```

La línea anterior solo declara el arreglo, pero no lo crea, pues Java concibe a los arreglos como referencias, así que para crear o instanciar el arreglo es necesario el operador **new**. La sintaxis para crear un arreglo es:

```
identificador =new tipo[num_de_elementos];
```

Las líneas de declaración y creación del arreglo pueden fusionarse en una sola quedando como se muestra a continuación:

```
tipo identificador[]=new tipo [num_de_elementos];
```

A continuación se muestran ejemplos de creación de arreglos:

```
//crea un arreglo de 7 elementos enteros
int calorías[]=new int[7];

//crea un arreglo de 31 elementos enteros
int quejas []=new int [31];

//crea un arreglo de 5 elementos flotantes
float medidas[]=new float [5];

//crea un arreglo de 10 elementos caracter
char grupos[]=new char[10];
```

Almacenar valores en el arreglo. Para almacenar valores en las diferentes celdas del arreglo basta con hacer uso del nombre del arreglo y la ubicación deseada, por ejemplo:

```
//se almacena el valor 2040 en la celda cero
calorias[0]=2040;

// se almacena el valor 2500 en la celda uno
calorias[1]=2500;

// se almacena el valor 2800 en la posición 38 si i=38
calorías [i]=2800;
```

También se puede almacenar un valor leído desde el teclado:

```
/* se almacena el valor leído desde el teclado en la celda cero
del arreglo calorías */
calorias[0]=leer.nextInt();

/*se almacena el valor leído desde el teclado en la celda dos del
arreglo medidas */
medidas[2]=leer.nextFloat();

//si i=4, guarda el valor leído de teclado en la celda 17
totales [5*i-3]=leer.nextInt();
```

Se puede observar que es posible hacer operaciones con los subíndices de un arreglo, cuidando solamente que el resultado de la operación sea un subíndice valido de los números naturales {0, 1, 2, 3, ….}. El subíndice más grande de un arreglo depende de la memoria de la computadora, lo cual significa que no se pueden almacenar datos en forma infinita en un arreglo, aunque puede llegar a ser considerable la cantidad de información que se tiene en un arreglo, dado que cada vez las computadoras tienen mayor memoria principal.

Lo más práctico para almacenar valores dentro de un arreglo es apoyarse en un ciclo que controle adecuadamente el índice de cada celda dentro del arreglo:

```
for (indice=0;indice<31;indice++){
  System.out.println("¿Cuántas quejas hubo en el día?: "+
(indice+1));
  quejas[indice]=leer.nextInt();
}
```

Así la variable índice comenzará en el valor de cero y consecutivamente permitirá meter valores en todas las celdas del arreglo, hasta llegar al treinta.

Para aplicar cualquier acción sobre un elemento en particular de un arreglo, bastará con hacer uso del nombre del arreglo y su índice.

Atributo `length`. En Java todos los arreglos tienen un atributo que conserva el valor del número de elementos que almacena. De esta manera, es posible utilizar el atributo `length` para controlar el índice de un arreglo.

```
for (indice=0;indice<quejas.length; indice++){
      System.out.println("Cuántas quejas hubo en el día "+
(indice+1));
      quejas[indice]=leer.nextInt();
}
```

En las líneas anteriores `quejas.length` representa la cantidad de lugares *ocupados o no* del arreglo `quejas`, porque no es necesario que se guarde información en todos los lugares del arreglo, ni tam-

poco la ocupación de los lugares es en forma secuencial, ya que puede guardarse un elemento en cualquier celda del arreglo en el momento en que se desee.

Inicializar un arreglo. Al igual que las variables convencionales, es posible inicializar un arreglo desde el momento en que se crea. Solo es necesario encerrar entre llaves los valores de inicialización y separar cada valor con coma. En este caso puede omitirse la dimensión, pues en la inicialización está de manera implícita.

```
int calorias[] = {2040, 2500, 1890, 2000, 2200, 2560, 2800};
float medidas[] = {3.15, 2.1, 3.5, 5.3, 9.1};
char grupos[] ={'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'};
```

Ejemplo 5.1. Escribir un programa para leer de teclado el número de calorías quemadas en cada día de la semana. Imprimir nuevamente el conjunto de calorías, pero ahora en orden contrario a como fueron leídas e imprimir su promedio.

```
import java.util.Scanner;
public class Prom_Calorias {

static Scanner leer=new Scanner(System.in);
public static void main(String[] args) {

//se crea un arreglo de 7 elementos enteros
int calorias[]=new int[7];
int indice,suma=0;
System.out.println("Captura las calorías consumidas cada día");

for(indice=0;indice<calorias.length;indice++){
   System.out.print("Día "+(indice+1)+" : ");

   //Se lee cada elemento del arreglo
   calorias[indice]=leer.nextInt();
   suma+=calorias[indice];
   }

//Imprime las calorías leídas en orden inverso
System.out.println("Información leída en orden inverso");
for(indice=calorias.length-1;indice>=0;indice--)
   System.out.print(calorias[indice]+" ");

System.out.println();  //Salta a la siguiente línea

System.out.println("El promedio de calorías quemadas es =" +
((float)suma/calorias.length));
}
}
```

```
run:
Captura las calorías consumidas cada día
Dia 1 : 2040
Dia 2 : 2500
Dia 3 : 1890
Dia 4 : 2000
Dia 5 : 2200
Dia 6 : 2560
Dia 7 : 2800
Información leída en orden inverso
2800 2560 2200 2000 1890 2500 2040
El promedio de calorías quemadas es = 2284.2856
GENERACIÓN CORRECTA (total time: 31 seconds)
```

En este ejemplo se piden los valores para cada una de las celdas del arreglo, para ello se emplea la variable índice que comienza con el valor de 0 y llega hasta un valor de 6 a través del atributo **`length`** del arreglo. En cada iteración o vuelta de ciclo se imprime el número de día para el cual debe introducirse vía teclado un valor entero. Es importante notar que en la instrucción **`System.out.print`**`("Día "+(índice + 1)+" : ")` el objetivo de usar `(indice + 1)` es visualizar a partir de 1 el día (para no mostrar dia 0), pero al momento de leer cada valor con la instrucción `calorías[índice]=leer.`**`nextInt`**`()` a la variable índice no se le suma uno, pues el primer elemento a guardarse en el arreglo quedará en la celda 0.

Después de leer cada uno de los valores del arreglo, éste se va acumulando en la variable `suma`. Al salir del ciclo todos los valores del arreglo se han acumulado en `suma`. Entonces se imprime el arreglo en orden inverso a como fue leído y se imprime el resultado de la variable `suma` dividido entre el número de elementos del arreglo, utilizando el atributo **`length`**. El motivo de utilizar ***`(float)`*** al momento de imprimir el promedio de calorías quemadas es mostrar con precisión decimal el resultado de la división `suma/calorias.`**`length`**, pues de no usarlo solo se mostraría un valor entero, debido a que los valores con los que se está realizando la división son enteros. A esta acción se le conoce como hacer *cast* o enmascarar un tipo de dato, en este caso se enmascara al resultado como flotante para que muestre decimales.

Ejemplo 5.2 Hacer un programa que tome uno a uno los elementos enteros de un arreglo llamado *bases* ya inicializado y en un segundo arreglo llamado *resultados* coloque el cuadrado de cada número del arreglo bases en caso de ser par, o el cubo si es impar.

```
//Clase donde se encuentra el método pow
import java.lang.Math;
public class Cuadrado_Cubo {
public static void main(String[] args) {
int bases[]={12,0,2,5,6,7,10};
int resultados[]=new int[bases.length];
int indice=0;
```

```
while(indice<bases.length){
   if(bases[indice]%2==0)  //en caso de ser par
      resultados[indice]=(int)Math.pow(bases[indice], 2);
   else   //si es impar
      resultados[indice]=(int)Math.pow(bases[indice], 3);
   indice++;
   } //llave final del while

System.out.println("El arreglo resultados queda con los valores ");
// se regresa indice a cero para comenzar en la celda 0
indice=0;
while(indice<resultados.length){
   System.out.println(resultados[indice]);
   indice++;
   } //llave final del while
 }
}
```

```
run:
El arreglo resultados queda con los valores
144
0
4
125
36
343
100
GENERACIÓN CORRECTA (total time: 2 seconds)
```

En el programa anterior se crean 2 arreglos: uno con el nombre `bases` y otro con el nombre `resultados`. El arreglo `bases` se inicializa directamente con los valores 12,0,2,5,6,7,10 y el arreglo `resultados` solo se crea del mismo tamaño que el primer arreglo, utilizando el atributo **`length`**. En el primer ciclo `while`, se checa con la condición `if(bases[indice]%2==0)` si cada elemento ubicado en la celda indicada por `indice` es par; de ser así el elemento es elevado al cuadrado y colocado en la misma ubicación de celda, pero en el arreglo `resultados`. De ser impar, el elemento es elevado al cubo y colocado en la misma ubicación de celda, pero en el arreglo `resultados`. Como la variable `indice` se incrementa en cada vuelta de ciclo, se van tomando uno a uno de manera consecutiva los elementos del arreglo. Al salir del primer ciclo `while`, la variable `indice` debe regresar a cero, pues se pretende mostrar los valores que quedaron en el arreglo `resultados` desde la ubicación cero hasta el final del arreglo. El programa utiliza la línea de importación `import java.lang.Math`, pues es ahí donde se encuentra la función `pow` que se encuentra en el paquete `Math` (De ahí que se haya escrito `Math.pow`) que lleva 2 argumentos, el primero es la base y el segundo el exponente. En el ejemplo la

base es cada elemento del arreglo `bases`. La función `pow` devuelve el resultado de elevar la base al exponente indicado. En el ejemplo cada celda del arreglo `resultados` recibe el resultado de elevar cada número del arreglo bases ya sea al cubo o al cuadrado. Se enmascara con (`int`) el resultado de cada potencia, debido a que originalmente la función `pow` devuelve un dato tipo `double` y haciendo el cast con (`int`) se obliga a que el resultado se interprete como entero para poderlo almacenar en arreglo `resultados`, que es de tipo entero.

Ejemplo 5.3 Escribir un programa que declare e inicialice un arreglo de caracteres de tal manera que éstos formen la palabra PROGRAMACION. El arreglo deberá de recorrer sus caracteres a la Derecha o a la Izquierda a petición del usuario de manera constante a través de un menú. Al recorrer los caracteres hacia la izquierda el caracter con índice cero pasará a la última celda del arreglo. Al recorrer el arreglo hacia la derecha el último caracter del arreglo pasará a la celda con índice cero. Cada que se recorran los caracteres debe mostrarse el arreglo para ver cómo quedó. Las opciones del menú serán: **1)** Recorrer a la Derecha **2)** Recorrer a la Izquierda **3)**Terminar.

```
import java.util.Scanner;
public class Recorrer_Caracteres {
static Scanner leer=new Scanner(System.in);
public static void main(String[] args) {
char carac[]={'P','R','O','G','R','A','M','A','C','I','O','N'},aux;
int i, opcion, longitud;
longitud=carac.length;
System.out.print("\nMENU\n1) Recorrer a la derecha\n2) Recorrer a
la izquierda\n3) Terminar\n\n");
do {
   System.out.print("La palabra es: ");
   for(i=0;i<longitud;i++)
   System.out.print(carac[i]+ " ");

   System.out.println();
   System.out.print("Qué deseas hacer: ");
   opcion=leer.nextInt();

   switch(opcion){
      case 1:  //se respalda el último caracter
               aux=carac[longitud-1];
              for(i=longitud-1;i>0;i--)
                 carac[i]=carac[i-1];

              /* en la celda cero se coloca el carácter
                respaldado */
              carac[0]=aux;
              break;
     case 2:  //se respalda el primer caracter
              aux=carac[0];
              for(i=0;i<longitud-1;i++)
```

```
                carac[i]=carac[i+1];
            /*en la última celda se coloca el carácter
              Respaldado */
             carac[longitud-1]=aux;

      }  //fin del switch
  }while(opcion!=3);
 } }
```

```
run:
MENU
1) Recorrer a la derecha
2) Recorrer a la izquierda
3) Terminar

La palabra es: P R O G R A M A C I O N
Qué deseas hacer: 2
La palabra es: R O G R A M A C I O N P
Qué deseas hacer: 1
La palabra es: P R O G R A M A C I O N
Qué deseas hacer: 1
La palabra es: N P R O G R A M A C I O
Qué deseas hacer: 3
GENERACIÓN CORRECTA (total time: 12 seconds)
```

En el ejemplo se crea un arreglo con el nombre `carac` que se inicializa con 12 elementos de tipo *caracter*. A continuación la variable `longitud` recibe el valor que corresponde al tamaño del arreglo `carac`, en este caso recibirá el valor de 12. Una vez impresos los caracteres del arreglo, se imprime el menú con las 3 opciones y se lee del teclado un valor para ser almacenado en la variable `opcion`. En seguida se abre ciclo `do-while` que comienza imprimiendo a través de un ciclo `for` cada uno de los caracteres del arreglo `carac`. Dependiendo del valor que tome la variable `opcion`, el control del programa se encaminará hacia el caso 1 o el caso 2. Si se trata del caso 1 entonces es importante respaldar el caracter que está en la última celda del arreglo con la instrucción `aux=carac[longitud-1]`. Esto debido a que con el ciclo definido por `for(i=longitud-1;i>0;i--)` cada caracter del arreglo pasará a la celda de la derecha con la instrucción `carac[i]=carac[i-1]` y si no se respalda el último caracter del arreglo, éste se perdería al sobre escribirse en él el carácter ubicado a su izquierda. Al finalizar el ciclo cada caracter se sobre escribió en la celda ubicada a su derecha, perdiéndose el caracter que anteriormente se encontraba ahí. Es entonces cuando con la instrucción `carac[0]=aux` se coloca el carácter respaldado en la celda 0 del arreglo, perdiendo el que anteriormente se encontraba ahí. Al entrar al caso 2 sucede exactamente lo mismo pero recorriendo los caracteres hacia la izquierda, de ahí que el caracter a respaldar sea el que está en la celda 0. El ciclo definido por `for(i=0;i<longitud-1;i++)` provoca que cada carácter sea sobre escrito con el caracter ubicado a su derecha, por lo tanto al finalizar el ciclo es necesario poner el caracter respaldado en la última celda del arreglo. Las opciones podrán seguirse eligiendo mientras no se elija la opción 3 que causa la terminación del programa.

A continuación se ejemplifica el recorrido de caracteres para el caso 1 (recorrer caracteres hacia la derecha) con el ciclo `for(i=longitud-1;i>0;i--)`:

opcion	aux	i	carac											
1	'N'	11	'P'	'R'	'O'	'G'	'R'	'A'	'M'	'A'	'C'	'I'	'O'	'N'
			[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]

La instrucción `carac[i]=carac[i-1]`, que para el valor actual de `i` sería `carac[11]=carac[10]` equivale a colocar el caracter ubicado en la celda 10, sobre el caracter ubicado en la celda 11, es decir el caracter 'O' se escribiría sobre el caracter 'N', perdiéndose éste último. En la siguiente iteración del ciclo `i` vale 10, entonces `carac[10]=carac[9]`, es decir el carácter 'I' se escribiría sobre el caracter 'O', perdiéndose éste último, de tal manera que `i` va cambiando durante el ciclo repitiendo este proceso hasta que `i` llegue a 0. Al concluir el ciclo los valores de las variables, incluyendo el arreglo quedarían así (la variable `i` fue decrementándose hasta llegar al valor de 0, que provoca la ruptura del ciclo):

opcion	aux	i	carac											
1	'N'	0	'P'	'P'	'R'	'O'	'G'	'R'	'A'	'M'	'A'	'C'	'T'	'O'
			[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]

Apreciar que el caracter 'P' quedó duplicado y el caracter 'N' que estaba en la celda cero se perdió (pero está respaldado en la variable `aux`). Es entonces cuando con la instrucción `carac[0]=aux` los valores de las variables, incluido el arreglo quedan así:

opcion	aux	i	carac											
1	'N'	0	'N'	'P'	'R'	'O'	'G'	'R'	'A'	'M'	'A'	'C'	'T'	'O'
			[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]

Este proceso se repetiría cada que se elija la opción 1 del menú. Al elegir la opción 2 sucedería algo similar, pero hacia el lado contrario.

Ejemplo 5.4. Escribir un programa que contenga un menú con las opciones: **1)** Llenar el arreglo **2)** Hacer Búsqueda binaria **3)** Terminar. La primera opción pedirá al usuario 4 números enteros para ser almacenados en el arreglo, cuidando que los números se introduzcan de manera ascendente en el arreglo, en otras palabras, el número que se acaba de introducir debe ser mayor que el anterior (la búsqueda binaria trabaja sobre arreglos ordenados). La opción 2) solicitará al usuario un número para ser buscado en el arreglo; si el número se encuentra en el arreglo el programa deberá mostrar en qué celda se encontró, de lo contrario mostrará el mensaje "el número que buscas no está en el arreglo". Para la búsqueda del elemento en el arreglo se aplicará la búsqueda binaria. El menú debe aparecer de manera continua hasta que el usuario elija la opción 3. Para las opciones 1 y 2 deberán escribirse 2 funciones.

```
import java.util.Scanner;

public class Busq_Bin {
```

```
static Scanner leer=new Scanner(System.in);
public static void main(String[] args) {
int arreglo[]=new int [4];
// bandera con cero, considerando que el arreglo no se ha llenado
int opcion=0, bandera=0;

System.out.print("MENU\n1.- Llenar el arreglo con números"
    + "\n2.- Hacer búsqueda binaria\n3.- Terminar");

while(opcion!=3) {
   System.out.print("\n¿Qué desea hacer?: ");
   opcion=leer.nextInt();
   switch(opcion) {
      case 1: meter_numeros(arreglo);
              //indicador de que llené el arreglo
              bandera =1;
              break;
      case 2: /* bandera apagada significa que no he
                capturado numeros  */
             if (bandera==0)
                System.out.print("Imposible buscar, el "
                + "arreglo no ha sido llenado");
              else
                 busqueda_bin(arreglo);
   } //fin del switch
  }  // fin del while
}  //de main

//////// Método para meter números en el arreglo ////////
static void meter_numeros(int arreglo[]){
int i;
System.out.print("Recuerda, los números que des deben ser de menor a
mayor\n");

System.out.print("Número? ");
arreglo[0]=leer.nextInt();

for(i=1;i<arreglo.length ;i++) {
   System.out.print("Numero? ");
   arreglo[i]=leer.nextInt();
   /* para asegurar que los numeros dados se ingresen en
      orden ascendente  */
   if (arreglo[i-1]>arreglo[i]){
      System.out.print("Número ignorado, solo llevas " + i
```

```
                        + " números capturados\n");
        i--; // Para evitar el avance a la siguiente celda
    }
}
System.out.print("Listo, has capturado todo el arreglo, los elementos en él
actualmente son: \n");
for(i=0; i<arreglo.length; i++)
    System.out.print(arreglo[i]+" ");
}

//////// Método para hacer búsqueda binaria ////////
static void busqueda_bin(int arreglo[]){
int numero, inferior, superior, actual;
System.out.print("Qué numero deseas buscar en el arreglo?: ");
numero=leer.nextInt();

//Se inicializan límites inferior y superior del arreglo
inferior=0;
superior=arreglo.length-1;

// actual es el índice de la mitad del arreglo
actual=(inferior+superior)/2;
while(inferior<=superior && numero!=arreglo[actual]){
    if(arreglo[actual]<numero)
        inferior=actual+1;
    else
        superior=actual-1;
    actual=(inferior+superior)/2;
    }
//Se checa por qué se rompió el ciclo
if (numero==arreglo[actual])
    System.out.print("El número que buscas está en la celda "
                    + actual);
else
    System.out.print("El número que buscas no está en el "
                    +  "arreglo\n ");
 }
}// de la clase
```

```
run:
MENU
1.- Llenar el arreglo con números
2.- Hacer búsqueda binaria
3.- Terminar

¿Que desea hacer?: 1
Recuerda, los números que des deben ser de menor a  mayor
Número? 3
Número? 45
Número? 28
Número ignorado, solo llevas 2 números capturados
Número? 56
Número? 78
Listo, has capturado todo el arreglo, los elementos en él actualmente son:
3 45 56 78
¿Qué desea hacer?: 2
¿Qué número deseas buscar en el arreglo?: 56
El número que buscas está en la celda 2
¿Que desea hacer?: 2
¿Qué número deseas buscar en el arreglo?: 63
El número que buscas no está en el arreglo

¿Qué desea hacer?: 3
GENERACIÓN CORRECTA (total time: 1 minute 21 seconds)
```

El programa comienza mostrando un menú, del cual se puede escoger una de 3 opciones. Si se escoge la opción 1, se hace un llamado a la función `meter_numeros` con el argumento `arreglo`. Después del llamado al método `meter_numeros`, la *`bandera`* cambia de valor, como indicador de que ya se han introducido valores al arreglo. Si se elige la opción 2 se hace un llamado al método `busqueda_bin` también con el argumento `arreglo`. Es importante notar que existe una `bandera` que es inicializada con cero, como indicador de que la función `meter_numeros` todavía no se ejecuta. El valor de la variable *`bandera`* se verifica si se elige la opción 2. Lo anterior con el fin de evitar una búsqueda si no se han introducido valores en el arreglo previamente.

El método `meter_numeros` recibe como parámetro el arreglo que le fue enviado como argumento, para meter en él los valores leídos desde el teclado. Esté método se encarga de ir pidiendo uno a uno los valores que habrán de quedar en el arreglo, validando que los números que queden en el arreglo estén dados de manera ascendente (de menor a mayor). En caso de que el número que se acaba de leer sea menor que el anteriormente leído, el número no será permitido dentro del arreglo. Es por esa razón que existe la sentencia `i--`, para no permitir el avance hacia la siguiente celda. Una vez que se han llenado todas las celdas del arreglo de manera ascendente, se hace una impresión del arreglo para que el usuario vea cómo quedó.

El método `búsqueda_bin` busca un número indicado por el usuario dentro del arreglo de números, es por eso que recibe como parámetro el arreglo usado como argumento en el llamado a esta función. La búsqueda binaria logra el ahorro de pasos en la búsqueda de un número dentro de un arre-

glo tomando en cuenta solo la mitad del arreglo donde hay probabilidad de encontrarlo e ignorando la mitad donde no hay posibilidades de que se encuentre. Para el uso del método de búsqueda binaria es requisito que el arreglo donde se pretende encontrar un elemento esté ordenado. Se manejan 3 índices, un índice `inferior` que es el rango mínimo donde es posible encontrar el elemento buscado y un índice `superior` que es el rango máximo donde es posible encontrar el elemento buscado. A partir de ellos surge un índice `actual`, que representa la celda ubicada en el medio de los rangos máximo y mínimo. El índice `actual` se calcula dividiendo entre dos la suma de los rangos `superior` e `inferior`. La primera vez que se calcula el índice `actual`, el índice se referirá a la celda del medio del arreglo. Si el número buscado es igual al número ubicado en el índice `actual` del arreglo, la búsqueda ha terminado, en caso contrario se ignorará la mitad del arreglo donde no es posible encontrar el elemento buscado. Para ignorar la parte del arreglo donde no hay probabilidades de encontrar un número, uno de los índices, ya sea el `superior` o el `inferior` será modificado para calcular nuevamente el índice `actual`. Si el número que se busca es mayor que el elemento ubicado en el índice `actual`, entonces debemos ignorar la mitad izquierda del arreglo moviendo el índice `inferior` una celda a la derecha del índice actual (`inferior=actual+1`). Si el número que se busca es menor que el elemento ubicado en el índice `actual`, entonces se debe ignorar la mitad derecha del arreglo moviendo el índice `superior` una celda a la izquierda del índice `actual` (`superior=actual-1`). Una vez modificado el índice correspondiente de acuerdo a la situación, se vuelve a calcular el índice `actual` y si el elemento ubicado en esa celda del arreglo es igual al número que se busca, entonces el elemento ya fue encontrado, de lo contrario habrá que modificar nuevamente alguno de los índices. Esto se hace continuamente hasta encontrar el número buscado o hasta que el índice inferior resulte mayor que el superior, este último caso indicaría que el número que se busca no está dentro del arreglo. Este método de búsqueda se llama búsqueda binaria debido a que en cada comparación el arreglo se considera en dos partes: aquella donde es posible encontrar el elemento y aquella donde no es posible encontrarlo, ignorando la búsqueda en la mitad donde no hay probabilidades de que se encuentre.

5.3 Arreglos bidimensionales: conceptos básicos, operaciones y aplicaciones

Se puede imaginar un arreglo bidimensional como una tabla que tiene cierto número de renglones y columnas, o como un plano con una combinación de coordenadas (*x, y*).

Al tratar un arreglo que tiene dos dimensiones (*arreglo bidimensional o matriz*) se requieren dos índices para identificar a cada elemento dentro de él. Uno de los índices indica el renglón y un segundo índice indica la columna de cada elemento. Un arreglo bidimensional puede representarse así:

Columnas, en este caso 9

Renglones, en este caso 5	0	1	2	3	4	5	6	7	8
0									
1									
2									
3									
4									

Al igual que para los arreglos unidimensionales, para utilizar una matriz ésta debe declararse y después crearse. La manera general para declarar una matriz es:

```
tipo identificador [][];
```

Para la matriz mostrada anteriormente su declaración sería, para guardar valores enteros:

```
int matriz [][];
```

Recordar que al declarar una matriz no se está creando el espacio en memoria para almacenar sus elementos, únicamente se está creando una referencia. Es por eso que se necesita el paso siguiente:

```
matriz= new int [5][9];
```

Los dos pasos anteriores pueden fusionarse en uno solo, de la siguiente forma:

```
/* crea arreglo de 45 celdas en las cuales se pueden
   guardar números enteros */
int matriz [][]=new int [5][9];

/*crea un arreglo de 50 elementos para guardar cadenas de
  caracteres */
String grupos[][]=new String[10][5];
```

En la sintaxis para identificar un elemento en particular dentro de una matriz, el primer índice siempre indicará el renglón (o fila) y el segundo indicará la columna.

Almacenar valores en una matriz. Para almacenar valores en las diferentes celdas del arreglo bidimensional ahora se ocuparán dos índices:

```
/*se almacena el valor 100 en el renglón cero columna tres
  del arreglo */
matriz[0][3]=100;

/* se almacena el valor 850 en el renglón cuatro columna
   cuatro del arreglo   */
matriz [4][4]=850;
```

También se puede almacenar un valor leído desde el teclado:

```
/* se almacena el valor leído desde el teclado en el
   renglón uno columna dos del arreglo  */
matriz[1][2]=leer.nextInt();

/*se almacena el valor leído desde el teclado en el renglón
  cuatro columna tres del arreglo  */
matriz[4][3]=leer.nextInt();
```

```
/* si i=3 y j=5, se almacena el valor leído de teclado en
   la fila 8, columna 7 del arreglo altura */
altura [3*i-1][j+2]=leer.nextInt();
```

Igual que en los arreglos unidimensionales, en los arreglos bidimensionales resulta más conveniente almacenar valores dentro de un arreglo apoyándose en ciclos que controlen adecuadamente ambos índices de cada celda en el arreglo:

```
for(renglon=0 ; renglon<5 ; renglon++)
   for (col=0; col<9 ; col++){
      System.out.println("Dime el valor que irá en el renglón "+
renglon + "columna" +col);
      matriz[renglon][col]=leer.nextInt();
   }
```

Así la variable `renglon` comenzará en el valor de cero y la variable `col` iniciará en cero también. Como la variable `renglon` no se modificará sino hasta que el ciclo de `col` haya terminado, los valores leídos del teclado se irán almacenando renglón por renglón, hasta meter valores en todas las celdas del arreglo.

Para aplicar cualquier acción sobre un elemento en particular de un arreglo, bastará con hacer uso del nombre del arreglo y sus dos índices.

Atributo length. Un arreglo bidimensional es un arreglo de arreglos, en el ejemplo que se ha venido mencionando, el arreglo `matriz` es un arreglo de 5 arreglos, donde cada arreglo tiene 9 celdas. Es así como el atributo **`length`** puede utilizarse inmediatamente después del nombre de la matriz y nos indicará el número de arreglos (renglones), y al utilizarse después de los corchetes del primer índice nos indicará el número de columnas de ese renglón. *Recordar que el índice que está más cercano al nombre del arreglo nos indica el índice del renglón.*

Analizar las siguientes líneas de código y el resultado de su ejecución:

```
public class promedios_matriz {
public static void main(String[] args) {
int matriz[][]=new int [3][4];
int renglones,columnas;
renglones = matriz.length; //renglones tomará el valor de 3
columnas = matriz[0].length; // columnas tomará el valor de 4
System.out.println("La matriz tiene " + renglones +
                   " renglones y "+columnas+ " columnas"
                   + " por renglón");
 }
}
```

```
run:
La matriz tiene 3 renglones y 4 columnas por renglón
GENERACIÓN CORRECTA (total time: 2 seconds)
```

Se crea un arreglo de tres renglones y cuatro columnas por renglón; **`matriz.length`** indica el número de renglones que tiene el arreglo bidimensional `matriz` y ese valor se asigna a la variable `renglones`, **`matriz[0].length`** indica el número de columnas del renglón cero y ese valor se asigna a la variable `columnas`. Por eso al imprimir las variables renglones y columnas veremos los valores 3 y 4.

Inicializar un arreglo bidimensional. Para inicializar un arreglo bidimensional es necesario hacerlo renglón a renglón. Cada renglón se inicializa encerrando entre llaves los valores que irán en él; debe escribirse una coma entre los valores como separador. A su vez, cada renglón debe separarse de los demás renglones con coma y todos los renglones deben estar encerrados por otro par de llaves. Por ejemplo la siguiente sentencia estaría creando un arreglo de tres renglones y cuatro elementos en cada renglón, donde cada renglón contiene los valores que se muestran:

```
int bidi[][]={{4,5,1,180},{123,65,3,3},{2,3,6,9}};
```

El arreglo creado con la línea anterior podría representarse así:

	0	1	2	3
0	4	5	1	180
1	123	65	3	3
2	2	3	6	9

Entonces el valor 65 quedó ubicado en el renglón 1, columna 1. El valor 9 quedó ubicado en el renglón 2, columna 3, etc. En otras palabras, `bidi[1][1]` contiene el valor 65, `bidi[2][3]` contiene el valor 9, etc.

Si se desea crear un arreglo de tipo caracter de 2 renglones y 3 columnas, puede hacerse de la siguiente manera:

```
char caracteres[][]={{'a','a','a'},
                     {'b','b','b'}};
```

Al inicializar cada renglón en una línea diferente, la inicialización se acerca más a la forma de una tabla o matriz (visualmente), aunque para la creación del arreglo no afecta hacerlo en una sola línea o en varias. El arreglo de caracteres que se acaba de inicializar puede representarse así:

	0	1	2
0	a	a	a
1	b	b	b

De tal manera que las celdas `caracteres[0][0]`, `caracteres[0][1]` y `caracteres[0][2]` guardan el valor 'a' y las celdas `caracteres[1][0]`, `caracteres[1][1]` y `caracteres[1][2]` guardan el valor 'b'.

Ejemplo 5.5 Considerar la creación de un arreglo bidimensional de 3 renglones y 5 columnas por renglón. Inicializar el arreglo con valores enteros aleatorios en el rango de 0 a 500. Ese arreglo representa el número de cartas de felicitación que recibieron 3 voluntarios del cuerpo de bomberos de cinco distintas colonias. Se desea calcular el número de cartas recibidas por cada voluntario, así como las cartas escritas por colonia. El número de cartas recibidas por cada voluntario deberá guardarse en un arreglo unidimensional de 3, el número de cartas escritas en cada colonia deberá guardarse en un arreglo unidimensional de 5. Mostrar el contenido de los arreglos unidimensionales para conocer cuántas cartas recibió cada bombero y cuántas cartas se escribieron en cada colonia.

```
import java.util.Random;
public class bomberos {
public static void main(String[] args) {
// Se crea el objeto con el que se generarán numeros aleatorios
Random aleatorio=new Random();

int cartas[][]=new int [3][5];

//Se crea el arreglo voluntarios de tamaño 3
int voluntarios[]=new int [cartas.length];

//Se crea el arreglo colonias de tamaño 5
int colonias []=new int [cartas[0].length];

int vol,col;  //indices para controlar renglón y columna
for(vol=0; vol<cartas.length; vol++){
   for(col=0; col<cartas[vol].length; col++){
      //Se meten números aleatorios en las celdas del arreglo
      cartas[vol][col]=aleatorio.nextInt(501);
      // Se acumula el valor generado en el arreglo voluntarios
      voluntarios[vol]+= cartas[vol][col];
      // Se acumula el valor generado en el arreglo colonias
       colonias[col]+=cartas[vol][col];
   }
  }
// Se imprime el contenido del arreglo voluntarios
for(vol=0; vol<cartas.length; vol++)
     System.out.println("El bombero "+vol+" recibió un total de "+
voluntarios[vol]+ " cartas ");
System.out.println();
// Se imprime el contenido del arreglo colonias
for(col=0; col<cartas[0].length; col++)
     System.out.println("La colonia "+col+" escribió un total de "+
colonias[col]+ " cartas ");
}
```

```
run:
El bombero 0 recibió un total de 756 cartas
El bombero 1 recibió un total de 1193 cartas
El bombero 2 recibió un total de 1634 cartas

La colonia 0 escribió un total de 507 cartas
La colonia 1 escribió un total de 896 cartas
La colonia 2 escribió un total de 618 cartas
La colonia 3 escribió un total de 698 cartas
La colonia 4 escribió un total de 864 cartas
GENERACIÓN CORRECTA (total time: 2 seconds)
```

El programa comienza creando el objeto con el que se generarán los números aleatorios que representarán el número de cartas enviadas a cada bombero desde las diferentes colonias. A continuación se crean los tres arreglos que pide el problema; un arreglo bidimensional con el nombre `cartas` con dimensiones de 3×5, así como el arreglo unidimensional `voluntarios` de tamaño 3 y el arreglo unidimensional `colonias` de tamaño 5. Los dos arreglos unidimensionales servirán para almacenar la suma del número de cartas recibidas por voluntario y por colonia respectivamente. A continuación comienzan a generarse números de manera aleatoria en un rango de 0 y 500, y cada número generado va almacenándose en el arreglo `cartas` y acumulándose al mismo tiempo en los arreglos `voluntarios` y *`colonias`*.

Cuando se genera un número para `cartas[0][0]`, éste se acumula en `voluntarios[0]` y en `colonias[0]`; cuando se genera un número para `cartas[0][1]`, éste se acumula en `voluntarios[0]` y en `colonias[1]`; cuando se genera un número para `cartas[0][2]`, éste se acumula en `voluntarios[0]` y en `colonias[2]` y así sucesivamente, según van cambiando los valores de las variables que sirven como índices (`vol` y `col`). La ejecución ejemplo que se acaba de mostrar supone que los números que se generaron aleatoriamente para el arreglo `cartas` son:

cartas

46	89	416	26	179
68	420	148	306	251
393	387	54	366	434

Por lo tanto los valores en los otros dos arreglos son:

cartas

46	89	416	26	179
68	420	148	306	251
393	387	54	366	434

179
1193
1634

voluntarios

507	896	618	698	864

colonias

Es importante señalar que dos ejecuciones del mismo código mostrarán diferentes resultados debido a que los números del arreglo `cartas` son aleatorios. Después de haber llenado el arreglo `cartas` y acumulado los valores correspondientes en los arreglos unidimensionales, éstos últimos se imprimen para mostrar, como exige el problema, el número de cartas recibidas por cada bombero y por cada colonia.

Ejemplo 5.6. El cuadro mágico es un juego que consiste en acomodar los números enteros del 1 al n^2 (n debe ser impar), en un cuadro de n×n. De tal manera que la suma de los elementos sea igual en las líneas, en las columnas y en las diagonales.

Existe un algoritmo que permite hacer el acomodo de los elementos de acuerdo a lo siguiente:

a) Colocar el 1 en la primera fila de la columna del centro.

b) Los siguientes elementos se van colocando consecutivamente en la celda que se encuentra a la "**derecha**" y "**arriba**" de donde se colocó el número anterior. Si esa celda ya está ocupada, entonces se colocará en la celda debajo del número anteriormente colocado. Repetir este paso hasta haber llenado el arreglo.

c) Imprimir el contenido del arreglo para poder observar los valores que quedaron en él.

Considerar que si el índice de columna tiene el valor de (n-1) (límite máximo) ya no hay columna a la derecha, pues está fuera de los límites del arreglo. En ese caso el índice debe regresar a 0 para continuar dentro del arreglo. Lo mismo sucede para el índice del renglón, cuando éste tiene el valor de cero (límite mínimo) ya no hay renglón hacia arriba, pues resultaría en un valor negativo, por lo tanto el índice debe hacerse (n-1). A continuación se muestra los cuadros mágicos para n = 3 y n = 5.

n = 3

8	1	6
3	5	7
4	9	2

n = 5

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

```
import java.util.Scanner;
public class Cuadro_magico {
public static void main(String[] args) {
Scanner leer = new Scanner(System.in);
int n;
System.out.print("n: ");
n=leer.nextInt();  //lee dimensiones del cuadro
if (n%2==0 || n<3)
   System.out.println("Número no válido ");
else {
//crea arreglo para cuadro mágico
int arr[][]=new int[n][n];

//Se calculan el renglón y columna iniciales
int i=1,ren=0,col=arr.length/2;

do{
   if(arr[ren][col]!=0)    //si la celda ocupada
      {ren=ren+2;          //colocar debajo del ultimo
      if(ren>arr.length)
         ren=ren-arr.length;
      col--;
      if(col<0)
         col=arr.length-1;
      }
   arr[ren][col]=i; //meter consecutivo
   col++; // a la derecha
   if(col==arr.length)
```

```
      col=0;
   ren--; // arriba
   if(ren<0)   //¿renglón está en el límite inferior?
      ren=arr.length-1;
   i++; // incrementar consecutivo
   }while(i<=Math.pow(arr.length,2));

//Se imprimen los valores que quedaron en el arreglo
  for(i=0; i<arr.length; i++){
   for(int j=0;j<arr.length; j++)
       System.out.printf("%4d",arr[i][j]);
    System.out.println();
    }  // fin del for de i
  }  //llave del else
 }  // llave de main
} // llave de la clase
```

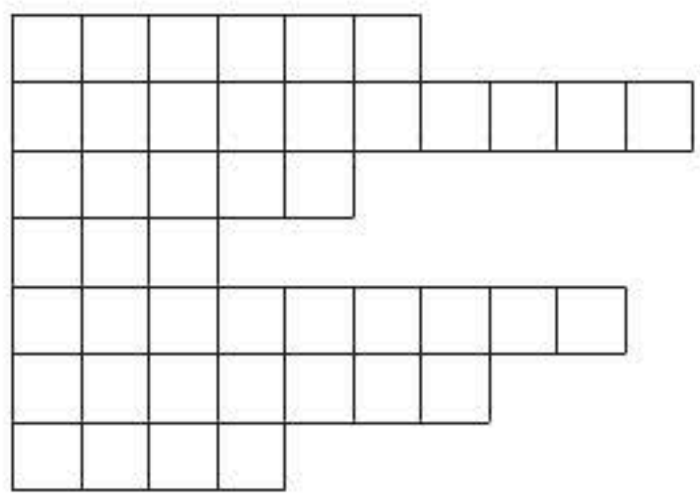
```
run:
n: 5
 17   24    1    8   15
 23    5    7   14   16
  4    6   13   20   22
 10   12   19   21    3
 11   18   25    2    9
GENERACIÓN CORRECTA (total time: 2 seconds)
```

El programa comienza solicitando el tamaño del cuadro mágico (n), si n es par o negativo manda el mensaje "Número no válido" y termina la ejecución del programa. Si n es positivo impar crea el arreglo con dimensiones n×n, por medio de la sentencia: `int arr[][]=new int[n][n];` esto permite manejar arreglos con las dimensiones requeridas con la finalidad de reservar solamente los lugares que se necesitan. La variable `ren` controlará el índice de los renglones del arreglo y la variable `col` controlará el índice de las columnas, `ren` comienza con el valor de cero y `col` con el valor de (n/2), para colocar el primer número en el renglón cero en la columna del centro de la tabla como lo indican las instrucciones para el llenado del arreglo. Después comienza un ciclo, cuya condición de terminación: `while(i<=Math.pow(arr.length,2)` indica que la variable `i` debe llegar a un valor máximo de n^2, que es el número de elementos a colocar en el arreglo. La primera sentencia del ciclo verifica si la celda con los índices `ren` y `col` dentro del arreglo está ocupada. Si es así se modifican los índices de manera que éstos tengan los valores de renglón y columnas correspondientes a la celda que está debajo del número anterior que fue colocado. Para eso `ren` se aumenta en 2, validando que no salga de los límites de arreglo, pues de ser así se le restará el número de renglones para que permanezca con un valor válido. Adicionalmente `col` disminuirá en 1, validando que *col* no llegue a ser un valor negativo, en cuyo caso se le asigna el límite superior del arreglo. En seguida se guarda el valor de `i` a la celda correspondiente, indicada por los índices `ren` y `col`. Para cumplir con el inciso (b) del algoritmo, `col` se incrementa en

1, que sería equivalente a ir una celda hacia la derecha del arreglo, cuidando de asignarle cero en caso de que llegue a n, pues (n-1) es el límite superior. Cumpliendo también con el inciso (b) del algoritmo, `ren` se disminuye en 1, lo cual equivale a ir una celda hacia arriba de la posición actual, cuidando de asignarle n en caso de que llegue a -1, pues 0 es el límite inferior. Después la instrucción `i++` provoca que se tenga listo el siguiente número consecutivo a ser almacenado en el arreglo. Y mientras el valor de `i` no llegue a n^2, se estará repitiendo el proceso. Por último se tienen dos ciclos `for` con los cuales se imprimen los números que quedaron en el arreglo.

5.3.1. Arreglos irregulares

Aunque es muy común que todos los renglones de un arreglo tengan el mismo número de columnas, Java permite la fácil creación de arreglos irregulares como el que se muestra:



Observar que se trata, en este ejemplo de un arreglo de 7 renglones, pero el número de columnas de cada renglón varía. El renglón cero tiene 6 celdas, el renglón 1 tiene10 celdas, etc.

Para crear un arreglo bidimensional irregular se debe declarar la referencia del arreglo y después crear individualmente para cada renglón la dimensión deseada (recordar que finalmente un arreglo bidimensional es un arreglo de arreglos). Si el arreglo anterior fuera para almacenar datos de tipo flotante, entonces su declaración y creación sería:

```
//Se declara un arreglo de 7 renglones
float arr[][]=new float [7][];

// Habrá seis celdas en la línea 0 del arreglo
arr[0]=new float [6];
arr[1]=new float [10];
arr[2]=new float [5];   // Cinco celdas en la línea 2
arr[3]=new float [3];
arr[4]=new float [9];   // Nueve celdas en la línea 4
arr[5]=new float [7];
arr[6]=new float [4];  // Cuatro celdas en la línea 6
```

Por lo tanto, al crear un arreglo unidimensional, para cada renglón del arreglo se puede especificar la dimensión de cada uno, como se hizo en el código anterior.

Ejemplo 5.7 Se desea registrar en un arreglo bidimensional el monto de los contratos logrados por una compañía de seguros. La compañía trabaja solamente 4 días de la semana, pero el número de contra-

tos logrados por día es variable y está en el rango de 0 a 20 contratos por día. Escribir un programa que registre en un arreglo el monto de cada contrato y la suma de esos contratos por cada día de la semana, de acuerdo al número de contratos que indique el usuario.

```
import java.util.Scanner;
public class contratos {
static Scanner leer=new Scanner(System.in);
public static void main(String[] args) {
//Se declara un arreglo de 4 renglones
float arr[][]=new float [4][];
int r,contratos;
float suma;
System.out.println("Ingrese el número de contratos logrados en
esta semana");

for(r=0;r<4;r++){
   System.out.print("Día "+(r+1)+" : ");
   contratos=leer.nextInt();

    //Si contratos no está en el rango 0-20
   if(contratos<0 || contratos >20)
      contratos=0;

   //Número de columnas por día
   arr[r]=new float [contratos+1];

   if(arr[r].length==1){
      System.out.println("SIN VENTAS");
      suma=0;
   }
   else{
    suma=0;
    for(contratos=0;contratos<arr[r].length-1;contratos++){
      System.out.print("Monto del contrato "+(contratos+1)+
                       " de "+(arr[r].length-1)+" : ");
      arr[r][contratos]=leer.nextFloat();
      suma+=arr[r][contratos];
      }
      arr[r][contratos]=suma;
   }
}

//Imprime tabla
System.out.println("Día\t Montos");
```

```
for (r=0; r<4; r++){
  System.out.printf("%3d\t",r+1);
  for (contratos=0; contratos<arr[r].length-1; contratos++)
      System.out.printf("%8.2f",arr[r][contratos]);
   System.out.printf("\tTotal = %.2f\n",arr[r][contratos]);
 }
}
}
```

```
run:
Ingrese el número de contratos logrados en esta semana
Dia 1 : 2
Monto del contrato 1 de 2 : 4570.54
Monto del contrato 2 de 2 : 3680.30
Dia 2 : 1
Monto del contrato 1 de 1 : 8900.60
Dia 3 : 0
SIN VENTAS
Dia 4 : 3
Monto del contrato 1 de 3 : 5040.40
Monto del contrato 2 de 3 : 1900.55
Monto del contrato 3 de 3 : 4730.85
Día     Montos
 1      4570.54    3680.30    Total = 8250.84
 2      8900.60          Total = 8900.60
 3                  Total = 0.00
 4      5040.40    1900.55    4730.85    Total = 11671.80
GENERACIÓN CORRECTA (total time: 2 minutes 32 seconds)
```

Con la sentencia **float** `arr[][]`=**new float** `[4][]` se declara un arreglo bidimensional de 4 renglones, pero aún no se crea el espacio para cada uno de los renglones. Cada renglón indicaría un día de la semana. Al pedir al usuario que introduzca el número de contratos que se lograron en cada día de la semana, se leerá un valor que será almacenado por la variable `contratos`, ésta es quien definirá la dimensión que tendrá cada uno de los renglones del arreglo con la instrucción `arr[r]`=**new float** `[contratos+1]`. Dependiendo del valor de la variable `contratos`, se irá formando la dimensión de cada renglón del arreglo, por lo que puede ser un arreglo irregular, pues de acuerdo al problema no todos los días se logra el mismo número de contratos, se le suma uno para la celda que guardará el importe total de todos los contratos del día. A continuación se pide ingresar el monto de cada uno de los contratos. Como cada renglón es de diferente tamaño, entonces se hace especialmente importante el uso del atributo **length** del arreglo, pues éste definirá de manera automática el valor máximo del índice que se usará para la columna de cada renglón. Con la información de la salida anterior el arreglo `arr` tiene la siguiente forma:

arr

	0	1	2	3
0	4570.54	3680.30	8250.84	
1	8900.60	8900.60		
2	0.00			
3	5040.40	1900.55	4730.85	11671.80

El día 1 contrato 2 es la información de la celda `arr[0][1]=3680.30` y el importe total de los contratos para ese día es `arr[0][2]=8250.84,` considerando que los arreglos comienzan en la fila cero y columna cero. En forma general el día es la fila y los contratos son las columnas. Además de que el importe total de los contratos está en la última celda de cada fila. De esta tabla es posible consultar el monto de un contrato determinado o el importe total de todos los contratos de un día.

5.3.2 Arreglos como argumentos y parámetros

Cuando se tocó el tema del paso de parámetros se mencionó que existe el paso de parámetros por valor y el paso de parámetros por referencia. Recordar que un identificador *solo es conocido dentro del contexto en que fue creado;* si se requiere que sea conocido fuera de ese contexto, entonces deberá ser recibido como parámetro en ese otro contexto (función). Los arreglos en Java son tratados como objetos y todos los objetos se pasan por referencia, es decir, al pasar un arreglo como parámetro a una función se está pasando la referencia misma, en otras palabras, se está pasando el bloque de memoria que contiene el arreglo mismo y no una copia de él, por lo que al hacer una modificación al arreglo dentro de la función que lo recibe como parámetro, esa modificación quedará permanente, pues no se está modificando una copia del arreglo sino el arreglo mismo. Si se tiene:

```
int A = 5;
int B;
B = A;
```

Entonces tanto `A` como `B` tienen el mismo valor. Si después se hace la asignación `B=10`, el valor de `A` seguirá siendo `5` pues fue `B` y no `A` quien se modificó. Con los arreglos debe tenerse cuidado en este tipo de sentencias ya que los arreglos son referencias y una asignación de referencias significa referirse al mismo segmento de memoria, o sea al mismo arreglo.

Ejemplo 5.8. Escriba un programa que haga la creación de un arreglo de caracteres inicializándolo con las vocales en minúscula. Crear otro arreglo bidimensional irregular de enteros, también inicialícelo. Demuestre que a través de otra referencia es posible modificar sus elementos.

```
public class Referencias_Arreglos {

public static void main(String[] args) {
char vocales[]={'a','e','i','o','u'};
char otro[];
int nums[][]={{1,2,3},{5,6,7,8,10}};
int nums2[][],r,c;
```

```
nums2=nums;  //nums2 ahora se refiere al mismo arreglo

otro=vocales; /* No se copian, ahora tanto otro como vocales
                se refieren al mismo arreglo */

System.out.println("Se imprime arreglo de caracteres a través de
la referencia otro");

for(r=0;r<otro.length;r++)
   System.out.print(otro[r]+ " ");
otro[0]='A';

System.out.print("\n\nDespués de la instrucción otro[0]='A';");

System.out.println("\nSe imprime arreglo de caracteres a través
de la referencia vocales");

for(r=0;r<vocales.length;r++)
   System.out.print(vocales[r]+ " ");

System.out.println("\n\nSe imprime arreglo de enteros a través
de nums2");

for(r=0;r<nums2.length;r++)
   for (c=0;c<nums2[r].length;c++)
      System.out.print("\nValor nums2["+r+"]["+c+"] es"
                       +nums2[r][c]);

System.out.println();
} //fin del método main
} // fin de la clase
```

```
run:
Se imprime arreglo de caracteres a través de la referencia otro
a e i o u

Después de la instrucción otro[0]='A';
Se imprime arreglo de caracteres a través de la referencia vocales
A e i o u

Se imprime arreglo de enteros a través de nums2
Valor nums2[0][0] es 1
```

(continúa en la siguiente página)

```
Valor nums2[0][1] es 2
Valor nums2[0][2] es 3
Valor nums2[1][0] es 5
Valor nums2[1][1] es 6
Valor nums2[1][2] es 7
Valor nums2[1][3] es 8
Valor nums2[1][4] es 10

GENERACIÓN CORRECTA (total time: 2seconds)
```

(continúa de la página 220)

Con la instrucción `nums2=nums` ambas referencias estarán manejando el mismo arreglo. Lo mismo pasa con la instrucción `otro=vocales`. Si el objetivo es hacer una copia de un arreglo, entonces deberán crearse 2 arreglos y luego hacer una asignación de elemento por elemento:

```
public class Copia_Arreglos {

public static void main(String[] args) {
char vocales[]={'a','e','i','o','u'};
char otro[]=new char[5];
int r;

//Se crea una copia de vocales en otro
for(r=0; r<vocales.length; r++)
   otro[r]=vocales[r];

otro[0]='A';
System.out.println("Se imprimen arreglos de caracteres"
                   + " después de otro[0]='A';");

for(r=0;r<vocales.length;r++)
   System.out.println("vocales["+r+"]es: " + vocales[r]+
                      "   otro["+r+"] es: " +otro[r]);
System.out.println();
    }
}
```

```
run:
Se imprimen arreglos de caracteres después de otro[0]='A';
vocales[0]es: a   otro[0] es: A
vocales[1]es: e   otro[1] es: e
vocales[2]es: i   otro[2] es: i
vocales[3]es: o   otro[3] es: o
vocales[4]es: u   otro[4] es: u

GENERACIÓN CORRECTA (total time: 2seconds)
```

En el ejemplo anterior se hace una copia uno a uno de los elementos del arreglo `vocales` sobre el arreglo `otro`. Como ahora hay dos arreglos con los mismos valores, uno referenciado por `vocales` y el segundo por la variable `otro`, la asignación `otro[0]='A'` no tiene efecto sobre el arreglo vocales.

Otra forma de hacer una copia del arreglo es a través del método `arraycopy` perteneciente a la clase `System`, de la siguiente manera:

```
public class Copia_Arreglos {
public static void main(String[] args) {
char vocales[]={'a','e','i','o','u'};
char otro[]=new char[5];
int r;

//Se copia el arreglo vocales en el arreglo otro
System.arraycopy(vocales,0,otro , 0, 5);
otro[3]='Z';

System.out.println("Se imprimen arreglos de caracteres después
de otro[3]='Z';");
for(r=0;r<vocales.length;r++)
   System.out.println("vocales["+r+"]es: " + vocales[r]
                      +  " otro["+ r+"] es: " +otro[r]);
System.out.println();
 }
}
```

```
run:
Se imprimen arreglos de caracteres después de otro[3]='Z';
vocales[0]es: a   otro[0] es: a
vocales[1]es: e   otro[1] es: e
vocales[2]es: i   otro[2] es: i
vocales[3]es: o   otro[3] es: Z
vocales[4]es: u   otro[4] es: u

GENERACIÓN CORRECTA (total time: 0 second
```

La sentencia `System.arraycopy(vocales,0,otro , 0, 5)` copia 5 elementos del arreglo `vocales` comenzando desde la posición cero hacia el arreglo `otro` en la posición cero. Los argumentos de la llamada significan:

`vocales` que indica el arreglo origen de donde se tomarán los valores, el segundo argumento (un cero) indica la posición en el arreglo `vocales` del primer elemento a copiar, el tercer argumento es el nombre del arreglo destino hacia donde se copiarán los datos, el siguiente argumento (otro cero) indica la posición del arreglo destino en la cual comenzarán a copiarse los valores y el último argumento indica el número de elementos que se copiarán. Después de haber copiado el arreglo `vocales` en el arreglo

otro, la instrucción otro[3]='Z' modifica el elemento tres del arreglo otro, pero este cambio no trasciende al arreglo vocales, pues se trata de dos arreglos diferentes.

Paso por referencia y por valor en métodos y funciones. En Java las variables simples se pasan *por valor*, lo cual significa que los cambios que sufran dichas variables dentro de un método solamente son conocidos dentro de ese método. Pero los arreglos se pasan *por referencia* lo cual implica que los cambios que se le hagan a dichos arreglos dentro del método realmente los registra el arreglo que se mandó, porque no se crean nuevos arreglos.

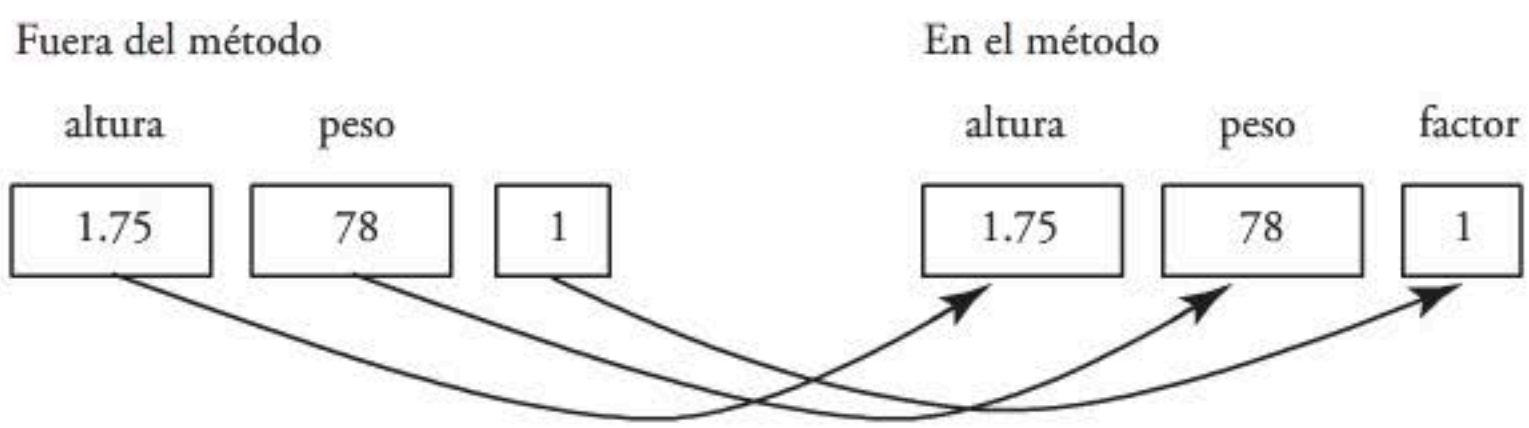
Ejemplo 5.9. Definir una matriz de 3 filas y 4 columnas asignándole valores inicialmente. Posteriormente invocar un método enviando como parámetros a la matriz, dos variables numéricas y una constante. Dentro del método multiplicar el arreglo y las variables por la constante e imprimir el resultado. Al salir del método imprimir nuevamente las variables para observar el cambio.

```
import java.util.Scanner;
public class Paso_por_referencia {
public static void main(String[] args) {
int arr[][]={{2,0,3,-4},{5,6,-1,3},{8,-2,7,9}};
double altura;
int peso;

altura=1.75;
peso=78;

//Multiplica la matriz y la variable altura por (1) y los imprime
System.out.println("Matriz multiplicada por: 1");
multiplica(arr,altura,peso,1);

//Cuando sale del método imprime altura y peso
System.out.println("Altura (fuera)="+altura
                   +"  Peso(fuera)="+peso);

//Multiplica la matriz y la variable altura por (-3) y los imprime
System.out.println("Matriz multiplicada por: -3");
multiplica(arr,altura,peso,-3);

//Cuando sale del método imprime altura y peso
System.out.println("Altura (fuera)="+altura
                  + "  Peso(fuera)="+peso);

/* Con otro método que hace lo mismo que el método Multiplica pero
los parametros se llaman diferente a las variables */
System.out.println("Matriz multiplicada por: 2");
otra_multiplica(arr,altura,peso,2);
```

```
//Cuando sale del método imprime altura y peso
System.out.println("Altura (fuera)="+altura
                   +"  Peso(fuera)="+peso);
} //fin de programa principal

/* Método para multiplicar el arreglo y los parámetros altura y
peso por un factor e imprimir el resultado */
static void multiplica(int [][] arr, double altura, int peso, int
factor){
  int i, j;
  for (i=0; i<arr.length; i++){
    for (j=0; j<arr[0].length; j++){
      arr[i][j]=arr[i][j]*factor;
      System.out.printf("%4d",arr[i][j]);
      }
      System.out.println();
    }
  altura=altura*factor;
  peso=peso*factor;
  System.out.println("Altura (dentro)="+altura
                     +"  Peso(dentro)="+peso);
 }

 /*Método para multiplicar el arreglo y los parámetros altura y
peso por un factor e imprimir el resultado */
static void otra_multiplica(int [][] x, double y, int z, int f){
  int i, j;
  for (i=0; i<x.length; i++){
    for (j=0; j<x[0].length; j++){
      x[i][j]=x[i][j]*f;
      System.out.printf("%4d",x[i][j]);
      }
      System.out.println();
  }
  y=y*f;
  z=z*f;
  System.out.println("Altura (dentro)="+y
                     +"  Peso(dentro)="+z);
}
}
```

```
run:
Matriz multiplicada por: 1
    2        0        3       -4
    5        6       -1        3
    8       -2        7        9
Altura (dentro)=1.75  Peso (dentro)=78
Altura (fuera)=1.75  Peso (fuera)=78
Matriz multiplicada por: -3
   -6        0       -9       12
   15       18        3       -9
  -24        6      -21      -27
Altura (dentro)=-5.25  Peso (dentro)=-234
Altura (fuera)=1.75  Peso (fuera)=78
Matriz multiplicada por: 2 otro método
  -12        0      -18       24
  -30      -36        6      -18
  -48       12      -42      -54
Altura (dentro)=3.5  Peso (dentro)=156
Altura (fuera)=1.75  Peso (fuera)=78
GENERACIÓN CORRECTA (total time: 0 seconds)
```

En el programa anterior se invoca el método: `multiplica(arr,altura,peso,1);` enviando como argumentos al arreglo: `arr`, las variables: `altura, peso` y la constante: `1`, mismos que fueron recibidos por los parámetros `arr` para la matriz y los parámetros: `altura, peso y factor` para recibir el valor de las variables y constante respectivamente. Aunque se llaman igual Java crea nuevos espacios de memoria para las nuevas variables numéricas `altura, peso y factor` que serán conocidas dentro del método, pero no crea ningún espacio para el arreglo, sino que trabaja con el espacio de memoria que ya tiene, como se muestra en la siguiente figura:

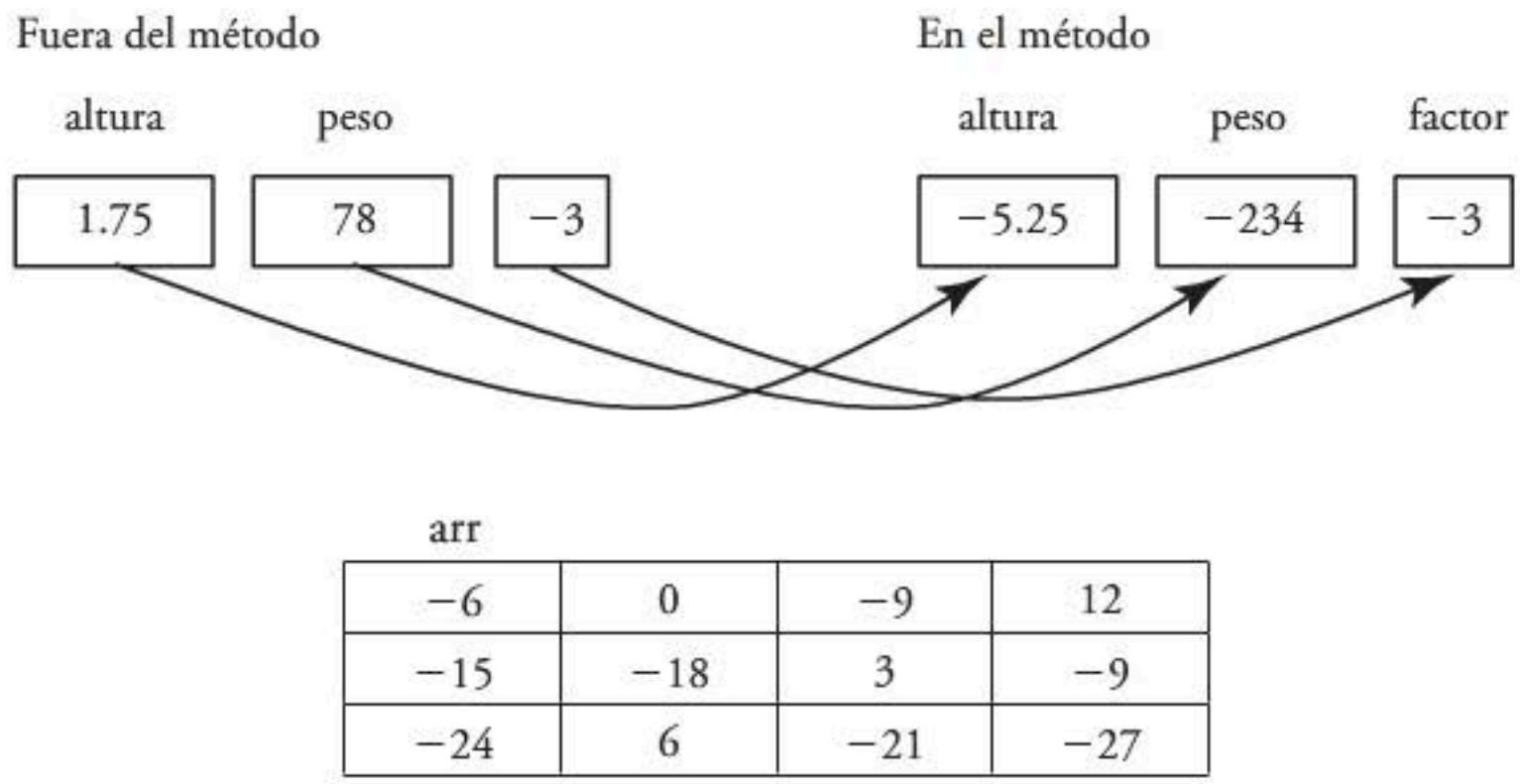


arr

2	0	3	−4
5	6	−1	3
8	−2	7	9

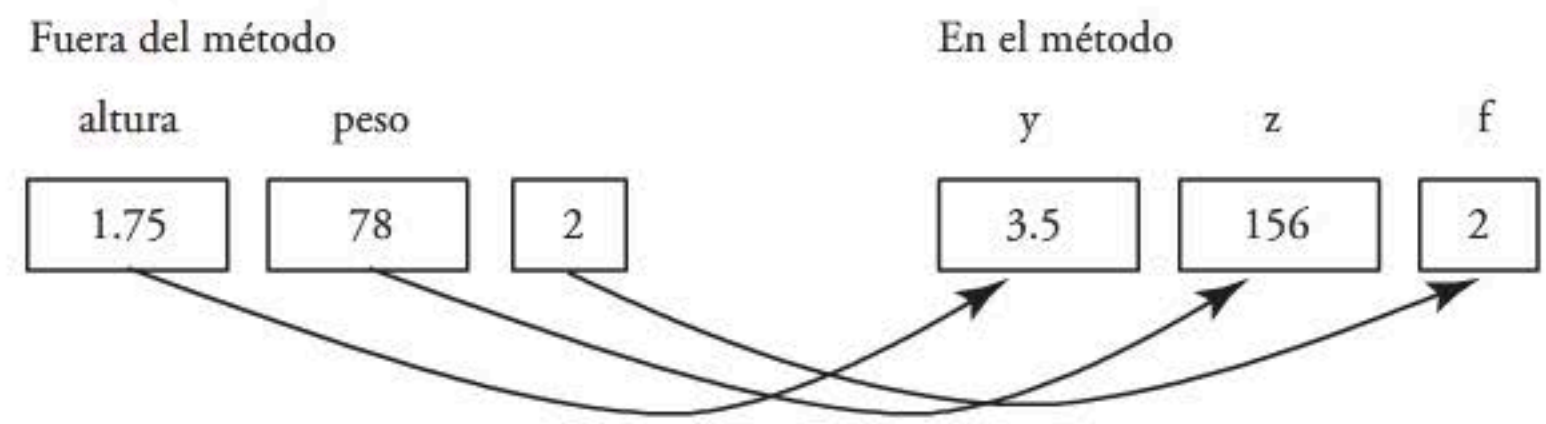
Cuando se llama: `multiplica(arr,altura,peso,1);` con factor igual a `1` las variables y el arreglo no sufren cambio alguno al multiplicarlos, y por lo tanto el valor es el mismo dentro y fuera del método.

Pero cuando se llama: `multiplica(arr,altura,peso,-3);` después de multiplicar el arreglo y los parámetros por el factor `-3`, los valores de arreglo, variables y parámetros son como se muestra a continuación:



Al salir del método se destruyen los espacios de memoria de los parámetros `altura, peso` y `factor` que se usaron dentro del método, por esa razón cuando se imprimen las variables fuera del método que llevan el mismo nombre `altura` y `peso` siguen conservado su valor inicial de `1.75` y `78` respectivamente. Lo que sucede es que los arreglos se pasan por referencia y por lo tanto se trabaja con los valores del arreglo pero no se crea un arreglo adicional para recibir los valores que se le mandan. Sin embargo las variables `altura, peso` y la constante se pasan por valor y la computadora usa nuevos espacios de memoria para cada uno de los parámetros, pero los valores de los parámetros solamente son conocidos dentro del método.

Finalmente cuando se llama el método: `otra_multiplica(arr,altura,peso,2);` los parámetros del método `otra_multiplica(int [][] x, double y, int z, int f)` tienen nombre diferente , algo que es permitido en leguaje de programación, siempre y cuando los parámetros respeten la posición y el tipo de dato, el nombre puede ser diferente. La imagen de la información después de llevar a cabo la multiplicación, pero antes de salir del método es como se muestra a continuación.



arr

−12	0	−18	24
−30	−36	6	−18
−48	12	−42	−54

Realmente los métodos: `multiplica` y `otra_multiplica` hacen exactamente lo mismo, (multiplicar la información por cierto factor e imprimirla) de tal manera que no era necesario crear otro método para llevar a cabo la misma actividad, con el primer método es suficiente para tener la misma salida, pero en este caso se hizo con diferentes métodos para mostrar que es posible recibir la información en parámetros con diferente nombre de arreglo y variables. Observar también que el parámetro `x` recibe la referencia del arreglo `arr`, pero como se trata de un arreglo, se trabaja con el espacio de memoria de `arr` sin crear nuevo espacio de memoria, aunque el parámetro tenga nombre distinto.

Ejemplo 5.10. Existen varios métodos para ordenar información, pero uno de los más usados cuando el número de elementos no es tan grande es el "Sort de la Burbuja", que consiste en comparar los elementos de un vector subiendo de posición los elementos menores y bajando los mayores. En la primera pasada hace (n-1) comparaciones, en la segunda realiza (n-2) comparaciones, porque se considera que el elemento de la última posición ya está en su lugar, en la tercer pasada las comparaciones son (n-3) porque ya no toma en cuenta el penúltimo elemento, debido a que ya está en su lugar, y así sucesivamente. El sort termina cuando ya no hay intercambios de elementos, de tal manera que se registran los intercambios, para llevar el control. Las comparaciones totales que realiza el sort dependen de la forma en que estén colocados los elementos inicialmente, el conjunto de elementos podría quedar ordenado en la primera pasada si no se registra ningún intercambio, pero en el peor de los casos hace las siguientes comparaciones:

$$\frac{n(n-1)}{2}$$

A continuación se muestra como quedan los elementos después de las primeras dos pasadas, para un grupo de n=5 elementos. Los elementos que se comparan están sombreados. Observar como después de la primera pasada el elemento de la última posición ya está en su lugar, y después de la segunda pasada, los dos últimos elementos ya ocupan el lugar definitivo.

3	3	3	3	3
7	7	−2	−2	−2
−2	−2	7	7	7
8	8	8	8	0
0	0	0	0	8

(n − 1) comparaciones

3	−2	−2	−2
−2	3	3	3
7	7	7	0
0	0	0	7
8	8	8	8

(n − 2) comparaciones

Escribir un programa para leer un conjunto de n elementos y ordenarlos en forma ascendente, usando para ello el sort de la burbuja.

```
import java.util.Scanner;
public class Sort_burbuja {
public static void main(String[] args){
    Scanner leer = new Scanner(System.in);
    int n;

    System.out.print("n: "); n=leer.nextInt();
    int arr []=new int[n];  //Crea el arreglo con n celdas

    //Lee los números y los coloca en el arreglo
    leerlos(arr);

    ordenalos(arr);       //Ordena el arreglo

     System.out.println("Ordenados ");
     imprime(arr);        //Imprime el arreglo ordenado
  }

    //Método para leer los elementos y colocarlos en el arreglo
    static void leerlos(int [] x){
    Scanner leer = new Scanner(System.in);
    System.out.println("Dame los "+x.length+" datos: ");
    for (int i=0; i<x.length; i++)
        x[i]=leer.nextInt();
    }

    //Método ordenar el arreglo en forma ascendente
    static void ordenalos(int [] x){
    int i=1, c, pos, t;
    c=x.length;      //Comparaciones a realizar
    while (i>0){ //sale del ciclo cuando no hay intercambios
      i=0;           //Intercambios
      c=c-1;       //disminuye una comparación en una pasada
      pos=0;      //Posición del elemento en el arreglo

    //ciclo para controlar los intercambios
    while (pos<c){
        if (x[pos]>x[pos+1]){
          t=x[pos];  //t guarda el dato temporalmente
          //Cambia el dato de abajo hacia arriba
          x[pos]=x[pos+1];
          x[pos+1]=t;        //toma el dato que tiene t
          i++;               //incrementa intercambios
          }
        pos++;
```

```
    }
  }
}

//Método para imprimir el arreglo
static void imprime(int [] arr){
  for (int i=0; i<arr.length; i++)
     System.out.println(arr [i]);
   }
 }
```

```
run:
n: 5
Dame los 5 datos:
12
−3
0
57
9
Ordenados
−3
0
9
12
57
GENERACIÓN CORRECTA (total time: 31 seconds)
```

En el programa se define el arreglo `arr` para guardar números enteros en `n` celdas por medio de la instrucción: **`int`** `arr []=`**`new int`**`[n];`. Es una forma dinámica de definir el arreglo ya que se reservan exclusivamente los lugares necesarios. El programa tiene tres métodos, uno para leer los datos, otra para ordenarlos y otro para imprimirlos. En los tres métodos el único parámetro es el propio arreglo. Para leer se invoca el método por medio de la instrucción `leerlos(arr);`. El arreglo está vacío y lo recibe el parámetro `x` que también se define como arreglo, para que pueda recibir la referencia del arreglo `arr` y guarde los datos leídos de teclado en `arr`. En general los cambios se hacen en el arreglo que se manda, en eso consiste el paso por referencia. El método encargado de ordenar la información es: **`static void`** `ordenalos(`**`int`** `[] x);` que recibe el arreglo con los datos ya que anteriormente fueron leídos los datos, pero están desordenados. Para ordenarlos compara el primer elemento del arreglo con el segundo, si el segundo es mayor que el primero realiza el intercambio entre ellos y en caso contrario deja los elementos en su misma posición. Después compara el segundo elemento con el tercero y si el tercero es mayor que el segundo los intercambia y así sucesivamente. Una vez que le da una pasada a todo el arreglo, considera que el elemento que está en la última posición ya está colocado en su lugar y comienza nuevamente la segunda pasada, comparando el primer elemento del arreglo con el segundo, el segundo con el tercero, registrando cuando se hacen intercambios, ya que el *sort de la burbuja* termina cuando ya no se hace intercambio alguno.

Ejemplo 5.11. Para sumar dos matrices A y B se requiere que ambas tengan las mismas dimensiones (mismo número de filas y mismo número de columnas). Los elementos de la matriz resultante de la suma, se obtienen sumando el primer elemento de la matriz A con el primer elemento de la matriz B, para obtener el primer elemento de la matriz C, después sumar el segundo elemento de la matriz A con el segundo de la matriz B para obtener el segundo elemento de la matriz C y así sucesivamente.

$$\overset{{}_2A_3}{\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{vmatrix}} + \overset{{}_2B_3}{\begin{vmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{12} & b_{23} \end{vmatrix}} = \overset{{}_2C_3}{\begin{vmatrix} c_{11} & c_{12} & c_{13} \\ c_{11} & c_{12} & c_{13} \end{vmatrix}}$$

Donde: $c_{11} = a_{11} + b_{11}$

Escribir un programa para generar aleatoriamente los elementos de dos matrices cualquiera A y B. Obtener la suma de las dos matrices en la matriz C. Mandar un mensaje en caso en que no sea posible realizar la suma porque las dimensiones de las matrices no coinciden.

```
import java.util.Random;
import java.util.Scanner;

public class Suma_matrices {
 public static void main(String[] args){

    Scanner leer = new Scanner(System.in);

    int fa, ca, fb, cb;

    System.out.print("Filas de A: "); fa=leer.nextInt();
    System.out.print("Columnas de A: "); ca=leer.nextInt();
    int a[][]=new int[fa][ca];  //Dimensiones de matriz A

    leemat(a); //Lee la matriz A

    System.out.println("Matriz A ");
    imprimemat (a);

    System.out.print("Filas de B: "); fb=leer.nextInt();
    System.out.print("Columnas de B: "); cb=leer.nextInt();
    int b[][]=new int[fb][cb];  // Dimensiones de matriz B

    leemat(b);

    System.out.println("Matriz B ");
    imprimemat (b);
```

```
    if(fa==fb && ca==cb){
       int c[][]=new int[fa][ca]; // Dimensiones de matriz C
       sumalas(c,a,b);
       System.out.println("Matriz C ");
       imprimemat (c);
       }
    else
      System.out.println("No se pueden sumar ");
    }

   //Método para leer las matrices
   static void leemat(int [][] x){
   //crea objeto para generar números
   Random aleatorio=new Random();
    for (int i=0; i<x.length; i++)
     for (int j=0; j<x [i].length; j++)
        //Genera elementos de la matriz
        x[i][j]=aleatorio.nextInt(21)-5;
    }

   //Método para imprimir las matrices
   static void imprimemat (int [][] x){
    for (int i=0; i<x.length; i++){
     for (int j=0; j<x[i].length; j++)
       //imprime elementos
      System.out.printf("%4d",x[i][j]);
      System.out.println();    //Salta a la siguiente línea
     }
    System.out.println(); // Deja salto de línea entre matrices
   }

   //Método para sumar las matrices
   static void sumalas(int [][] c, int [][] x, int [][] y){
    for (int i=0; i<x.length; i++)
     for (int j=0; j<x[i].length; j++)
       //suma los elementos correspondientes
       c[i][j]=x[i][j]+y[i][j];        }
   }
```

```
run:
Filas de A: 2
Columnas de A: 3
Matriz A
 13      12      12
 -2      -1       4

Filas de B: 2
Columnas de B: 3
Matriz B
 -4      -5       9
 -1       1       4

Matriz C
  9       7      21
 -3       0       8

GENERACIÓN CORRECTA
(total time: 17 seconds)
```

```
run:
Filas de A: 3
Columnas de A: 4
Matriz A
 11       0      -5       0
  2       5       5       3
  6       5       0      10

Filas de B: 4
Columnas de B: 3
Matriz B
  5       0      15
 10      -3       9
 -4       9       0
  7       8       1

No se pueden sumar
GENERACIÓN CORRECTA
(total time: 9 seconds)
```

En el programa anterior se creó la matriz "`a`" después de que se conoció el número de filas (`fa`) y columnas (`ca`) de dicha matriz usando la instrucción: **`int`** `a[][]=`**`new int`**`[fa][ca];`. Lo mismo se hizo para crear las matrices `b` y `c` que se crearon hasta que fueron conocidas sus dimensiones. Se puede observar también que se utilizó un solo método para leer las dos matrices: **`static void`** `leemat(`**`int`**` [][] x)` y el único parámetro que se le mando, fue la matriz a leer en el momento en que fue invocado dicho método aprovechando el paso por referencia. La ejecución del método se hizo con la instrucción: `leemat(a);` que manda como único parámetro la matriz `a,` con esto le está enviando el espacio en memoria de dicha matriz, pero también sus dimensiones, aunque en el método el nombre de dicha matriz que recibe la información puede llamarse igual o diferente, como ocurrió en este caso que se le llamó x al parámetro que recibe la matriz. Ya en el método se generaron elementos enteros entre -5 y 15 que se guardaron en la matriz "`x`", pero como dicha matriz tenía la referencia de la matriz "`a`", realmente se estaban guardando los elementos en la matriz "`a`". También se usó un solo método para imprimir las tres matrices `a,` `b` y `c` aprovechando el paso por referencia. Observar que la combinación de métodos y funciones y el paso por referencia permite aprovechar y optimizar el código de un programa.

5.4 Arreglos multidimensionales: conceptos básicos, operaciones y aplicaciones.

No son muy comunes las situaciones donde se requiera almacenar datos en arreglos de más de dos dimensiones, pero sí existen. Por ejemplo los arreglos tridimensionales se pueden imaginar en forma de cubo.

Dicho en otras palabras, son arreglos bidimensionales puestos uno sobre otro. Si imaginamos una caja que contiene diferentes charolas de chocolates, cada charola sería el plano, y en cada charola habría cierto número de chocolates acomodados en filas y columnas.

Ejemplo 5.12. Crear un arreglo tridimensional llamado `chocolates`, inicializarlo con datos e imprimir el contenido de cada una de las celdas de dicho arreglo:

```
public class Arr_Tridim {

public static void main(String[] args) {
//primer plano, arreglo de 2 X 4
//segundo plano, arreglo de  4 X 2
// tercer plano, arreglo de 3 X 3

int chocolates[][][]={{{1,2,3,4},{5,6,7,8}},
                      {{9,10},{11,12},{13,14},{15,16}},
                      {{17,18,19},{20,21,22},{23,24,25}}};
int p,r,c;
int n=0;   //para imprimir en dos columnas
for(p=0;p<chocolates.length;p++){  //Ciclo de los planos
  System.out.println("\nPLANO/CHAROLA NÚMERO : "+p);
  //Ciclo de los renglones del plano p
  for(r=0;r<chocolates[p].length;r++)
    // Ciclo de las columnas del plano p del renglón r
    for(c=0;c<chocolates[p][r].length;c++){
      if(n%2==0)
          System.out.print("chocolates["+p+"]["+r+"]["+c
                          +"]contiene el: "
                           +chocolates[p][r][c]);
        else
          System.out.println("chocolates["+p+"]["+r+"]["
                             +c + "]contiene el: "
                             +chocolates[p][r][c]);
        n++;
      }
   }
  System.out.println();
 }
}
```

```
run:
PLANO/CHAROLA NÚMERO : 0
chocolates[0][0][0]contiene el: 1  chocolates[0][0][1]contiene el: 2
chocolates[0][0][2]contiene el: 3  chocolates[0][0][3]contiene el: 4
chocolates[0][1][0]contiene el: 5  chocolates[0][1][1]contiene el: 6
chocolates[0][1][2]contiene el: 7  chocolates[0][1][3]contiene el: 8
```

(continúa en la siguiente página)

```
PLANO/CHAROLA NÚMERO : 1
chocolates[1][0][0]contiene el: 9  chocolates[1][0][1]contiene el: 10
chocolates[1][1][0]contiene el: 11  chocolates[1][1][1]contiene el: 12
chocolates[1][2][0]contiene el: 13  chocolates[1][2][1]contiene el: 14
chocolates[1][3][0]contiene el: 15  chocolates[1][3][1]contiene el: 16

PLANO/CHAROLA NÚMERO : 2
chocolates[2][0][0]contiene el: 17  chocolates[2][0][1]contiene el: 18
chocolates[2][0][2]contiene el: 19  chocolates[2][1][0]contiene el: 20
chocolates[2][1][1]contiene el: 21  chocolates[2][1][2]contiene el: 22
chocolates[2][2][0]contiene el: 23  chocolates[2][2][1]contiene el: 24
chocolates[2][2][2]contiene el: 25
GENERACIÓN CORRECTA (total time: 0 seconds)
```

(continúa de la página 233

En el código anterior se crea e inicializa un arreglo de 3 planos, en primer plano se trata de un arreglo bidimensional de dos renglones por cuatro columnas por renglón, el segundo plano contiene un arreglo de cuatro renglones por dos columnas por renglón y el tercer plano tiene un arreglo de tres renglones por tres columnas por renglón. Recordar que el primer índice (el más cercano al nombre del arreglo) se refiere al plano, el segundo índice a las filas o renglones y el tercero a las columnas.

En Java es posible crear arreglos de 1, 2, 3, 4,.., n dimensiones, en donde se tiene un arreglo dentro de otro arreglo, proporcionando al programador las herramientas para potenciar sus soluciones.

5.5 Resumen

Gran cantidad de problemas requieren del almacenamiento de datos en arreglos, ya sea arreglos de una sola dimensión también llamados vectores, de dos dimensiones conocidos como matrices o multidimensionales. Los arreglos resultan muy prácticos al ser capaces de almacenar varios datos de un mismo tipo, todos ellos con un mismo nombre, diferenciándose unos de otros por un *índice*, que debe ser una variable o constante de tipo entero. Dependiendo de la dimensión de un arreglo es necesario cierto número de índices.

La creación de un arreglo es un proceso de 2 pasos: la declaración y la creación . Al declarar un arreglo únicamente se está estableciendo una referencia hacia él, mas no se está creando el espacio de memoria para que éste sea capaz de almacenar valores. Una referencia puede entenderse como el nombre a través del cual va a manejarse el arreglo, pero no el espacio de memoria para guardar valores. Para crear el arreglo se utiliza el operador new con la referencia declarada previamente, es entonces cuando se establece el espacio para los valores del arreglo. Los dos pasos para crear un arreglo pueden realizarse en dos líneas de código o fusionar ambas líneas en una sola.

Los arreglos de más de una dimensión pueden ser irregulares, cuando tienen diferente número de columnas en cada renglón o cuando tienen en cada plano una matriz de diferentes dimensiones.

Para introducir valores en las celdas de un arreglo puede hacerse de manera directa con una asignación, a través de una instrucción de lectura o con una inicialización.

En Java todos los arreglos tienen el atributo **`length`**, que proporciona información sobre la dimensión de un arreglo. Según el objeto al que se aplique, `length` indica el número de columnas de un

vector, el número de renglones de un arreglo bidimensional, el número de columnas de cierto renglón de un arreglo bidimensional y el número de plano de un arreglo tridimensional. Esto es muy útil porque los ciclos que accesan a las celdas de los arreglos pueden controlarse en base a `length` y evitar intentos de acceso fuera de los límites del arreglo.

Los arreglos en Java tienen una indexación basada en cero, esto significa que todos los elementos en él comienzan a numerarse desde cero en cualquier dimensión o plano.

Es un error común intentar copiar un arreglo en otro igualando dos referencias. Lo que sucede realmente al igualar una referencia a otra es que ambas referencias estarán manejando un mismo arreglo. Si este es el caso entonces una modificación a través de cualquiera de las referencias se verá reflejada en el mismo arreglo.

Si se desea copiar un arreglo en otro se deben crear los dos arreglos y entonces copiar uno a uno los elementos de los arreglos dentro de un ciclo. Otra forma de copiarlos es a través del método `arraycopy` de la clase `System`.

En Java los arreglos son objetos y todos los objetos al ser usados como argumentos en la invocación a un método, son pasados como referencias en los parámetros, de tal manera que si el arreglo es modificado en el método que lo recibe como parámetro, las modificaciones quedarán permanentes en el arreglo.

5.6 Problemas

1. Escribir un programa que llene un arreglo de 6 renglones por 5 columnas con números enteros aleatorios en el rango de 100 a 900, columna a columna. A continuación imprima el contenido del arreglo renglón por renglón. Utilice el atributo `length`.
2. Escribir un programa que llene una matriz de 7 × 13 con los números adecuados para formar dentro de ella un triángulo de Pascal. Comenzar colocando el número 1 en el renglón 0, columna 6. De ahí en adelante el contenido del resto del arreglo se obtiene sumando los números que estén por encima y a los costados de cada celda. La matriz, una vez con los valores correspondientes quedaría así:

0	0	0	0	0	0	**1**	0	0	0	0	0	0
0	0	0	0	0	**1**	0	**1**	0	0	0	0	0
0	0	0	0	**1**	0	**2**	0	**1**	0	0	0	0
0	0	0	**1**	0	**3**	0	**3**	0	**1**	0	0	0
0	0	**1**	0	**4**	0	**6**	0	**4**	0	**1**	0	0
0	**1**	0	**5**	0	**10**	0	**10**	0	**5**	0	**1**	0
1	0	**6**	0	**15**	0	**20**	0	**15**	0	**6**	0	**1**

3. Agregar el código necesario al programa anterior para imprimir la matriz de manera que solo queden visibles los números que forman el triángulo.

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5  10  10   5   1
    1   6  15  20  15   6   1
```

4. Escribir un programa que considere una matriz de n renglones, donde cada uno de ellos tendrá un número variable de columnas. Ese número está entre uno y diez generado aleatoriamente. Una vez sabiendo el número de columnas, el programa debe generar también aleatoriamente unos y ceros para ser almacenados en las celdas de ese renglón. Después de haber llenado el renglón debe interpretarse la serie de unos y ceros que quedaron en él y calcular su equivalente en base decimal. Dicha equivalencia debe almacenarse en un arreglo unidimensional de n. Mostrar el contenido de ambos arreglos para ver los dígitos que forman el número binario y su correspondiente en base decimal.

```
Cuántas filas tiene el arreglo: 5
Bits generados aleatoriamente para cada fila son:
0   equivale a 0 decimal
1  0  0  1  1  1  0  1  0  0   equivale a 628 decimal
0  0  0  0  1  1  1  0   equivale a 14 decimal
1  0  1  0   equivale a 10 decimal
1  0  0  0  0   equivale a 16 decimal
```

5. Considere que un arreglo de 4 × 7 fue llenado con números aleatorios del 10 al 99. Escribir un programa que imprima el contenido del arreglo y a continuación pregunte un *número de renglón* y un *número de columna* para tomarlos como inicio, para pasar de esa celda a la que está por debajo y a la derecha de la actual, continuar ese patrón de paso *N* veces, imprimiendo el contenido de la celda por la que se va pasando. Organizar el programa en los métodos: generar, imprimir_arreglo e imprimir_recorrido. El arreglo debe declararse en el método main.

```
Los números generados en el arreglo son:
30    52    73    22    39    14    28
29    45    75    27    85    80    43
35    11    75    86    48    97    12
62    90    36    65    58    65    11
```

(continúa en la siguiente página)

```
¿Renglón inicial? 3
¿Columna inicial? 6
¿Cuántas veces hacer el recorrido? 9
Los números por los que se pasó son:
11    30    45    75    65    39    80    12    62
```

(continúa de la página 236

6. Analice el siguiente código y diga qué valores se imprimirían.

```
package resp_cap5;
public class Arreglos {
public static void main(String[] args) {
int arr[][]={{1,9,5,-2,-2,0},
             {1,3,12,76},
             {9,5,7,80,45,34,26,12,45,-1,-9},
             {5,7,2}};
int r,c,uno,dos;
for(r=0; r<arr.length; r++){
   c=0;
   uno=arr[r][c];
   dos=arr[r][c];
   for(c=1; c<arr[r].length; c++){
      if (uno>arr[r][c])
         uno=arr[r][c];
      if (dos<arr[r][c])
         dos=arr[r][c];
   }
System.out.println(uno+ "     "+dos);
  }
 }
}
```

7. ¿Cuál sería el enunciado del problema para el código anterior?

8. Represente con un dibujo el arreglo A con los valores en las celdas correspondientes, considerando la siguiente declaración:

```
char A[][]={{'a','z','r','e','G','Q','q'},
            {'w','i','T','d','H','j','s','L','k','d'},
            {'f','a'},
            {'x','c','v','m'},
            {'a','f','r','h','y','B','o'}};
```

9. Escriba un programa para meter el carácter '*' en una matriz irregular, de manera que se forme un triángulo como el que se muestra (cada renglón es de un tamaño diferente).

*						
*	*					
*	*	*				
*	*	*	*			
*	*	*	*	*		
*	*	*	*	*	*	
*	*	*	*	*	*	*

El número de renglones será de acuerdo al valor indicado por el usuario.

```
¿De cuántos renglones quieres la figura de triángulo? 3
*
* *
* * *
```

10. Escriba las líneas de código necesarias para imprimir una matriz M tal que cada uno de sus renglones tiene una dimensión diferente.

11. Hacer los cambios correspondientes al programa del *Sort de la burbuja* visto en el ejercicio 5.10 de este capítulo, para leer n nombres, ordenarlos en forma alfabética e imprimir la relación de nombres ordenados. Convertir los nombres a mayúsculas antes de ordenar el arreglo.

```
n: 4
Dame los 4 nombres:
Carlos
Pedro
Alicia
Benjamin

Ordenados
ALICIA
BENJAMIN
CARLOS
PEDRO
```

12. Escribir un programa en Java que realice lo siguiente:

a) Que genere una matriz cuadrada $n \times n$ con elementos entre 0 y 12, generados aleatoriamente.
b) Que sume los elementos de la diagonal.

c) Que sume los elementos que están arriba de la diagonal y que son múltiplos de 3.

```
n: 4

Matriz

5    6    0    7
4    8   10    3
9    2    2    8
7    5    9    4

Suma de la diagonal = 19
Suma de múltiplos de 3 arriba de la diagonal = 9
```

13. Escribir un programa en Java que realice lo siguiente:

    a) Que pida un número *n* y lea los *n* datos de consola.
    b) Que encuentre la diferencia entre los números consecutivos y los imprima.
    c) Que encuentre la diferencia mayor y diga entre qué números se encuentra.

```
n: 6
Dame los 6 números:
5
−1
6
10
8
4
Diferencia entre consecutivos
6
7
4
2
4
La diferencia mayor es 7 y está entre el −1 y 6
```

14. Para multiplicar dos matrices ${}_mA_n$ y ${}_iB_j$ se requiere que el número de columnas de la primera matriz sea igual al número de filas de la segunda. Esto significa que n = i. La matriz resultante del producto, tiene el número de filas de la primera matriz y el número de columnas de la segunda.

$$ {}_mA_n \times {}_iB_j = {}_mC_j $$

El primer elemento de la matriz C, resulta de multiplicar la primera fila de la matriz A por la primera columna de la matriz B. El segundo elemento de C, se obtiene al multiplicar la primera fila de la matriz A, por la segunda columna de la matriz B y así sucesivamente.

$$\overset{{}_2A_3}{\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{vmatrix}} + \overset{{}_3B_4}{\begin{vmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \end{vmatrix}} = \overset{{}_2C_3}{\begin{vmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{11} & c_{12} & c_{13} & c_{14} \end{vmatrix}}$$

$$c_{11} = a_{11} \times b_{11} + a_{12} \times b_{21} + a_{13} \times b_{31}$$

Escribir un programa para leer las dimensiones de dos matrices cualquiera A y B. Generar los elementos aleatoriamente con valores entre -9 y 20. Obtener la multiplicación de las dos matrices en la matriz C. Mandar un mensaje en caso en que no sea posible realizar la multiplicación, porque las dimensiones de las matrices no cumplen con la condición requerida.

```
Filas de A: 2
Columnas de A: 3
Matriz A
-1    3    7
 2    4    9

Filas de B: 3
Columnas de B: 3
Matriz B
 6    1   10
-2    3    7
 4    5    0

Matriz C
16   43   11
40   59   48
```

15. Escribir un programa para realizar lo siguiente:
    - Leer las dimensiones de una matriz y generar sus elementos aleatoriamente.
    - Imprimir la transpuesta de la matriz, que consiste en intercambiar las filas por columnas.
    - Si la matriz es cuadrada que mande el mensaje "Matriz cuadrada" y que sume los elementos de la diagonal que sean impares.
    - Si la matriz no es cuadrada que sume los elementos de la periferia de la matriz que sean múltiplos de 7, es decir los elementos de la primera y última filas, además de los elementos de la primera y última columnas que sean múltiplos de 7.

```
Filas: 3
Columnas : 4
Matriz
  3    7   -4   21
  5   10   28    2
-14    8   -2    9

Transpuesta:
  3    5  -14
  7   10    8
 -4   28   -2
 21    2    9

No es cuadrada.
Suma de múltiplos de 7 de periferia = 14
```

```
Filas: 4
Columnas : 4
Matriz
 12    5   -4   21
  5   13   28    2
 -7    6   -3    9
 10   17    0   -1

Transpuesta:
 12    5   -7   10
  5   13    6   17
 -4   28   -3    0
 21    2    9   -1

Es cuadrada.
Suma de múltiplos de 3 de diagonal = 9
```

16. Desde el punto de vista estadístico la moda de un conjunto de datos es aquel elemento que se repite más veces. Escribir un programa para:

    a) Generar aleatoriamente un conjunto de n datos enteros entre ***w*** y ***x***, y guardarlos en un arreglo.
    b) Imprimir los datos generados.
    c) Encontrar la moda del conjunto de datos generados en el inciso (a).

    Considerar que si son dos o más datos los que se repiten más veces, entonces la moda son todos esos elementos que tienen mayor frecuencia.

```
n: 24
w: 10
x: 25
Datos generados entre 10 y 25:
19 12 17 24 12 16 24 14 21 20
13 25 17 21 23 19 14 18 16 12
15 20 12 22

La moda es el 12 y se repite 4 veces
```

17. Escribir un programa para realizar lo siguiente:

    a) Leer un conjunto de n alumnos con x calificaciones cada uno de ellos.
    b) Calcular el promedio de cada uno de los alumnos y el promedio del grupo.
    c) Imprimir la información en forma de tabla, por orden alfabético del nombre del alumno.

```
No. de alumnos: 3
No. de materias por alumno: 3
Nombre: José
Calif 1:78
Calif 2: 81
Calif 3: 90
Nombre: Alicia
Calif 1:71
Calif 2: 88
Calif 3: 83
Nombre: Carlos
Calif 1:70
Calif 2: 56
Calif 3: 82
```

Nombre	Calif 1	Calif 2	Calif 3	Promedio
Alicia	71	88	83	80
Carlos	70	56	82	69
José	78	81	90	83

Promedio del Gpo. = 77

18. Escribir un programa que realice lo siguiente:
    a) Generar aleatoriamente n números enteros entre 0 y 50 que se puedan visualizar en la pantalla y guardarlos en un vector.
    b) Encontrar los elementos mayor y menor del vector.
    c) Indicar la posición (o posiciones) del elemento mayor y del elemento menor.

```
n: 8
Valores generados:
34  46  12  46  31  7  12  28
Mayor = 46 en posición: 2, 4
Menor = 7 en posición: 6
```

19. La desviación estándar es la distancia que hay de los datos con respecto a su media. La media aritmética es el promedio de los datos. Sus fórmulas son las siguientes:

$$\bar{x} = \frac{\sum_{i=i}^{n} xi}{n} \qquad S = \sqrt{\frac{\sum_{i=1}^{n} (xi - \bar{x})^2}{n-1}}$$

Donde:

n: Número de observaciones o datos.
$\bar{x}$: Media aritmética de los datos.
S: Desviación estándar.
xi= Dato i_ésimo.

Escribir un programa para:

a) Generar aleatoriamente ***n*** datos enteros entre ***w*** y ***x***, y guardarlos en un arreglo.
b) Imprimir los datos generados.
c) Encontrar la media y la desviación estándar de ese conjunto de datos.

```
n: 7
w: 33
x: 37
Datos  generados entre 33 y 37:
36   34   34   35   34   36   34
Obs          Datos        (xi-xmed)^2
1            36           1.65306122
2            34           0.51020408
3            34           0.51020408
4            35           0.08163265
5            34           0.51020408
6            36           1.65306122
7            34           0.51020408
suma=        243          5.42857143

xmed= 34.714
S= 0.9512
```

20. Para sumar dos cantidades en cualquier sistema numérico, se suman de la misma manera que como se hace en el sistema decimal, la única diferencia es la base del sistema numérico. La base del sistema decimal es 10, en sistema binario es 2, en octal es 8, en el sistema trinario es 3 y así sucesivamente. A continuación se tiene la suma de dos cantidades en base 2 y base 14:

$$\begin{array}{rcccccccc} & & 1 & 0 & 1 & 0 & 1 & 0_{(2)} \\ + & 1 & 1 & 0 & 1 & 1 & 1 & 1_{(2)} \\ \hline 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1_{(2)} \end{array} \qquad \begin{array}{rccccc} & 8 & 9 & D & A & 1_{(14)} \\ + & & C & 7 & 6 & B_{(14)} \\ \hline & 9 & 8 & 7 & 2 & C_{(14)} \end{array}$$

En base 14 los dígitos válidos son: 0,1,2,…,9 y las letras A,B,C y D. De la misma manera que en el sistema hexadecimal, las letras tienen un valor numérico como se muestra: A=10, B=11, C=12 y D=13. De tal manera que:

$1 + B = 1+11= 12$ — Como 12 es la letra C en base 14, se coloca debajo de la línea

$A + 6 = 10 + 6 = 16$ — Como el 16 no se vale en base 14. Se divide entre la base (14), se coloca el residuo de la división debajo de la línea y el cociente se le suma a la columna de la izquierda.

$$\begin{array}{r} 1 \\ 14\overline{)16} \\ 2 \end{array}$$

$1 + D + 7 = 1 + 13 + 7 = 21$ — El 21 no es válido en base 14, por lo tanto se divide entre la base, el residuo de la división se coloca debajo de la línea y el cociente se le suma a la columna de la izquierda.

$$\begin{array}{r} 1 \\ 14\overline{)21} \\ 7 \end{array}$$

Y así sucesivamente hasta terminar de realizar la suma.

Escribir un programa para sumar dos cantidades en cualquier sistema numérico comprendido entre la base 2 y la base 16:

```
Base: 5
Sumando A:  301432
Sumando B:   41031
0301432
0041031
0343013
```

```
Base: 16
Sumando A:  C75A04
Sumando B:  F3897D
0C75A04
0F3897D
1BAE381
```

21. Estructurando la solución por medio de métodos. Elaborar un programa para crear una matriz utilizando el parámetro N para indicar el número de renglones que deberá tener, donde el primer renglón debe tener también N columnas y el número de renglones siguientes debe ir incrementándose en 2. Almacenar en la matriz valores enteros aleatorios, de manera tal que en los renglones pares (0,2,

4,..,etc.) queden valores pares y en los renglones impares queden valores impares. Los valores aleatorios generados deben estar en el rango de 50 a 99. Visualizar los valores almacenados en la matriz. A continuación de muestra cómo se vería el método main:

```
public static void main(String[] args) {

/* Comienza la ejecución en main, llamando al método meter_nums_
en_matriz */

//Introducir datos en matriz de 4 renglones
meter_nums_en_matriz(4);
}
```

22. Usando métodos, hacer un programa que almacene los valores de la serie Ulam de n números en una matriz, de forma que cada renglón de la matriz tenga el número de columnas necesarias para almacenar los valores de cada serie. Los valores de los que se desea conocer su serie Ulam deben ser ingresados por el usuario y quedar almacenados en otro arreglo unidimensional.

```
run:
Para cuantos números deseas obtener su serie de Ulam :3
Ingresa  valor (mayor que cero): 6
Ingresa  valor (mayor que cero): 2
Ingresa  valor (mayor que cero): 7

La serie Ulam de 6 es
 6 3 10 5 16 8 4 2 1
La serie Ulam de 2 es
 2 1
La serie Ulam de 7 es
 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
GENERACIÓN CORRECTA (total time: 12 seconds)
```

23. Hacer un programa que declare e inicialice de manera directa un arreglo de caracteres, siendo cada elemento del arreglo un caracter numérico ('3','0','1',etc.). En el programa deje en una variable tipo `String` los caracteres en el arreglo. A continuación convierta la cadena en un entero y finalmente muestre el resultado de la cadena convertida a número multiplicada por dos.

```
run:
La cadena es  153862
Convertido a número y multiplicado por 2 es  307724
La cadena es  401
Convertido a número y multiplicado por 2 es  802
GENERACIÓN CORRECTA (total time: 1 second)
```

INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTADA A OBJETOS

6

La función de un buen software es hacer que lo complejo aparente ser simple.

Grady Booch

Competencia de la unidad

Construir programas sencillos bajo el paradigma de la programación orientada a objetos.

- Conocer los conceptos centrales de la programación orientada a objetos.
- Aplicar los conceptos centrales de la programación orientada a objetos en el diseño y construcción de programas.

Control de flujo

Contenido

6.1. Introducción.
6.2. Orígenes de la programación orientada a objetos.
6.3. Beneficios de la programación orientada a objetos.
6.4. Clases.
6.5. Métodos.
6.6. Objetos.
6.7. Sobrecarga de métodos.
6.8. Herencia.
6.9. Otros conceptos del paradigma de programación orientada a objetos.
6.10. Resumen.
6.11. Problemas.

6.1 Introducción

Un paradigma de programación es una forma específica de dar solución a un problema. Es decir, cuando se tiene un problema pueden existir varias alternativas de solución y cada una de ellas representa un paradigma diferente.

Actualmente, existen diversos paradigmas de programación como el imperativo, el declarativo, el funcional, el orientado a aspectos, el orientado a componentes, entre otros; pero, el paradigma de Programación Orientada a Objetos (POO) es uno de los más utilizados, debido a los beneficios que trae consigo; uno de ellos es que el diseño y la construcción de programas son muy similares a la forma en que los humanos entienden el entorno. Por ejemplo, si se observa alrededor se pueden identificar objetos (un lápiz, una televisión, un teléfono, una manzana, etc.) y de acuerdo a esto es posible saber lo que cada objeto es capaz de hacer o lo que se puede hacer con esos objetos (el lápiz puede escribir, la televisión puede transmitir un programa, el teléfono puede enviar mensajes, la manzana puede comerse, etc.); se sabe lo que cada objeto es capaz de hacer, porque se conoce (consciente o inconscientemente) la clase a la que pertenece. Así, como la manzana pertenece a la clase "Alimentos Frutales" entonces puede comerse; nadie con sentido común pretendería comerse un teléfono, una televisión o un lápiz, pues esos objetos pertenecen a clases diferentes con características que los hacen no aptos para comerse.

Cada clase tiene ciertas características (atributos) y acciones (métodos) propias de ella. La clase *teléfono_celular* tiene, entre otras, las características de color, marca, precio, saldo, números frecuentes y las acciones de enviar mensajes, recibir mensajes, recibir llamadas, hacer llamadas, abonar saldo, por mencionar algunas. Cualquier objeto que pertenezca a la clase teléfonos celulares tendrá las características y podrá realizar las acciones que se acaban de mencionar (véase Figura 6.1).

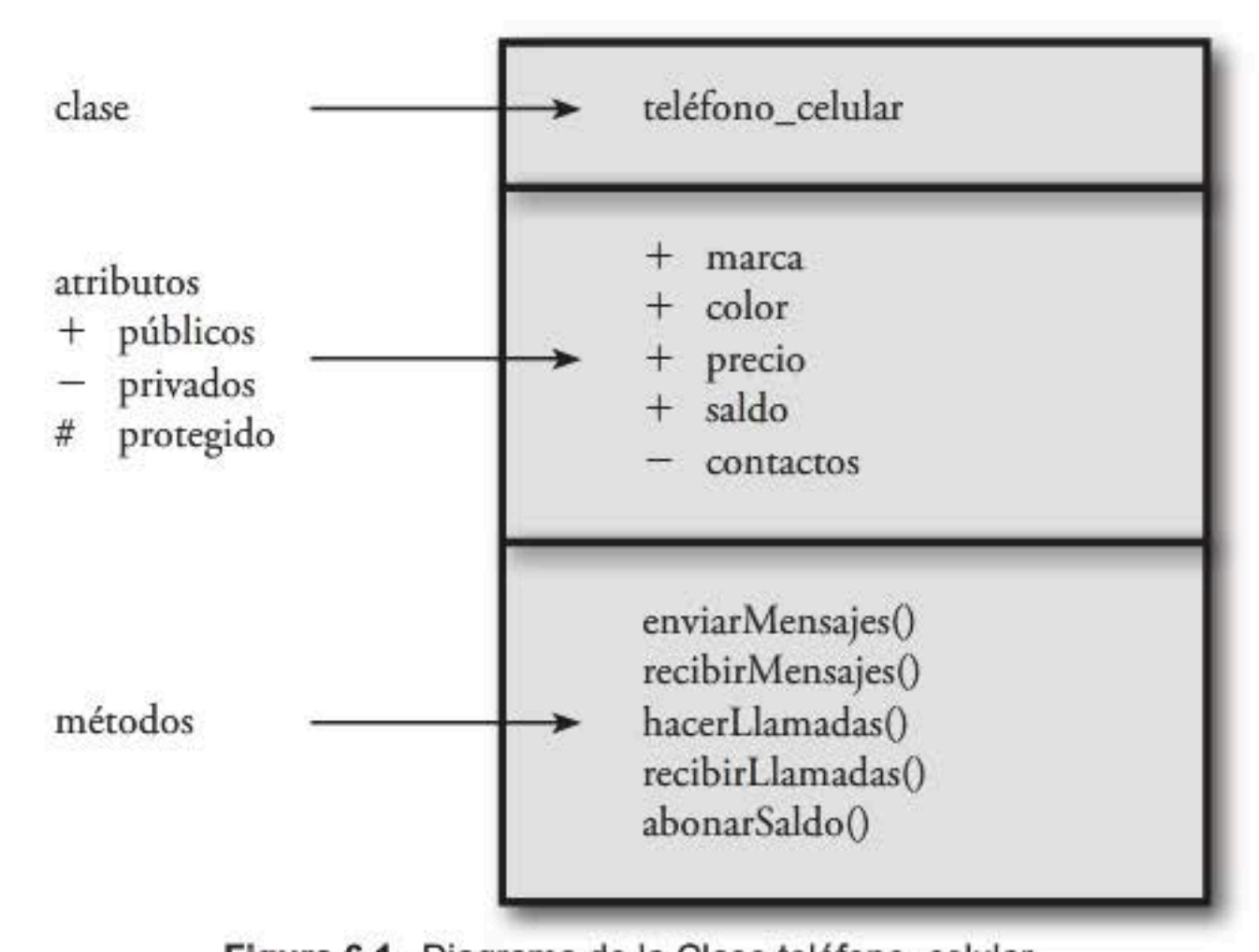


Figura 6.1. Diagrama de la Clase teléfono_celular.

Ahora, pensando en un objeto en particular de esa clase se puede tener un teléfono rojo de marca nokia, con un precio de $800, un saldo de $100 y ciertos números frecuentes, capaz de enviar y recibir mensajes, hacer y recibir llamadas y abonar saldo. Otro objeto de esa misma clase sería un teléfono azul, de

marca motorola, con un precio de $500, un saldo de $100 y ciertos números frecuentes, capaz de enviar y recibir mensajes, hacer y recibir llamadas y abonar saldo.

Cada que se realiza una acción (método) con un objeto, sus características (atributos) se pueden ver alteradas. Si se hacen varias llamadas, el atributo *saldo* disminuirá; si se abona al *saldo*, éste se incrementará. Existen atributos que pueden ser modificados por cualquier otro objeto, pero hay atributos que no podrían modificarse tan fácilmente. En el caso del teléfono se pueden modificar los números registrados como frecuentes si el objeto *usuario* así lo indica, pero la marca no podrá ser cambiada. Los atributos que están expuestos a cambios o modificaciones se consideran públicos; los que están protegidos contra alteraciones externas se consideran privados.

Todos los objetos pertenecen a alguna clase, tienen ciertas características; algunas de ellas susceptibles de ser modificadas (públicas) y otras no (privadas), y son capaces de realizar ciertas acciones. Las clases a su vez pueden pertenecer o ser subclases de otras clases, por ejemplo la clase *hombre* es una subclase de la clase *humano*, la clase *mujer* es también una subclase de la clase humano. *'Rebeca'* y '*Ana*' son objetos de la clase *mujer*, *'Diego'* es un objeto de la clase *hombre* (véase Figura 6.2).

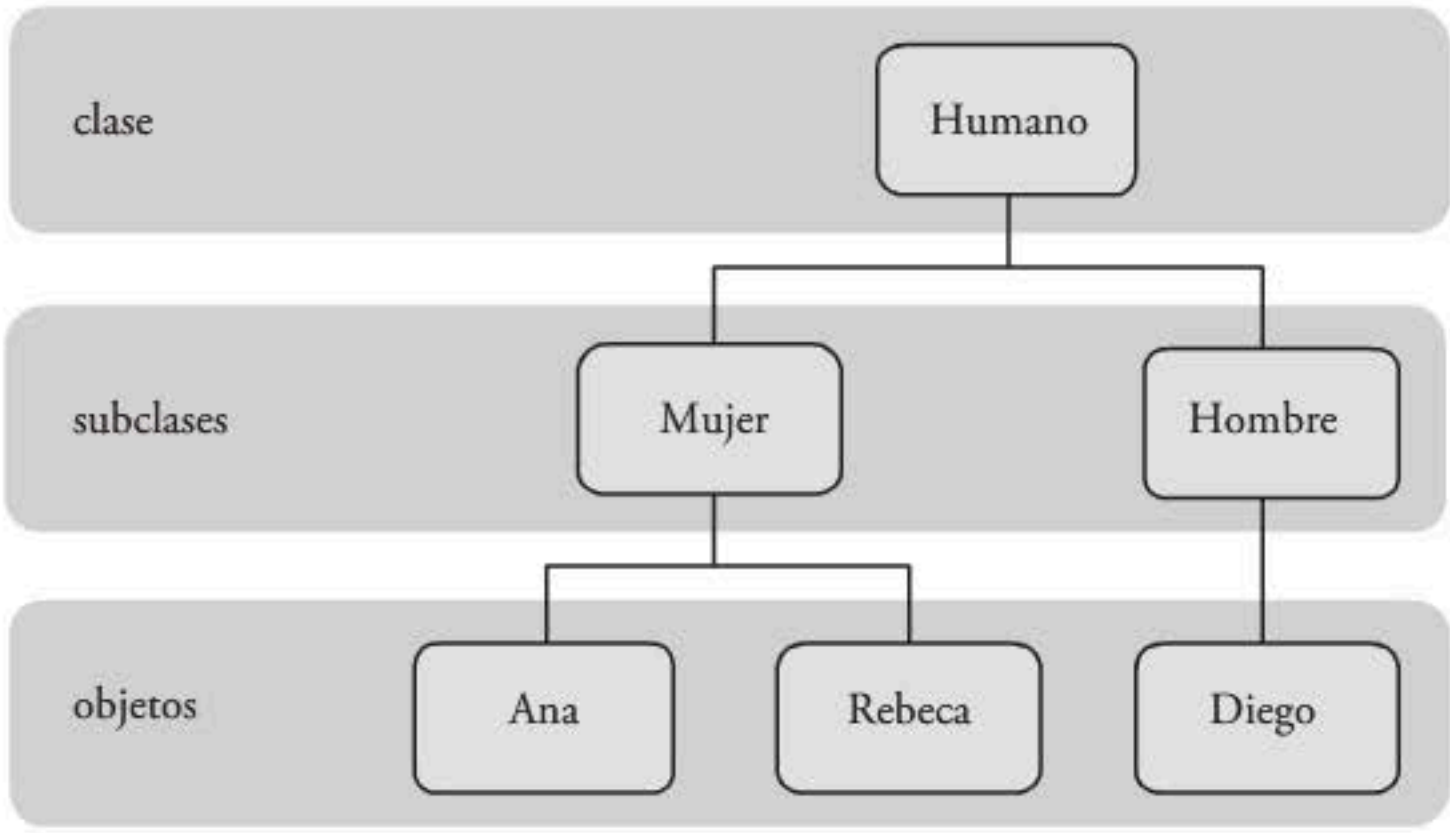


Figura 6.2. Ejemplo de clases, subclases y objetos.

6.2 Orígenes de la programación orientada a objetos

La Programación Orientada a Objetos surgió en el año de 1967 en el Centro Noruego de Computación, cuando Kristen Nygaard (Premio Turing 2001) y Ole Johan Dahl (Premio Turing 2001) diseñaron el lenguaje de programación Simula 67, el cual permitía hacer simulaciones de naves. Durante las simulaciones, los investigadores se percataron del problema que existía al no contar con una adecuada identificación de las naves, porque eran tantas las combinaciones que el número de simulaciones crecía de manera considerable; para dar solución a este problema nació la idea de agrupar las naves en diferentes clases de objetos para que cada uno tuviera sus propios datos y comportamientos.

En 1980, se dio a conocer Smalltalk, un lenguaje diseñado por Alan Kay (Premio Turing 2003), en el Centro de Investigación en Palo Alto de Xerox. Este lenguaje fue creado con fines educativos, para que a través de ese "mundo virtual" fuera posible interactuar con "objetos virtuales" los cuales se comunicaban

entre sí por medio de mensajes. De esta manera, se definieron por primera vez los conceptos del paradigma de la POO: clase, objeto, atributo, método, etc. Durante la década de los ochentas del siglo pasado, la POO cobró mucha popularidad entre los programadores y fue así que en 1983 Bjarne Stroustrup diseñó un nuevo lenguaje orientado a objetos llamado C++, el cual conservaba la sintaxis del exitoso lenguaje C y adquiría las ventajas del nuevo paradigma de programación.

Finalmente, en la década de los noventas nació Java, otro lenguaje con alto soporte para el paradigma de la Programación Orientada a Objetos. Este ha sido el lenguaje con más adeptos al paradigma de la POO, en gran medida por la relación que tiene con internet, pues cualquier navegador tiene un intérprete de la Máquina Virtual de Java (JVM por sus siglas en inglés, *Java Virtual Machine*).

En la siguiente figura se muestran los aspectos más significativos de la historia de la POO:

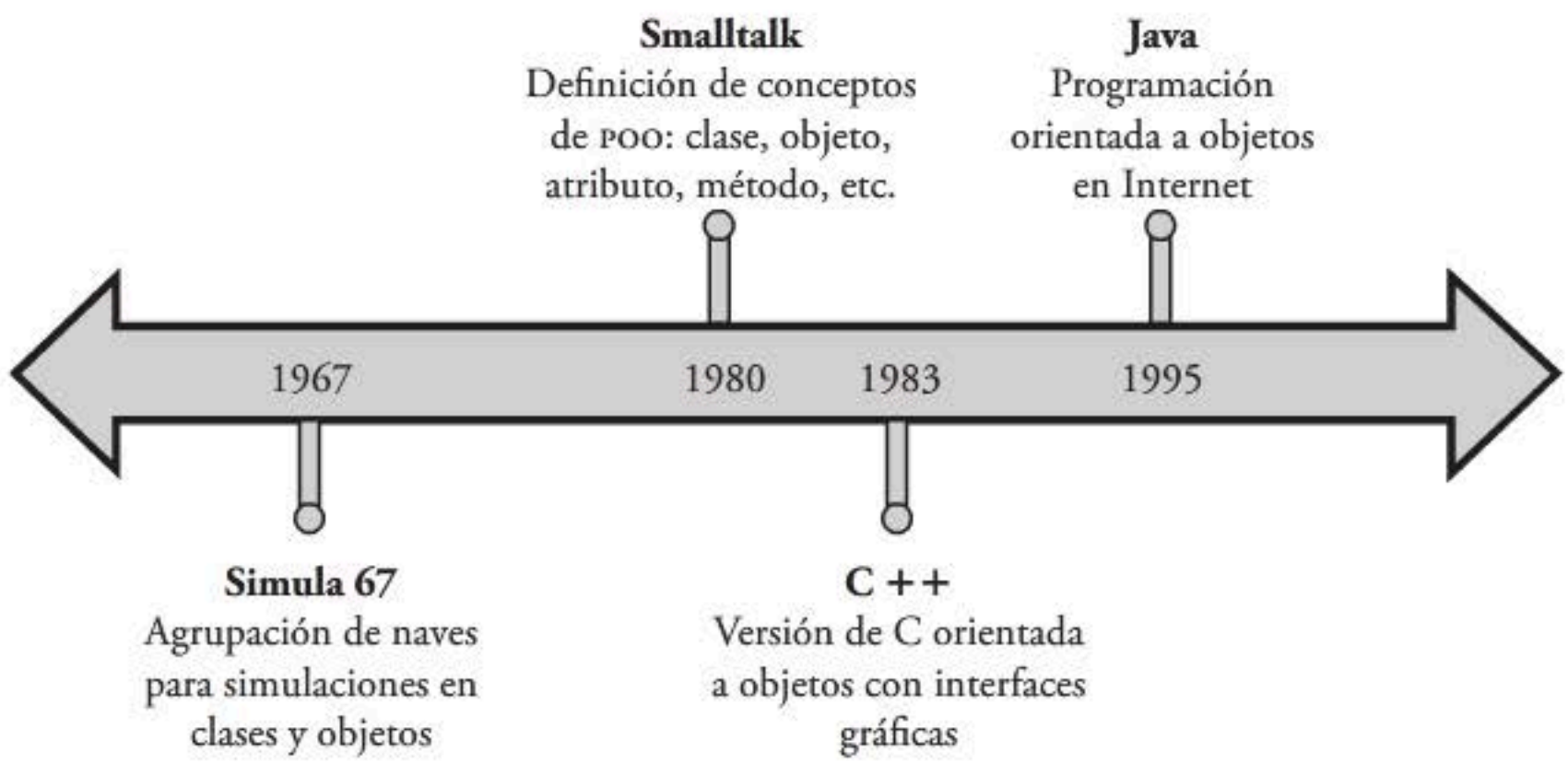


Figura 6.3. Historia de la POO.

6.3 Beneficios de la Programación Orientada a Objetos

Entre los beneficios que ofrece el paradigma de la Programación Orientada a Objetos están:

Modelación Comprensible: La forma de modelar una solución mediante el paradigma de la POO es muy similar a la visión que tienen los seres humanos respecto a la realidad.

Reutilización: La posibilidad de no comenzar nuevas aplicaciones desde cero, pues el diseño que da la POO al código permite usar nuevamente código creado con anterioridad.

Flexibilidad: El paradigma de la POO crea código flexible y facilita la adaptación de código ya escrito a nuevas necesidades sin mucho esfuerzo. Esto repercute en un mantenimiento más sencillo y menos costoso de las aplicaciones.

Escalabilidad: Las aplicaciones pueden ir creciendo a la par de las nuevas necesidades debido a la reutilización y características como la herencia.

6.4 Clases

Una clase es un tipo de dato que contiene en una misma estructura atributos (variables) y métodos (funciones).

La sintaxis básica para definir una clase en Java es la siguiente:

```
especificador_acceso class nombre_clase {

      //atributos

      //métodos
}
```

El primer elemento, el *especificador de acceso*, indica la visibilidad de la clase (quiénes pueden verla), existen dos tipos:

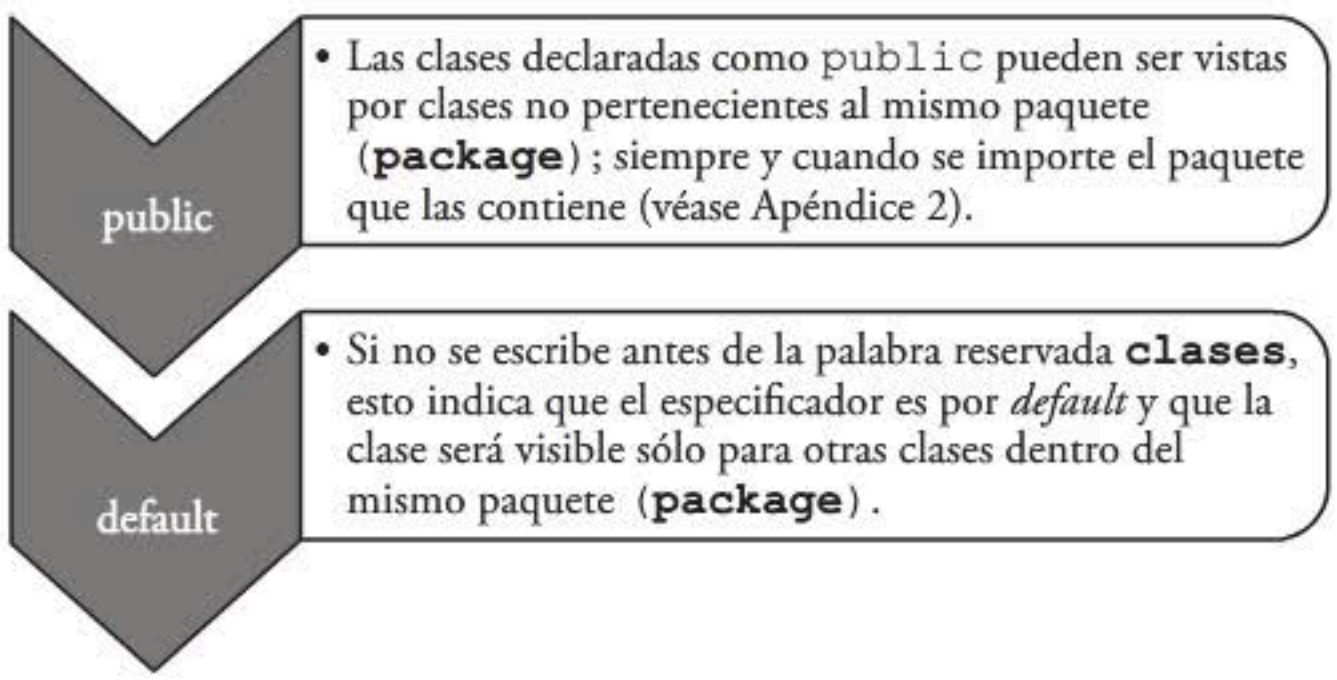


Figura 6.4. Especificadores de acceso o visibilidad de una clase.

El segundo elemento es escribir la palabra reservada **class** y después el nombre de la clase; a continuación se escriben entre llaves los atributos y los métodos de dicha clase.

Los atributos de una clase son variables que almacenan la información de cada objeto y también pueden hacer uso de ciertos especificadores de acceso (véase Figura 6.5), para lo cual se utiliza la siguiente sintaxis:

```
especificador_acceso tipo nombre_atributo;
```

Es decir, se hace la declaración tal como se vio en la sección 3.7.4 de este libro pero anteponiendo el tipo de especificador de acceso. Algunos ejemplos son:

```
/* se define atributo public de tipo int llamado pares, iniciali-
zado en 2 */
public int pares = 2;

/* se define un atributo default de tipo double llamado impares */
double impares;
```

```
/* se define un atributo protected de tipo char llamado letra con
un valor de 'f'  */
protected char letra ='f';

/* se define un atributo private de tipo String llamado nombre */
private String  nombre;
```

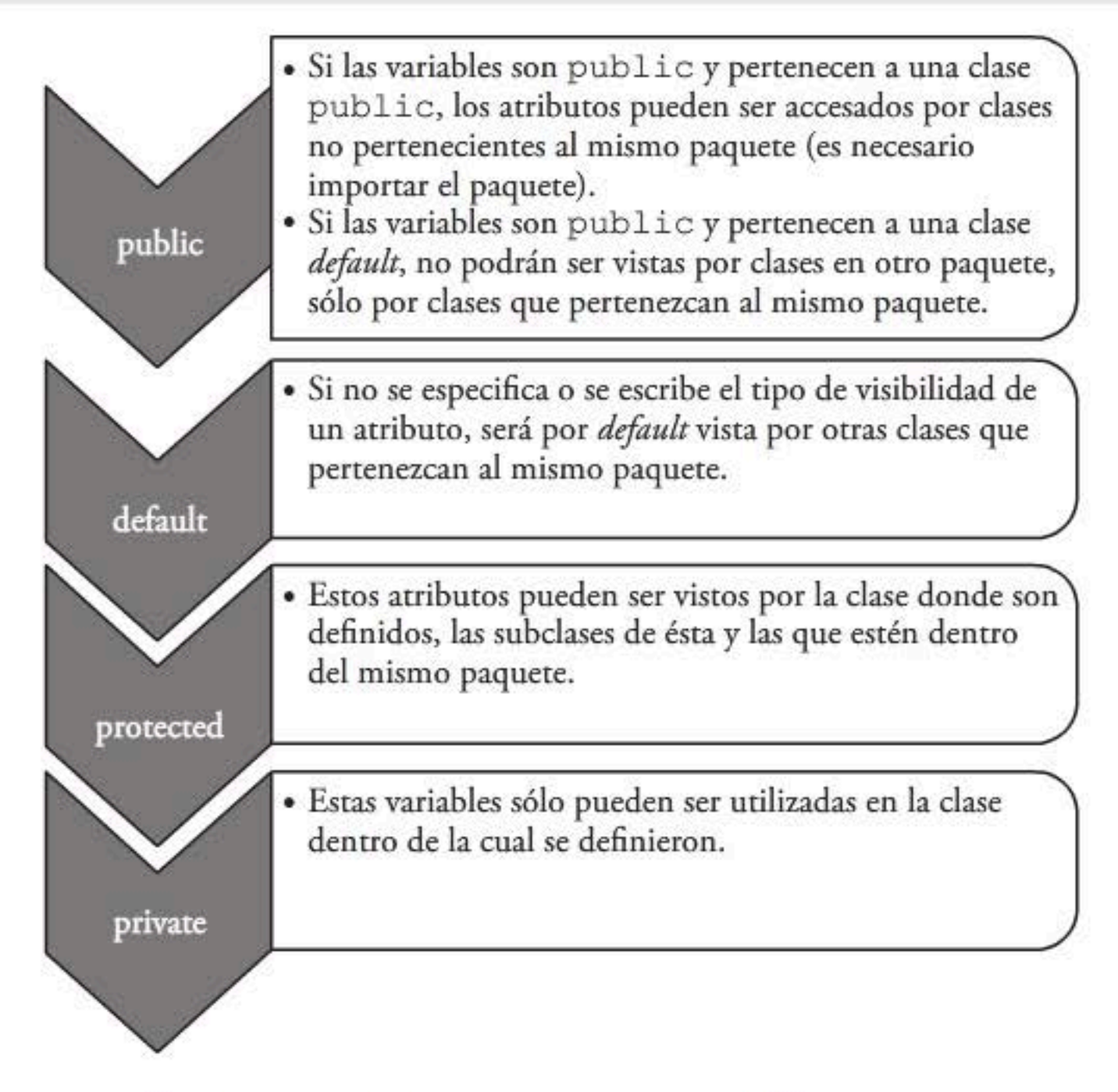


Figura 6.5. Especificadores de acceso o visibilidad de un atributo.

En la siguiente tabla se muestran algunos ejemplos de lo descrito en esta sección:

Tabla 6.1. Ejemplos de uso de especificadores de acceso en clases y atributos.

Ejemplo	Descripción

`package control_escolar;`
 	
`class Alumno {`
`  /*atributos miembros de la clase Alumno*/`
	Se define una clase llamada `Alumno`, la cual tiene un especificador de acceso de tipo *default* (porque no se escribió nada antes de la palabra reservada **`class`**); de manera que sólo podrá ser vista o utilizada por otras clases que pertenezcan al mismo paquete (**`package`** `control_escolar`).

(*Continúa*)

Tabla 6.1. Ejemplos de uso de especificadores de acceso en clases y atributos. (*Continuación*)

Ejemplo	Descripción

`public String nombre;`
`public String apellidos;`
`int edad;`
`protected double promedio;`
`private int calif_1;`
`private int calif_2;`
`private int calif_3;`
 	
`/*aquí se listarían los mé-`
`Todos*/`
 	
`}`
	También, se hizo uso de especificadores de acceso para los atributos que pertenecen a esa clase (atributos miembros): los atributos `nombre` y `apellidos` son **`public`** y al pertenecer a una clase de tipo *default* sólo pueden ser utilizadas por otras clases del paquete `control_escolar` (cuando se tienen atributos **`public`** dentro de clases *default*, los atributos funcionan como si fueran *default*), el atributo `edad` al ser *default* puede ser visualizado sólo por las clases que pertenecen al paquete `control_escolar`, el atributo `promedio` al ser **`protected`** puede ser manejado por las clases que hereden de la clase `Alumno` (sus clases hijas) y los atributos `calif_1`, `calif_2` y `calif_3` al ser **`private`** pueden ser usados sólo por la clase `Alumno`.

`package control_escolar;`
 	
`public class Alumno {`
`/*atributos miembros de la`
`clase Alumno*/`
 	
`public String nombre;`
`public String apellidos;`
`int edad;`
`protected double promedio;`
`private int calif_1;`
`private int calif_2;`
`private int calif_3;`
`/*aquí se listarían los mé-`
`todos*/`
 	
`}`
	Se define la clase llamada `Alumno`, la cual tiene un especificador de acceso **`public`**; por lo que podrá ser vista por clases que estén en el mismo paquete (**`package`** `control_escolar`) y por clases que no pertenezcan a él, siempre y cuando se importe dicho paquete (`control_escolar`).
	
	Además, se emplearon especificadores de acceso para los atributos: `nombre` y `apellidos` son **`public`** y al pertenecer a una clase de tipo **`public`** pueden ser utilizadas por clases dentro del paquete `control_escolar` y por otras que estén en un paquete diferente (es necesario importar), el atributo `edad` al ser *default* sólo puede ser visualizado por las clases que pertenecen al paquete `control_escolar`, el atributo `promedio` al ser **`protected`** puede ser manejado por las clases que hereden de la clase `Alumno` (sus clases hijas) y los atributos `calif_1`, `calif_2` y `calif_3` al ser **`private`** pueden ser usados sólo por la clase `Alumno`.

En los ejemplos de la Tabla 6.1 se creó una abstracción del mundo real, esto a través del uso de la POO, porque es claro ver que si se tiene una clase llamada `Alumno` con las características o atributos de `nombre`, `apellidos`, `edad`, `promedio`, `calif_1`, `calif_2` y `calif_3`, es posible obtener muchos objetos de esta clase, como por ejemplo un objeto cuyo `nombre` sea "*Alejandro*", con `apellidos` "*Martínez García*", de una `edad` de *18* años, con un `promedio` de *86.66* obtenido a partir de sus tres calificaciones `calif_1` de *95*, `calif_2` de *80* y `calif_3` de *85*; o bien, otro objeto de la clase `Alumno` con `nombre` de "*Paola*", `apellidos` "*Hernández Lara*", de una `edad` de *20* años, con `promedio` de *93.33* y calificaciones `calif_1` de *90*, `calif_2` de *100* y `calif_3` de *90*, todo esto tal como se muestra en la siguiente figura:

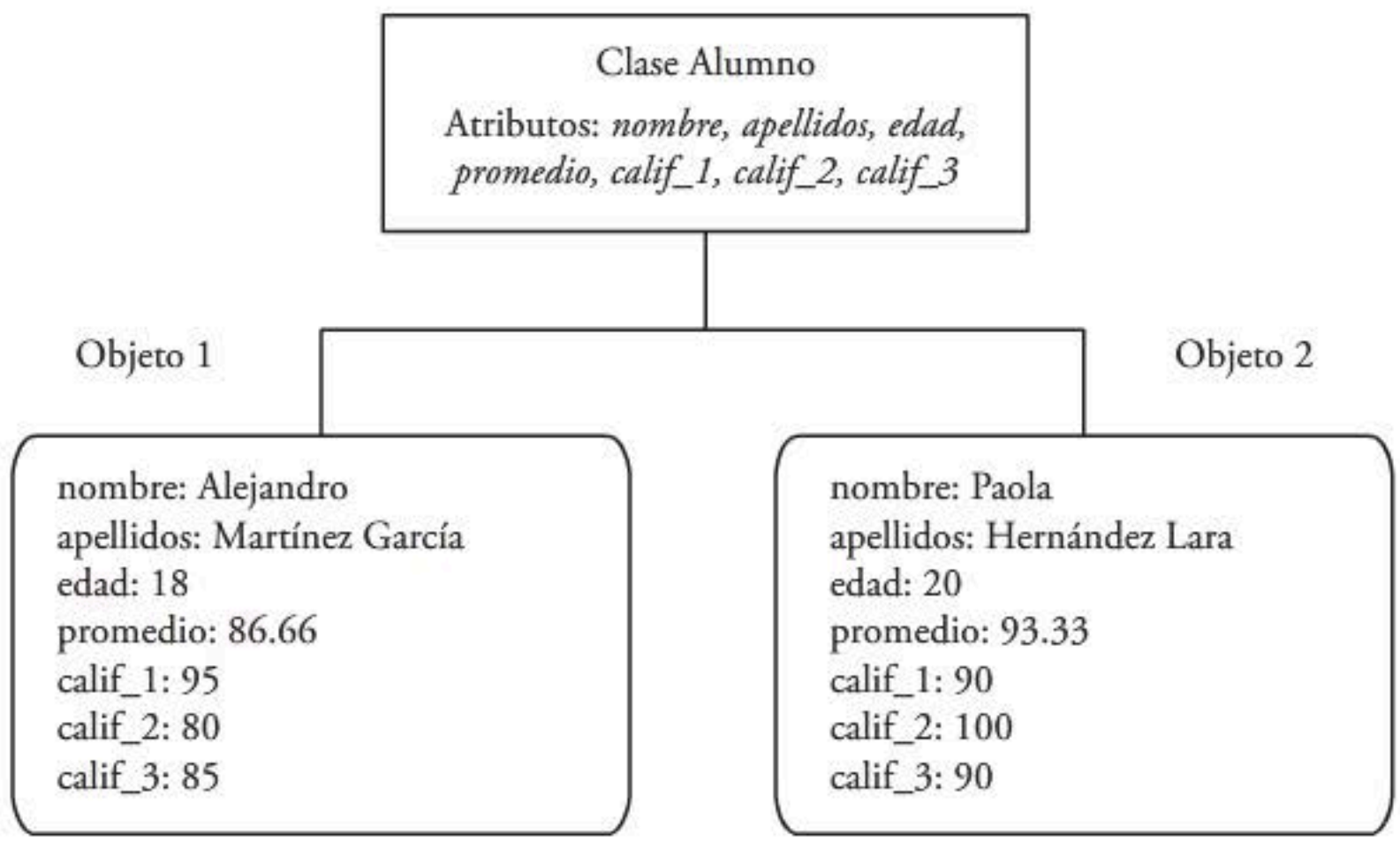


Figura 6.6. Objetos de la Clase Alumno.

6.5 Métodos

Los métodos son funciones de una clase, su sintaxis básica es la siguiente:

```
especificador _acceso [static] tipo_retorno nombre_método (pará-
metros) {
      //sentencias
}
```

El especificador de acceso corresponde a alguno de los cuatro tipos mencionados en la Figura 6.5; es decir, **`public`**, **`default`**, **`protected`** y **`private`** los cuales funcionan igual para los atributos y para los métodos.

La palabra **`static`** es opcional y se emplea para definir un método que puede llamarse desde otra clase sin necesidad de hacerlo a través de un objeto.

Después se define el tipo de valor que regresa el método: **void** (no regresa nada), **int**, **double**, **float**, etc., para en seguida, colocar el nombre del método y entre paréntesis los parámetros que requiere. Finalmente se encierran entre llaves las sentencias o acciones que realizará cierto método.

Ejemplo 6.1. Considerar el código del siguiente programa:

```
package matemáticas;
public class Aritmética {
    public double a;
    public double b;
    public double multiplicacion;
    public double division;
    public static double suma;

    //constructor de la clase
    public Aritmética(String mensaje, double x, double y){
        System.out.println(mensaje);
        a = x;
        b = y;
        multiplicacion=0;
        division=0;
        suma=0;
    }

public void multiplicar( ){
 multiplicacion = a*b;
System.out.println("Multiplicación="+multiplicacion);
}

    void dividir ( ){
        division = a/b;
        System.out.println("División="+division);
    }

    public static double sumar(double n, double m){
        suma = n + m;
        return suma;
    }
}
```

El programa anterior está integrado por una clase llamada `Aritmética`, la cual tiene los siguientes métodos:

- `Aritmética`, que es **public** y se llama igual que la clase, este método recibe el nombre de **constructor** y en este caso se emplea para inicializar los atributos (`a, b, multiplicacion`, `division` y `suma`) y para imprimir un mensaje. De manera general, un **constructor** es

un método que debe cumplir con cuatro características: 1) tener el mismo nombre de la clase, 2) no regresar valores y no requiere la palabra **`void`** para indicarlo, 3) sólo se ejecuta cuando se crea un objeto de la clase a la que pertenece (por ejemplo, si se crean 5 objetos de una clase se llamará 5 veces al constructor) y 4) su función principal es la de inicializar los atributos de los objetos. Cuando no se define un constructor para una clase, Java crea uno automáticamente. Dentro del constructor, si no hay sentencias que indiquen algo diferente, las variables numéricas se inicializan en cero, las boolenas en true y las referencias en null.

En algunos lenguajes que soportan la Programación Orientada a Objetos existe un método definido como destructor, que es la contraparte del constructor. Su objetivo es realizar ciertas acciones antes de que un objeto sea destruido. Java no cuenta con un método destructor propiamente dicho, sino que tiene un mecanismo de recolección de basura conocido como GC (*Garbage Collector*, Recolector de Basura), que optimiza el uso de la memoria quitando de ésta los objetos que ya no serán utilizados.

- `multiplicar`, que es **`public`** por lo que puede ser visto por clases que no estén dentro del paquete `matemáticas` (siempre y cuando se importe), éste método utiliza los valores de los atributos `a` y `b` que fueron inicializados en el constructor para poder calcular la `multiplicación` de los mismos e imprimirla. Se califica como **void** porque no se devolverá ningún valor dentro del método (no incluye la sentencia **return** en su código).
- `dividir` que tiene un especificador de acceso *default* (es decir, se omite el especificador), así que puede ser visto sólo por las clases que estén dentro del paquete `matemáticas`, este método requiere los valores de tipo **`double`** `a` y `b` inicializados en el constructor para calcular e imprimir la `division` de los dos números. Se califica como **void** porque no se devolverá ningún valor dentro del método (no incluye la sentencia **return** en su código).
- `sumar` es un método **`static`**, por lo que puede ser llamado sin necesidad de instanciar la clase `Aritmética` (sin crear un objeto), éste método recibe dos valores **`double`** `n` y `m` para calcular y *devolver* su `suma`. Un detalle importante a notar es que si se requieren variables dentro de un método **`static`**, éstas deben ser también **`static`** (en este ejemplo `suma`) para que no haya problemas con el compilador de Java. Se califica como **double** porque ahora si hay un valor que se devuelve y éste es de tipo double (atributo suma). La sentencia ***return*** *suma;* da la pauta de que el método ya no es de tipo void.

De manera general, se puede decir que existe una relación muy estrecha entre los atributos y los métodos de una clase; la ejecución de los métodos puede cambiar el valor de los atributos. Por ejemplo, si se piensa en la clase `Casa` (véase Figura 6.7), ésta tiene los atributos: `ubicación`, `propietario`, `color`, `metros_de_construcción`, entre otros. También contiene los métodos `pintar`, `remodelar`, `albergar`, etc. Y al existir relación entre atributos y métodos, el método `pintar` puede cambiar el valor del atributo `color`, el método `remodelar` puede cambiar el número de `metros_de_construcción`.

De ésta manera, podría tenerse un objeto perteneciente a la clase `Casa` que esté ubicada en la calle *"Veracruz número 14 de la colonia Molino de Parras"*, cuyo propietario es *"Esteban González"*, de color "*amarillo*", con *90 mts cuadrados* de construcción, capaz de ser remodelado, ser pintado o albergar personas. Otro objeto casa podría estar ubicado en la calle "*Abasolo número 156 de la colonia Ventura Puente*", cuyo propietario es "*Eva Ramírez*", de color "*café*", con *125 mts cuadrados* de construcción, capaz de ser remodelado, ser pintado o albergar personas. Como puede apreciarse todos los objetos de una misma clase tendrán los mismos atributos y métodos, con la diferencia de que los valores de cada atributo son diferentes: los dos objetos tienen un `color`, sólo que en un objeto el color es "*amarillo*" y en el otro es

"*café*". Los dos tienen un `propietario`, pero el propietario del primer objeto es "*Esteban González*" y en el otro es "*Eva Ramírez*".

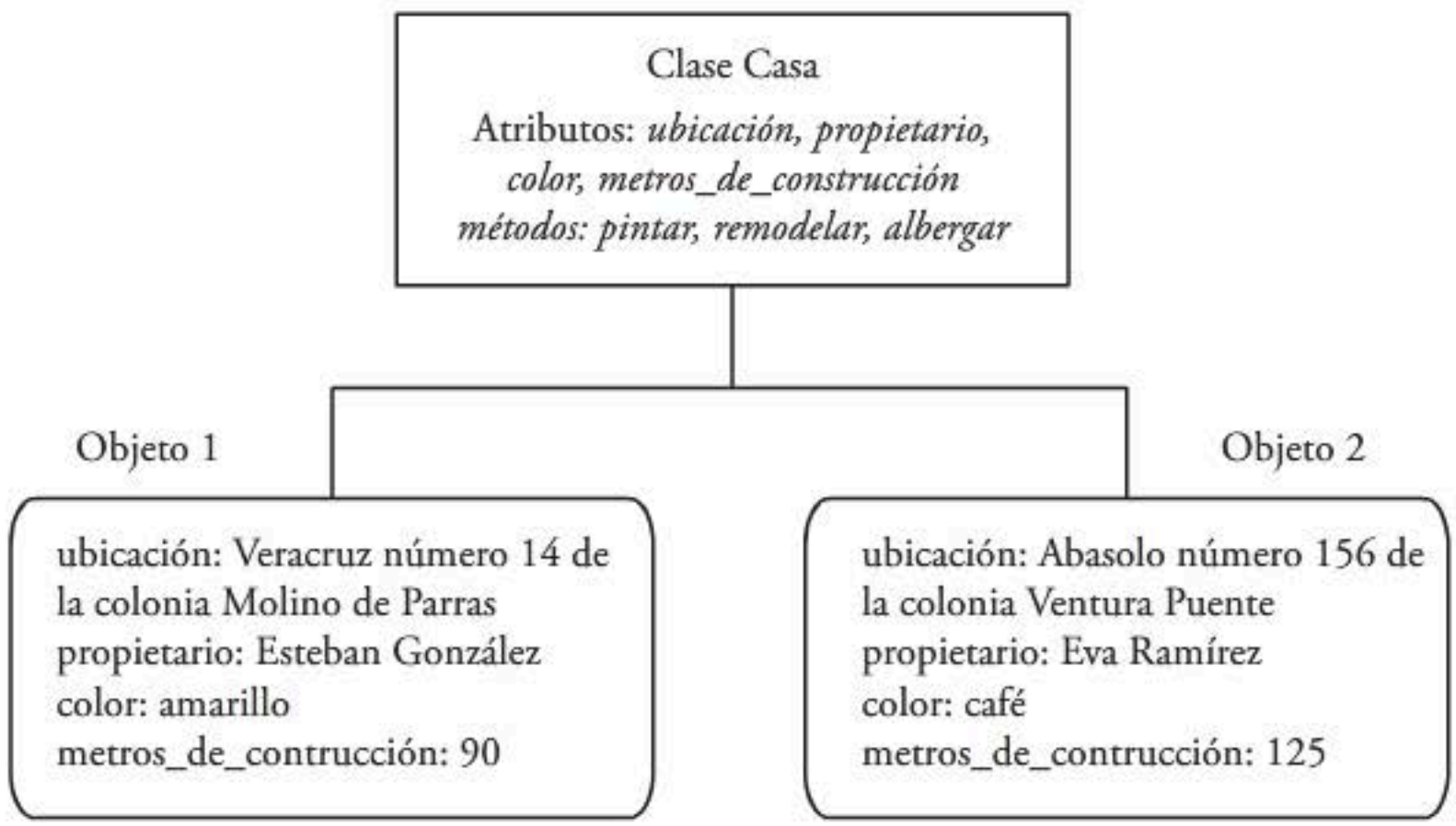


Figura 6.7. Objetos de la Clase Casa.

6.6 Objetos

Un objeto es una instancia de una clase, de la misma manera que una variable es una instancia de un tipo de dato (considerar que en la sentencia **`int`** `edad`, la variable `edad` es una instancia del tipo de dato **`int`**); por lo que, se puede ver una clase como el tipo de dato de un objeto.

Para acceder a los atributos y/o métodos de una clase es necesario crear un objeto, la sintaxis para esto es:

```
nombre_clase nombre_objeto = new nombre_clase ( );
```

Cuando se crea un objeto de una clase, se invoca al constructor de dicha clase, que es un método que se llama igual que la clase y que no devuelve ningún tipo de dato pero sí puede recibir parámetros. El constructor de una clase se ejecuta cada vez que se crea un nuevo objeto.

Para que funcione el código de un método de una clase desde un objeto de esa clase, es necesario utilizar la variable que contiene al objeto (`nombre_objeto`) seguido del operador punto (.) y el nombre del método; de manera que se deben seguir las siguientes reglas en la sintaxis:

```
/*Cuando se llama a un método pero éste no recibe ni regresa va-
lores */
nombre_objeto.nombre_metodo( );

/*Cuando se llama a un método que sí recibe un valor, pero no de-
volverá ningún valor */
nombre_objeto.nombre_metodo(valor);
```

```
/*Cuando se llama a un método que recibe más de un valor, pero no
devolverá ningún valor. Los valores recibidos deben separarse por
comas */
nombre_objeto.nombre_metodo(valor, valor2, valor3, valor4);

/*Cuando se llama a un método que no recibe valores pero sí regresa
valores */
tipo_variable variable = nombre_objeto.nombre_metodo( );

/*Cuando se llama a un método que sí recibe y regresa valores */
tipo_variable variable = nombre_objeto.nombre_metodo(valor );

/*Cuando se llama a un método que recibe más de un valor y en él
habrá valor de retorno. Los valores recibidos deben separarse con
comas */
tipo_variable variable = nombre_objeto.nombre_metodo(valor, va-
lor2);
```

Los valores recibidos pueden ser valores constantes o variables que almacenan algún valor.

Ejemplo 6.2 En la siguiente clase llamada `Principal_Matemáticas` se muestra cómo llamar el constructor y los métodos de la clase `Aritmética` (vista en el ejemplo 6.1).

```
package matemáticas;
import java.util.Scanner;
public class Principal_Matemáticas {
public static void main (String args []){

/*se crea un objeto llamado leer de la clase Scanner
  para leer de consola */
  Scanner leer = new Scanner(System.in);

      //se imprimen dos letreros y se leen dos valores
        System.out.print("Dame un número: ");
        double valor_1 = leer.nextDouble();
        System.out.print("Dame otro número: ");
        double valor_2 = leer.nextDouble();
/*se crea un objeto llamado operación de la clase
  Aritmética, se llama automáticamente al constructor
  de la clase Aritmética y se le manda la cadena
  "Creando un Objeto" como parámetro */
   Aritmética operacion=new Aritmética("Creando un Objeto",
valor_1,    valor_2);
```

```
//con el objeto operacion se llaman los siguientes métodos
  operacion.multiplicar();
  operacion.dividir();
/*se calcula la suma llamando al método sumar, notar que no
  se requiere declarar un objeto, basta con anteponer el
  nombre de la clase, porque sumar es static */
  double suma_1 = Aritmética.sumar(valor_1,  valor_2);
    //se imprime suma_1
      System.out.println("Suma="+suma_1);
    }
}
```

run:
Dame un número: 50
Dame otro número: 2,5
Creando un Objeto
Multiplicación = 125.0
División = 20.0
Suma = 52.5
BUILD SUCCESSFUL (total time: 4 seconds)

run:
Dame un número: −27,89
Dame otro número: 4,97
Creando un Objeto
Multiplicación = 138.6133
División = −5.611670020120725
Suma = −22.92
BUILD SUCCESSFUL (total time: 4 seconds)

En este ejemplo, se tiene una clase llamada `Principal_Matemáticas` que permite crear varios objetos de la clase `Aritmética`; en este caso se creó un objeto llamado *operacion*, con el que se ejecutaron los tres métodos de la clase (como se muestra en las dos salidas de ejecución de los recuadros, véase Figura 6.8).

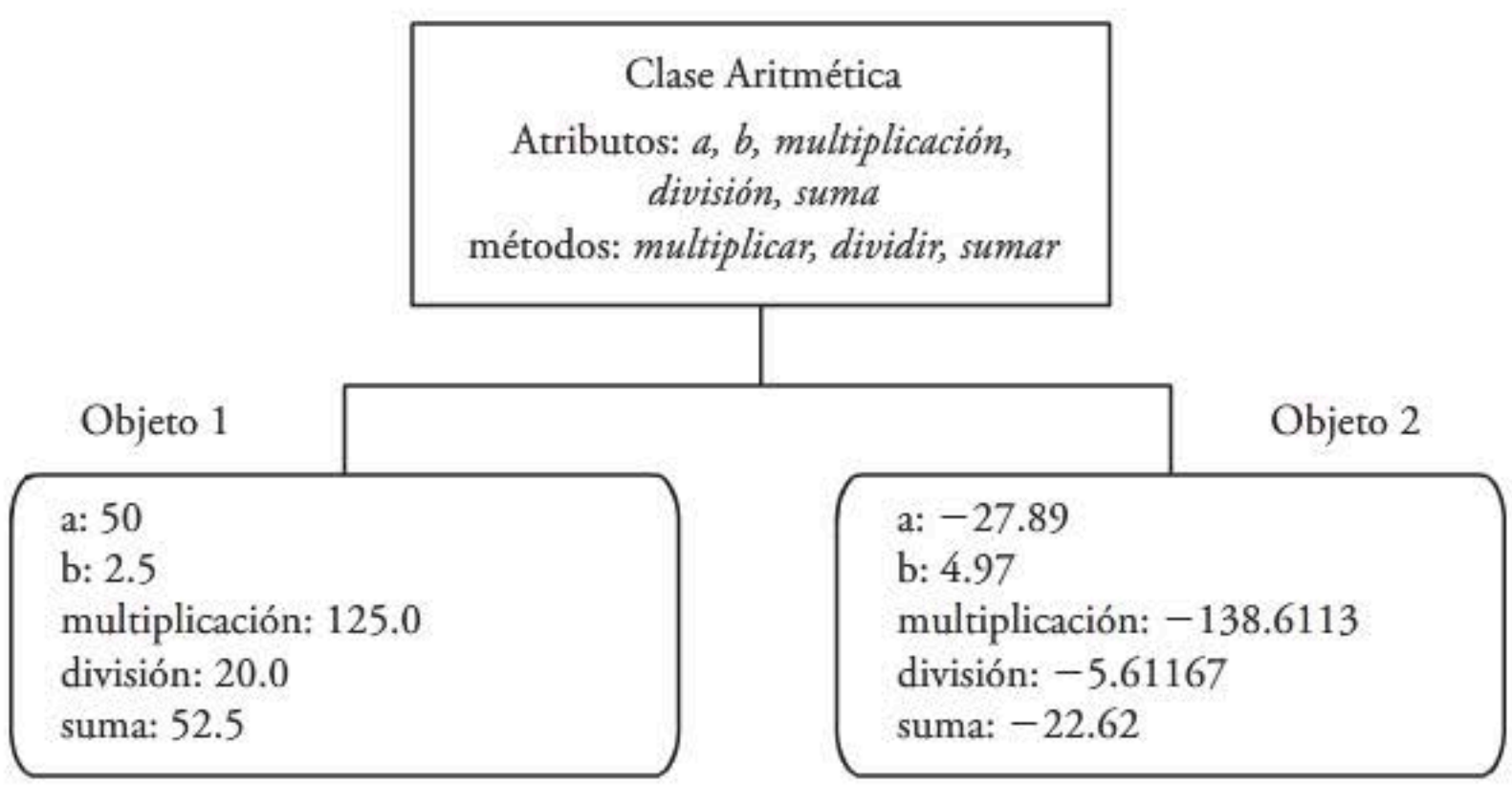


Figura 6.8. Objetos de la Clase Casa.

De manera que se puede resumir el funcionamiento de la clase `Principal_Matemáticas` y de la clase `Aritmética` a través de la siguiente figura:

```
package matematicas;
import java.util.Scanner;
public class Principal_Matematias {
public static void main (String args []){
  Scanner leer=new Scanner(System.in);
  System.out.print("Dame un número: ");
  double valor_1 = leer.nextDouble();
  System.out.print("Dame otro número: ");
  double valor_2 = leer.nextDouble();
  Aritmetica operacion=new Aritmetica("Creando  (1)
un objeto ", valor_1,valor_2);
  operación.multiplicar();    (2)
  operación.dividir(); (3)
  double suma_1=Aritmetica.sumar(valor_1,valor_2); (4)
  System.out.println("Suma= "+suma_1);
 }
}
```

Esta clase es la que contiene en main por eso es la que tiene el control sobre la ejecución

```
package matematicas;
public class Aritmetica {
  public double a;
  public double b;
  public double multiplicacion;
  public double division;
  public static double suma;

  public Aritmetica (String mensaje,double x,double y){
   System.out.println (mensaje);
   a = x;
   b = y;
   multiplicacion = 0;
   division = 0;
  }

  public void multiplicar() {
    Multiplicación = a*b;
  }

  void dividir () {
    division = a/b;
    System.out.println ("División = "+division);
  }
public static double sumar(double n, double m){
  suma = n+m;
  return suma;
 }
}
```

1. Se crea un objeto llamado `operacion` de la clase `Aritmética`, automáticamente se invoca al constructor de la clase Aritmética (porque se crea un objeto) y se mandan los valores para inicializar los atributos de la clase `Aritmética`
2. Utilizando el objeto operacion se llama al método `multiplicar`.
3. Utilizando el objeto operacion se llama al método `dividir`.
4. Se llama al método sumar de la clase Artimética sin hacer uso de un objeto de esa clase porque el método está definido como static; una vez que se realizan las acciones en el método sumar, éste devuelve el resultado para que se guarde en la variable suma_1.

Figura 6.9. Comportamiento de las Clases `Principal_Matemáticas` y `Aritmética`.

Ejemplo 6.3 Tomando en cuenta el siguiente diagrama (figura 6.10), crear una clase llamada `Perro` con los atributos y métodos indicados, y otra llamada `PerroPrin` la cual tendrá la función de inicio de ejecución para poder crear dos objetos de tipo `Perro`.

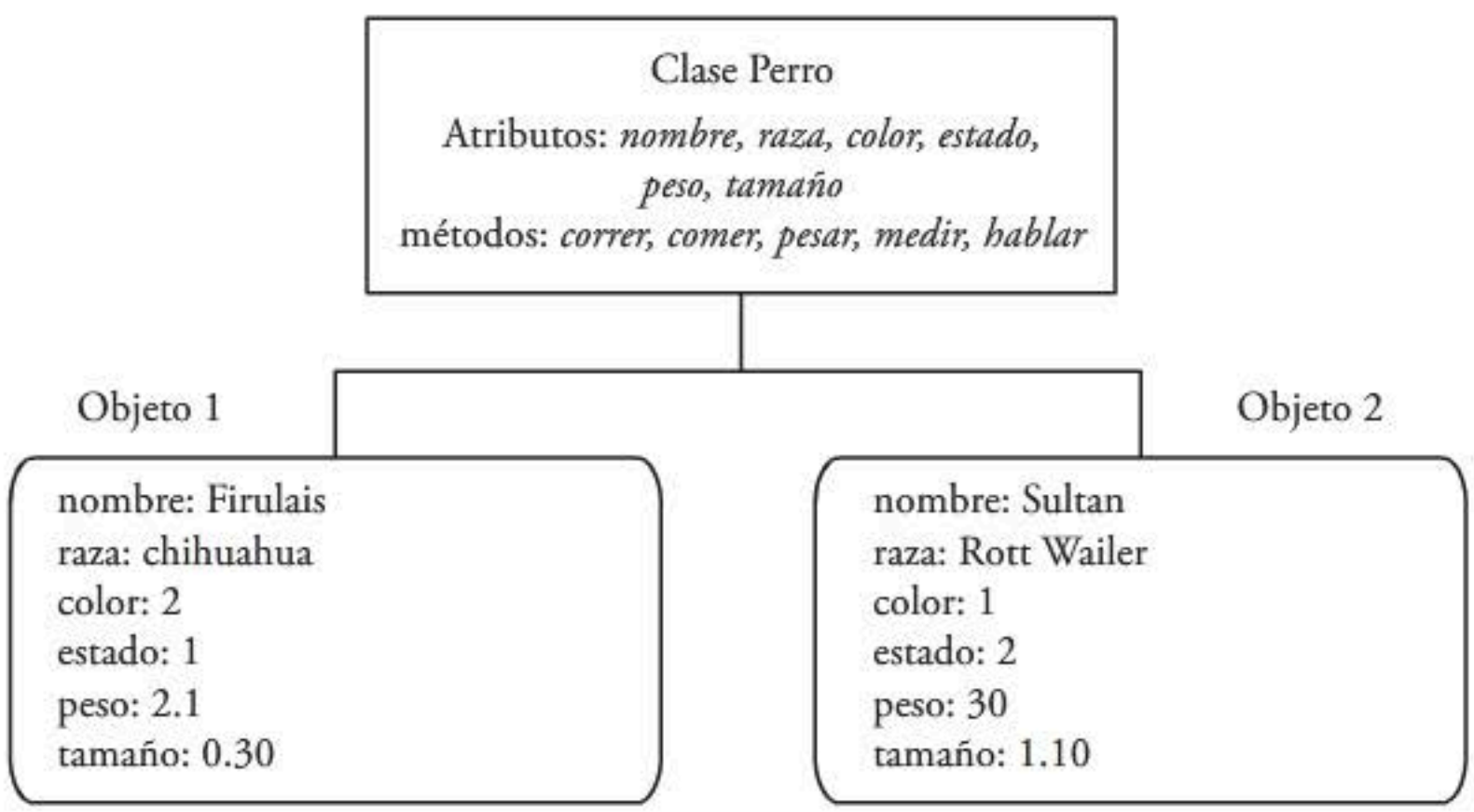


Figura 6.10. Objetos de la Perro.

Clase Perro

```
package perritos;
public class Perro {

//atributos privados de la clase
  private int color,estado;
  private float peso,tamanio;
  private String raza;
```

```
//atributo accesible desde fuera de la clase
  public String nombre;

//constructor
public  Perro(String n, int c, float p, float t, String r, int e){
System.out.println("Soy el constructor, se acaba de crear "
+ " un objeto");

/* se inicializan los atributos de la clase con el valor de los
Parámetros*/

    nombre=n;
    peso=p;
    tamanio=t;
    raza=r;
    estado= (e>=1 && e<=3) ? e:2;
    color= (c>=1 && c<=3) ? c:1;
}

//método comer
  public void comer(float cant){
  //al comer se modifican el peso y el estado
  peso+=cant/20;
  estado=3;
}

//método correr
  public void correr(){
  //al correr se modifican el peso y el estado
    peso-=peso/50;
    if (estado>1)
      estado--;
  }

//método hablar
  public void hablar(){
  System.out.println("Me llamo "+nombre+" peso "+peso+" kgs");
  System.out.println("Mido "+tamanio+" mts "+" soy de la raza"+raza );
  System.out.print("Soy de color ");
  switch(color){
      case 1: System.out.print("negro "); break;
      case 2: System.out.print("blanco "); break;
      case 3: System.out.print("cafe "); break;
    }
```

```
    System.out.print(" y en este momento me siento  ");
  switch(estado){
      case 1: System.out.println("hambriento "); break;
      case 2: System.out.println("satisfecho "); break;
      case 3: System.out.println("lleno "); break;
    }
    System.out.println();
  }
//método pesar
  public float pesar(){
    return peso;
  }

//método medir
  public float medir(){
     return tamanio;
  }
}// Llave de cierre de la clase Perros

//Clase PerroPrin
package perritos;
public class PerroPrin {
  public static void main(String[] args){

   //Se crea el objeto perro1
Perro perro1=new Perro("Firulais ", 2, 2.1f, 0.30f, "Chihuahua", 1);

//Se crea el objeto perro2
Perro perro2= new Perro ("Sultan", 1, 30f, 1.10f, "Rottweiler", 2);

 float alim=0.200f;

//MENSAJES QUE SE ENVIARÁN AL OBJETO perro1
//alim se pasa como argumento, el perro1 come
  perro1.comer(alim);
 System.out.println("Después de comer "+alim+" kgs de alimento, "
+ "el perro  1 habla:");
   perro1.hablar();

//MENSAJES QUE SE ENVIARÁN AL OBJETO perro2
  perro2.correr();
  perro2.correr();
 System.out.println("Después de correr 2 veces el perro 2 "+" habla:");
  perro2.hablar();
 }
}
```

La clase `Perro_Prin` que es donde se crean 2 objetos de la clase `Perro` (`perro1` y `perro2`), envía diferentes mensajes a cada objeto a selección del usuario. La evidencia de que el constructor se ejecuta automáticamente al crear objetos puede apreciarse en los mensajes *"Soy el constructor, se acaba de crear un objeto"*. Observar que cuando se llaman los métodos `comer` y `correr`, los valores de los atributos `peso` y `estado` se ven modificados, ambos disminuyen cuando realizan la acción de `correr` y aumentan con la acción de `comer`. A continuación se muestra la ejecución de la clase `Perro_Prin`, que hace uso de la clase `Perro`:

```
run:
Soy el constructor, se acaba de crear un objeto
Soy el constructor, se acaba de crear un objeto
Después de comer 0.2 kgs de alimento, el perro 1 habla :
Me llamo Firulais peso 2.11 kgs
Mido 0.3 mts  soy de la raza Chihuahua
Soy de color blanco  y en este momento me siento  lleno

Después de correr 2 veces el perro 2 habla :
Me llamo Sultán peso 28.812 kgs
Mido 1.1 mts  soy de la raza Rottweiler
Soy de color negro  y en este momento me siento  hambriento

GENERACIÓN CORRECTA (total time: 0 seconds)
```

Ejemplo 6.4 Retomando el comportamiento de clase `teléfono_celular` mencionado previamente (Sección 6.1), hacer una clase con el nombre `Celular`, con 3 atributos enteros que serían los números frecuentes y un atributo flotante para almacenar el saldo, todos privados. La clase también deberá contener los siguientes métodos:

- Un método constructor a través del cual sea posible inicializar los números frecuentes del teléfono, así como el saldo inicial de éste.
- Un método para realizar llamadas a cierto número. El número al cual llamar deberá ser el parámetro del método. Si se llama a un número frecuente el costo por minuto de la llamada será de $2.50, de lo contrario se cargará un costo de $5.50. Considerar que haya saldo suficiente para iniciar y continuar la llamada por varios minutos.
- Un método para enviar mensajes de texto, considerando que el costo por mensaje es de $2.00 y que haya saldo suficiente para enviar el mensaje.
- Un método para conocer el saldo actual del teléfono.
- Un método para abonar saldo al teléfono. El saldo a abonar se hará a través de un parámetro de tipo entero.

Se requiere:

1.- Diseñar la clase `Celular` con las características mencionadas.
2.- Crear 2 objetos de clase `Celular`.
3.- Enviar diferentes mensajes a cada objeto y observar cómo se ven afectados los valores de los atributos de cada objeto.

Clase Celular

```
package telcelulares;
import java.util.Scanner;
import java.io.*;
class Celular{
//atributos para números frecuentes
  private int numfec1,numfec2,numfec3;
  private float saldo;     //atributo para el saldo
  Scanner leer=new Scanner(System.in);

//constructor
  public Celular(int n1, int n2, int n3, float sal){
  numfec1=n1;  // Se inicializan números frecuentes y saldo
  numfec2=n2;
  numfec3=n3;
  saldo=sal;
 }

//método llamar
  public void llamar(int num){
  //Se determina el costo por minuto
    float costominuto=5.5f;

    int minutos=0,resp=1;
    if (num==numfec1 || num==numfec2 || num==numfec3)
       //cambia el costo/min al ser número frecuente
      costominuto=2.5f;

 while ((saldo>=costominuto) && resp==1){
   minutos++;
   saldo-=costominuto;
  System.out.print("Hasta el momento ha hablado "+ minutos+ " mi-
nutos, a"+ "un costo de "+costominuto+" ¿desea otro minuto? 1=SI
2=NO ");
   resp=leer.nextInt();
 }
   if (resp==2)
    System.out.println("Al colgar, usted ha hablado " + minutos +
" minutos");
   if (resp==1 && saldo<=costominuto)
  System.out.println("Se agotó el saldo, usted ha hablado "
 + minutos + " minutos");
}

//método para enviar mensaje
//throws es exigido por la clase bufferedReader
```

```
  //para lectura de cadenas con el objeto entrada
    BufferedReader entrada=new BufferedReader
(new InputStreamReader(System.in));

    if (saldo>=2){
      System.out.println("Introduzca el mensaje: ");
      msg=entrada.readLine();
      System.out.println("Se ha enviado el MSG "+msg);
      saldo-=2;
    }
    else
    System.out.println("Imposible mandar mensaje,
SALDO INSUFICIENTE, abone ficha \n\n");
  }

//método para conocer saldo actual
  public float saldo(){
    return saldo;
  }

//método para abonar saldo
  public void abonar(int ficha){
    saldo+=ficha; //se acumula el valor de ficha al saldo
    System.out.println("Usted ha abonado $" + ficha +
" pesos de saldo");
  }
}
```

Clase TelCelular

```
public class TelCelular {
public static void main(String []args) throws IOException{

/*Se crea objeto, primeros argumentos son números
frecuentes, el último es el saldo */
  Celular t1=new Celular(540,1010, 1014,20);
  Celular t2=new Celular(600,700, 800,50);
  System.out.println("INICIALMENTE El saldo del teléfono 1 es $" +
t1.saldo());
  System.out.println("Llamando con el teléfono 1 ");
  t1.llamar(1010);
  System.out.println("Enviando mensaje con el teléfono 1");
  t1.mensaje();
```

```
  System.out.println("Se abonaran $30 al teléfono 1");
  t1.abonar(30);
  System.out.println("DESPUES DE USAR el teléfono 1 su saldo es $" +
t1.saldo());
  System.out.println("INICIALMENTE El saldo del teléfono 2 es $" +
t2.saldo());
  System.out.println("Enviando mensaje con el teléfono 2");
  t2.mensaje();
  System.out.println("DESPUES DE USAR el teléfono 2 su saldo es $" +
t2.saldo());
  }
}
```

```
run:
INICIALMENTE El saldo del teléfono 1 es $20.0
Llamando con el teléfono 1
Hasta el momento ha hablado 1 minutos, a un costo de 2.5 ¿desea otro minuto? 1=SI 2=NO  1
Hasta el momento ha hablado 2 minutos, a un costo de 2.5 ¿desea otro minuto? 1=SI 2=NO  2
Al colgar, usted ha hablado 2 minutos
Enviando mensaje con el teléfono 1
Introduzca el mensaje:
ENVIAME EL FORMATO AHORA  POR FAVOR
Se ha enviado el MSG ENVIAME EL FORMATO AHORA  POR FAVOR
Se abonaran $30 al teléfono 1
Usted ha abonado $30 pesos de saldo
DESPUES DE USAR el teléfono 1 su saldo es $43.0
INICIALMENTE El saldo del teléfono 2 es $50.0
Enviando mensaje con el teléfono 2
Introduzca el mensaje:
CLARO, LO ENVIO EN UN MOMENTO
Se ha enviado el MSG CLARO, LO ENVIO EN UN MOMENTO
DESPUES DE USAR el teléfono 2 su saldo es $48.0
GENERACIÓN CORRECTA (total time: 22 seconds)
```

Es importante recordar que las variables que se declaran dentro de la clase (es decir, los atributos) se reconocen en todos los métodos. Las variables que se declaran dentro de un método (variables locales) sólo serán reconocidas dentro del método en el cual se declararon. El hecho de que todos los métodos conozcan a los atributos de la clase ahorra en gran medida el paso de parámetros de un método a otro.

Para la lectura de cadenas desde el teclado, se utilizó en este ejemplo la línea:

```
BufferedReader entrada = new BufferedReader (new InputStreamReader
(System.in));
```

La explicación en detalle del funcionamiento de las clases `BufferedReader` y `InputStreamReader` salen del contexto de esta obra, pero se puede decir que la línea de código anterior crea un objeto de la clase `InputStreamReader` que es usado para crear e inicializar el objeto identificado como `entrada` que pertenece a la clase `BufferedReader`. Lo anterior permite leer líneas de caracteres, incluyendo como parte de ésta el carácter espacio, desde el teclado, usando el método `readLine()`.

En ocasiones el leer una línea de caracteres con un objeto de la clase `Scanner` después de haber efectuado lecturas de datos anteriormente, la lectura de caracteres es ignorada, de ahí que se utiliza la lectura del mensaje con la clase `BufferedReader`.

En el programa se hace uso de la instrucción **`throws IOException`** al final de los paréntesis de los métodos **`main`**`()` y `mensaje()`. El método **`readLine`** demanda manejar las posibles excepciones que pudieran lanzarse en caso de que se diera un error en la entrada. Al escribir **`throws IOException`** enseguida de los paréntesis de los métodos que llaman al método **readLine**, se delega al sistema de manejo de excepciones la excepción (en caso de llegar a producirse), por lo que ya no es necesario escribir código extra para manejarla de manera particular.

6.7 Sobrecarga de métodos

Es una técnica que permite definir diferentes versiones de un método, todos con el mismo nombre pero *diferente lista de parámetros* (número y/o tipo). De esta manera, cuando se invoque al método sobrecargado, el compilador de Java determinará cuál versión elegir dependiendo del tipo y/o número de parámetros que requiere cada método.

Es importante mencionar que cada versión debe ser única en su lista de parámetros, de lo contrario el compilador marcará un error.

Si se deseara impedir la sobrecarga, sólo es necesario colocar la palabra **`final`** en un método, de esta manera, no existirán dos métodos con un mismo nombre.

Ejemplo 6.5. En el siguiente programa se muestra una clase llamada `Versiones_Metodos` la cual tiene un método llamado `calcular` que no puede sobrecargarse porque tiene la palabra **`final`**, mientras que el método `multiplicar` aparece sobrecargado (hay varias versiones de él).

```
package versiones;
public class Versiones_Metodos {

  /*no se puede redefinir ni anular en subclases porque tiene fi-
nal */
    public final int calcular (){
        return 0;
    }

  //primera versión del método multiplicar
    public int multiplicar(){
        return 1;
    }
```

```
 //segunda versión del método multiplicar
    public int multiplicar(int a){
        return a*2;
    }
  //tercera versión del método multiplicar
    public double multiplicar(double a){
       return 3.4*a;
    }
}
```

Si se quiere ver el funcionamiento de la clase anterior es necesario crear otra clase que tenga la sentencia de inicio de ejecución (**main**) o incluirla en la clase anterior y llamar los métodos. La siguiente clase muestra cómo llamar los métodos de `Versiones_Metodos` desde la clase `Principal_Versiones`.

```
package versiones;
public class Principal_Versiones {
public static void main (String args []){
  //se crea un objeto de la clase Versiones_Metodos
    Versiones_Metodos v = new Versiones_Metodos();

  //se imprime lo que devulven los métodos
    System.out.println("Esto es calcular: "+v.calcular());
    System.out.println("Esto es multiplicar: "+ v.multiplicar());
    System.out.println("Esto es multiplicar: "+ v.multiplicar
(100));
    System.out.println("Esto es multiplicar: "+ v.multiplicar
(23.789));
 }
}
```

```
run:
Esto es calcular: 0
Esto es multiplicar: 1
Esto es multiplicar: 200
Esto es multiplicar: 80.8826
BUILD SUCCESSFUL (total time: 0 seconds)
```

6.8 Herencia

Esta técnica permite que una clase hija (clase que hereda de otra) adquiera de la clase padre las siguientes propiedades:

1.- Automáticamente obtiene todos los atributos (variables) miembro de la clase padre.
2.- Obtiene todos los métodos miembro de la clase padre; por lo que, la clase hija podrá funcionar de la misma manera que la padre.
3.- La clase hija puede definir nuevos atributos (variables) y métodos (funciones).

Una clase sólo puede heredar de una única clase (en Java no hay herencia múltiple).

Para utilizar la herencia en Java se emplea la palabra reservada **extends**, la cual indica que la nueva clase creada (hija) será una extensión (heredará) de la clase que se escribe justo después de la palabra **extends**.

Ejemplo 6.6. En el siguiente ejemplo se tienen dos clases: clase padre (`Programa_B`) y clase hija (`Programa_A`). La clase `Programa_A` hereda (**extends**) de la clase padre (`Programa_B`) sus métodos y atributos; de manera que no es necesario que la clase hija cree una instancia (objeto) de la clase padre para poder llamar o utilizar sus métodos y/o atributos; por lo que, es como si todo lo que está escrito en `Programa_B` estuviera escrito dentro del `Programa_A`.

Clase Padre

```
public class Programa_B {
    public static double peso = 50.4;
    public static double altura = 1.45;
    public static double ancho;

    public Programa_B(){
    ancho = 3.90;
    System.out.println("El ancho es: "+ancho);
    System.out.println("El peso es: "+peso);
   }

    public static double multiplicarPeso(double mult){
        return peso*mult;
    }

    public static double multiplicarAltura(double mult){
        return altura*mult;
    }
}
```

Clase Hija

```
public class Programa_A extends Programa_B{
public static void main(String[] args) {
   double resultado = multiplicarPeso(7);
   System.out.println("Resultado Peso="+resultado);
  System.out.println("Resultado Altura="+ multiplicarAltura(10));
  }
}
```

```
run:
Resultado Peso=352.8
Resultado Altura=14.5
BUILD SUCCESSFUL (total time: 0 seconds)
```

Si se desea evitar que una clase herede a otra, sólo es necesario definirla como **final**, por ejemplo:

```
public final class Programa_B {
      //sentencias
}
```

De esta manera, la clase `Programa_B` no podrá tener clases hijas (heredar). Por medio de la figura 6.11 es posible entender lo mencionado anteriormente.

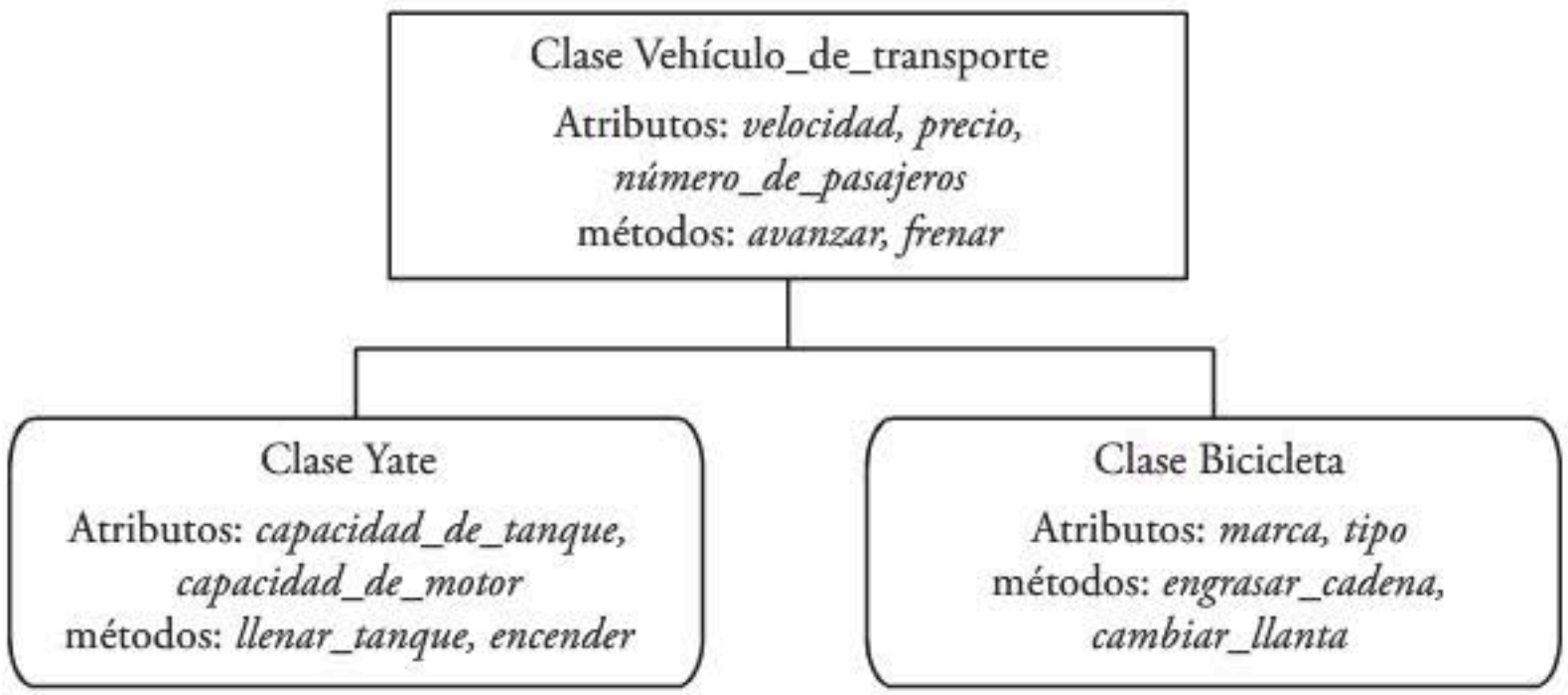


Figura 6.11. Objetos de la clase Vehículo_de_transporte

La figura muestra la relación de herencia de la clase `vehículo_de_transporte` con las clases `yate` y `bicicleta`. La clase `yate` recibe herencia de la clase `vehículo_de_transporte`, por lo que contiene los atributos `velocidad`, `precio`, `número_de_pasajeros`, `capacidad_de_tanque` y `capacidad_de_motor`; los primeros tres heredados y los dos últimos definidos en ella. También contiene los métodos `avanzar`, `frenar`, `llenar_tanque` y `encender`; los primeros dos recibidos por herencia y los dos últimos definidos en ella.

De la misma manera la clase `bicicleta` recibe herencia de la clase vehículo de transporte, por lo que contiene los atributos `velocidad`, `precio`, `número_de_pasajeros`, `marca` y `tipo`; los primeros tres heredados y los últimos dos definidos en ella. Contiene también los métodos `avanzar`, `frenar`, `engrasar_cadena` y `cambiar_llanta`; los primeros dos recibidos por herencia y los dos últimos definidos en ella. A través de la herencia es posible crear clases más específicas a partir de clases más genéricas, pasando por herencia características comunes que son de utilidad a clases más particulares. La clase `yate` debe ser diferente de la clase `bicicleta`, pues ésta última no tiene un motor ni un tanque, sin embargo tienen cosas comunes como número de pasajeros, precio y velocidad, además de que ambas son capaces de avanzar y frenar.

6.9 Otros conceptos del paradigma de la programación orientada a objetos

Dentro del paradigma de la POO existen otras propiedades claves que deben comprenderse entre las cuales se pueden mencionar:

Abstracción: Es concentrarse sobre las características o hechos más importantes de los objetos, dejando de lado los detalles. Por ejemplo: Se conoce lo que es *enviar un mensaje por teléfono celular* y al pensar en ello olvidamos detalles como que al elegir una letra para escribir el mensaje presionamos cierta área del teclado que activará un circuito que enviará una señal para que aparezca una letra en una pantalla y que para que aparezca la letra se prenden o se apagan ciertas áreas de ella para dar la apariencia de la letra y que para elegir un destinatario se deben buscar los datos de éste en forma de un tipo de archivo, etc. Con la abstracción la concentración está directamente en lo importante de las cosas, sin considerar detalles.

Polimorfismo: Es la propiedad que tienen los objetos de responder de manera diferente a un mismo mensaje. Si se piensa en el mensaje (acción) *acomodar*, los objetos responderán de manera diferente dependiendo de objeto que se trate. Si se trata de acomodar latas de refresco, éstas se acomodarían con la base circular como apoyo, si se trata de acomodar cajas, el acomodo dependería si se trata de cajas de chocolate, cajas vacías, cajas de papel, etc. Si el mensaje fuera comunicar, la comunicación sería diferente con un objeto humano que con un objeto delfín.

Encapsulamiento: Concepto que se refiere a integrar los elementos necesarios dentro de una entidad. Por ejemplo los atributos velocidad, precio, número de pasajeros y los métodos avanzar y frenar están encapsulados en la clase Vehículo de transporte. Los atributos marca, tipo y los métodos engrasar cadena y cambiar llanta están encapsulados en la clase Bicicleta.

Ocultamiento: Propiedad que permite aislar o proteger del exterior a los atributos de un objeto, de tal manera que no puedan ser modificados de manera arbitraria.

Modularidad: Es la propiedad que exige dividir una aplicación grande en partes más pequeñas, independientes entre sí. Las acciones necesarias para el método *correr*, son diferentes e independientes de las necesarias para el método *saltar*, por ejemplo.

6.10 Resumen

El paradigma de la Programación Orientada a Objetos (POO) es actualmente utilizado por una gran cantidad de desarrolladores de *software* debido a las ventajas que representa. Gran parte de su popularidad se debe a que la manera en que se modelan las soluciones de problemas es muy similar al razonamiento humano, basado en clases y objetos. Los orígenes de la POO se comenzaron a hacer evidentes en lenguajes como SIMULA 67, Smalltalk, pasando por C++ hasta llegar a lenguajes como Java.

Los beneficios que ofrece el paradigma de la POO se hacen presentes en sus características de flexibilidad, encapsulamiento de información, escalabilidad, reutilización, etc.

A través de mecanismos como la herencia y el polimorfismo el *software* construido bajo este paradigma puede reutilizar gran parte de código construido previamente para crear nuevos productos de *software*.

6.11 Problemas

1. Responda las siguientes preguntas.

a) ¿Cuáles son los nombres de los lenguajes de programación que fueron dando origen al paradigma de Programación Orientada a Objetos?.

b) ¿Cuáles son los beneficios del paradigma de Programación Orientado a Objetos? .
c) ¿Cómo se llama la característica de la POO que permite transmitir sus características y métodos de una clase a otra?.
d) ¿Cómo se llama la característica de la POO que permite la concentración en los aspectos importantes de los objetos, dejando de lado los detalles para la construcción del mismo?.
e) ¿Cómo se llama la propiedad de la POO que hace posible que dos o más objetos respondan de manera diferente a un mismo mensaje?.
f) ¿Cómo se le llama a la propiedad de la POO que permite integrar los atributos y métodos necesarios en una misma clase?.
g) ¿Cómo se le llama a la propiedad de la POO que permite proteger a los atributos y miembros de una clase de una entidad externa?.
h) ¿Cómo se le llama a la propiedad de la POO que divide las acciones de los objetos en tareas más pequeñas, claramente definidas e independientes?

2. Escriba 3 métodos y 3 atributos para cada una de las clases de los siguientes incisos:

a) Clase Juguete
b) Clase Ropa de playa
c) Clase Medio de almacenamiento digital
d) Clase Publicación impresa
e) Clase Alimento chatarra

3. Escriba el nombre de tres objetos para cada una de las clases de la pregunta anterior, asignando un valor para cada uno de sus atributos.

4. Escriba en la columna **Concepto** el nombre que mejor defina en cada uno de los siguientes enunciados.

Definición	Concepto
Es el conjunto de características y acciones a partir del cual se crean los objetos	
Son las acciones que los objetos son capaces de realizar y representan su comportamiento a través de código	
Son las características que tienen los objetos y representan los datos de la clase	
Está determinado por el valor de los atributos de un objeto en un momento específico	
Es un ejemplo particular o instancia de una clase	
Calificador de los atributos de una clase que los hace totalmente accesibles desde cualquier entorno.	
Calificador de los atributos utilizados para el ocultamiento y protección de los mismos.	
Método especial que se llama igual que la clase a la que pertenece y se utiliza para inicializar los atributos	
Es la petición de ejecución de un método a través de un objeto	

5. Construya una clase RADIO con los atributos pertinentes, capaz de subir volumen, bajar volumen, sintonizar estación, encender y apagar. Considere que no es posible realizar acción alguna con la radio cuando está apagada. Una salida típica se muestra a continuación:

```
Radio Apagada
 MENU
1. Encender          2. Apagar          3. Sintonizar
4. Subir Volumen     5. Bajar Volumen   6. Abandonar

¿Qué acción desea ejecutar?: 1
Se acaba de encender
¿Qué acción desea ejecutar?: 3
¿Qué estación?  (1  a la 10): 7
Volumen ||| Estación 7  Otra estación ?  1=SI   2=NO: 2
¿Qué acción desea ejecutar?: 5
Volumen ||| Estación 7
¿Bajar más?  1=SI   2=NO: 1
Volumen || Estación 7
¿Bajar más?  1=SI   2=NO: 2
¿Qué acción desea ejecutar?: 6
ADIOS
```

6. Escribir una clase TEXTO, que incluya como uno de sus atributos un arreglo de caracteres, el cual debe inicializarse en el constructor, al momento de crear objetos, capaz de recorrer sus caracteres a la izquierda, recorrer sus caracteres a la derecha y mostrar sus caracteres.

```
OMURCIELAG
GOMURCIELA
AGOMURCIEL
LAGOMURCIE
ELAGOMURCI
IELAGOMURC
CIELAGOMUR
IELAGOMURC
ELAGOMURCI
LAGOMURCIE
```

7. Construir una clase AHORRO capaz de ahorrar (indicando como parámetro la cantidad a ahorrar), retirar (indicando como parámetro la cantidad a retirar y devolviendo la cantidad que se pudo retirar) y mostrar_ahorro (para conocer el monto actual del ahorro). Utilizar en el constructor un parámetro que indique la cantidad inicial de ahorro, si el valor del parámetro es negativo, la alcancía iniciará con un ahorro inicial de 150 pesos.

```
Al momento tienes un ahorro de $100 pesos
Has retirado 50 pesos
Al momento tienes un ahorro de $50 pesos
Al momento tienes un ahorro de $285 pesos
Has retirado 285 pesos
Al momento tienes un ahorro de $0 pesos
```

8. Escriba **V** para verdadero y **F** para falso a cada una de las siguientes afirmaciones.

- La POO se inició con la creación de C++ ( )
- Enviar un mensaje a un objeto es sinónimo de ejecutar un método a través de un objeto ( )
- Es posible crear un objeto sin la existencia de clases ( )
- El constructor puede ser invocado cada vez que se necesite reinicializar los atributos de un objeto ( )
- El constructor se ejecuta cada vez que un objeto es creado ( )
- Para que los métodos puedan accesar a los atributos de su clase, ésos últimos deben tener un acceso público ( )
- El método constructor no tiene tipo ( )
- Los elementos calificados como públicos en una clase son accesados sin restricciones en cualquier entorno ( )
- Los métodos no pueden ser públicos ( )
- Si una clase A recibe herencia de una clase B, entonces contendrá los atributos declarados dentro de ella más los de B ( )

9. Diseñar una clase llamada *LIBRETA* que contenga los atributos *número de hojas totales, hojas limpias y hojas utilizadas*, así como el estado de cada una de sus hojas. Cada hoja puede estar limpia, utilizada o haber sido arrancada. El comportamiento de la clase será así:

- Al crear una libreta se indicará cuantas hojas iniciales contendrá. El mínimo de hojas es 10 y el máximo es de 150. Si hay algún intento de crear una libreta con un número de hojas fuera de los rangos mencionados la libreta se creará con 50 hojas por default.
- Utilizar un arreglo de caracteres y tomar cada una de sus celdas para considerar el estado de cada una de las hojas; '0' es hoja limpia, 'A' es hoja arrancada y 'x' es hoja utilizada.
- Es posible arrancar hojas de la libreta, indicando el número de hoja que se quiere arrancar, pueden arrancarse hojas una a una de manera repetitiva, hasta indicar un número de hoja inexistente.
- Es posible utilizar hojas de la libreta, indicando el número de hoja que se quiere utilizar, pueden utilizarse hojas una a una de manera repetitiva, hasta indicar un número de hoja inexistente.
- Es posible agregar hojas a la libreta considerando que no debe rebasar el límite de 150 hojas.
- Es posible ver las características actuales de la libreta para lo cual debe imprimirse el número de hojas de la libreta, cuántas han sido arrancadas, cuántas están utilizadas y cuántas están limpias.

Se requiere:

1.- Diseñar la clase LIBRETA de acuerdo a las especificaciones anteriores.
2.- Crear un objeto de la clase LIBRETA.
3.- Enviar los diferentes mensajes posibles al objeto creado con el fin cambiar y ver el estado del objeto en diferentes momentos.

```
ANTES de enviar mensajes al objeto se Scribe, su estado es:
Libreta de:10 hojas, 10 están limpias,0 están utilizadas y 0 están arrancadas
Que hoja quieres arrancar? 1- 10  2
Que hoja quieres arrancar? 1- 10  13
No existe tal hoja esta libreta solo era de 10 hojas
Cual hoja utilizaras?5
Cual hoja utilizaras?13
No existe tal hoja esta libreta solo era de 10 hojas
Cuantas deseas agregar?5
DESPUES de enviar mensajes al objeto se escribe, su estado es:
Libreta de:15 hojas, 13 están limpias,1 están utilizadas y 1 están arrancadas
```

10. Diseñar una clase llamada `Intercambiar` en donde se realice lo siguiente:

- Se creará un arreglo de caracteres con ciertas dimensiones y características en el constructor.
- El método constructor debe tener tres parámetros. Los dos primeros servirán para definir el número de renglones y número de columnas con los que se creará el arreglo. El tercer parámetro será utilizado como el primer valor carácter a colocar en el arreglo, el resto de las celdas se llenará con caracteres ASCII sucesivos a partir del carácter de inicio (*Si el carácter de inicio fue 'g', el resto de las celdas se llenará con 'h','i', etc.*). Considerar la creación del arreglo con una dimensión de 3×8 por default cuando la dimensión propuesta sea inválida *(valores negativos o ceros).*
- Un método `mostrar()` para presentar en pantalla el contenido del arreglo.
- Un método `intercolumnas()` para hacer el intercambio de los caracteres que se encuentren en 2 columnas. El método debe considerar dos parámetros que indicaran las columnas a intercambiar. El método debe mostrar el contenido del arreglo una vez hecho el intercambio.
- Un método `interrenglones()` para hacer el intercambio de los caracteres que se encuentren en 2 renglones. El método debe considerar dos parámetros que indicaran los renglones a intercambiar. El método debe mostrar el contenido del arreglo una vez hecho el intercambio.

```
run:
Se ha creado un arreglo de 4 X 8
A    B    C    D    E    F    G    H
I    J    K    L    M    N    O    P
Q    R    S    T    U    V    W    X
Y    Z    [    \    ]    ^    _    `
Despues de intercambiar columna 1 por columna 3
A    D    C    B    E    F    G    H
I    L    K    J    M    N    O    P
Q    T    S    R    U    V    W    X
Y    \    [    Z    ]    ^    _    `
Despues de intercambiar renglon 0 por renglon 2
Q    T    S    R    U    V    W    X
I    L    K    J    M    N    O    P
A    D    C    B    E    F    G    H
Y    \    [    Z    ]    ^    _    `
```

(continúa en la siguiente página)

```
Despues de intercambiar columna 1 por columna 2
Q    S    T    R    U    V    W    X
I    K    L    J    M    N    O    P
A    C    D    B    E    F    G    H
Y    [    \    Z    ]    ^    _    `
GENERACIÓN CORRECTA (total time: 4 seconds)
```

(continúa de la página 274)

11. Observe el siguiente código y la salida resultado de su ejecución:

```
public class Fracciones_Prin {
public static void main(String[] args) {
Fracciones q=new Fracciones();   // Se crea el objeto

//  Se envían los diferentes mensajes al objeto q
  //permite ingresar dos fracciones
    q.ingresar_fracciones();

  //suma y muestra el resultado de las fracciones
    q.sumar();

  //resta y muestra el resultado de las dos fracciones
    q.restar();

  //multiplica y muestra el resultado de las fracciones
    q.multiplicar();

  //divide y muestra el resultado de las dos fracciones
    q.dividir();
  }
}
```

```
run:

Valor del primer numerador:  3
Valor del primer denominador:  4
Valor del segundo numerador:  −7
Valor del segundo denominador:  8
Las fracciones ingresadas son:  3/4  y  −7/8
3/4 +  −7/8 = −4/32  que simplificado es  −1/8
3/4 −  −7/8 = 52/32  que simplificado es  1  5/8
3/4 *  −7/8 = −21/32  que simplificado es  −21/32
3/4 /  −7/8 = −24/28  que simplificado es  −6/7

GENERACIÓN CORRECTA (total time: 4 seconds)
```

Diseñe la clase necesaria para que el código que acaba de analizar dé como resultado la salida mostrada, considerando agregar un método privado que simplifique el resultado de la operación.

12. Para la multiplicación de 2 matrices es indispensable que el número de columnas de la primera matriz sea igual al número de renglones de la segunda, generando como resultado una matriz con el número de renglones de la primera y el número de columnas de la segunda. Considerando lo anterior, observe el siguiente código y la salida resultado de su ejecución. A continuación diseñe la clase necesaria para obtener la salida mostrada, considerando 3 matrices como atributos privados y respetando el envío de mensajes tal como se muestra en la clase Mult_Matrices_Prin.

```
public class Mult_Matrices_Prin {
public static void main(String[] args) {

/* Se crea un objeto del tipo Matrices, los argumentos del cons-
tructor son las dimensiones para crear las 3 matrices, dos que
habrán de multiplicarse y una tercera que recibirá el resulta-
do. El primer argumento es el número de renglones de la matriz
1, el segundo  argumento es el número de columnas de la  matriz
1 y a la vez el número de renglones de la matriz 2,el tercer
argumento es el número de columnas de la matriz 2  */
Matrices objetoM=new Matrices(3,2,4);

/* Se envía el mensaje ingresar_valores, que meterá valores
aleatorios entre 0 y 8 a las dos matrices que han de multipli-
carse  */
objetoM.ingresar_valores();

/*Se envía el mensaje multiplicar y el resultado quedará en otra
matriz */
objetoM.multiplicar();

 //Se envía el mensaje mostrar para ver la matriz resultante
 objetoM.mostrar();      }
}
```

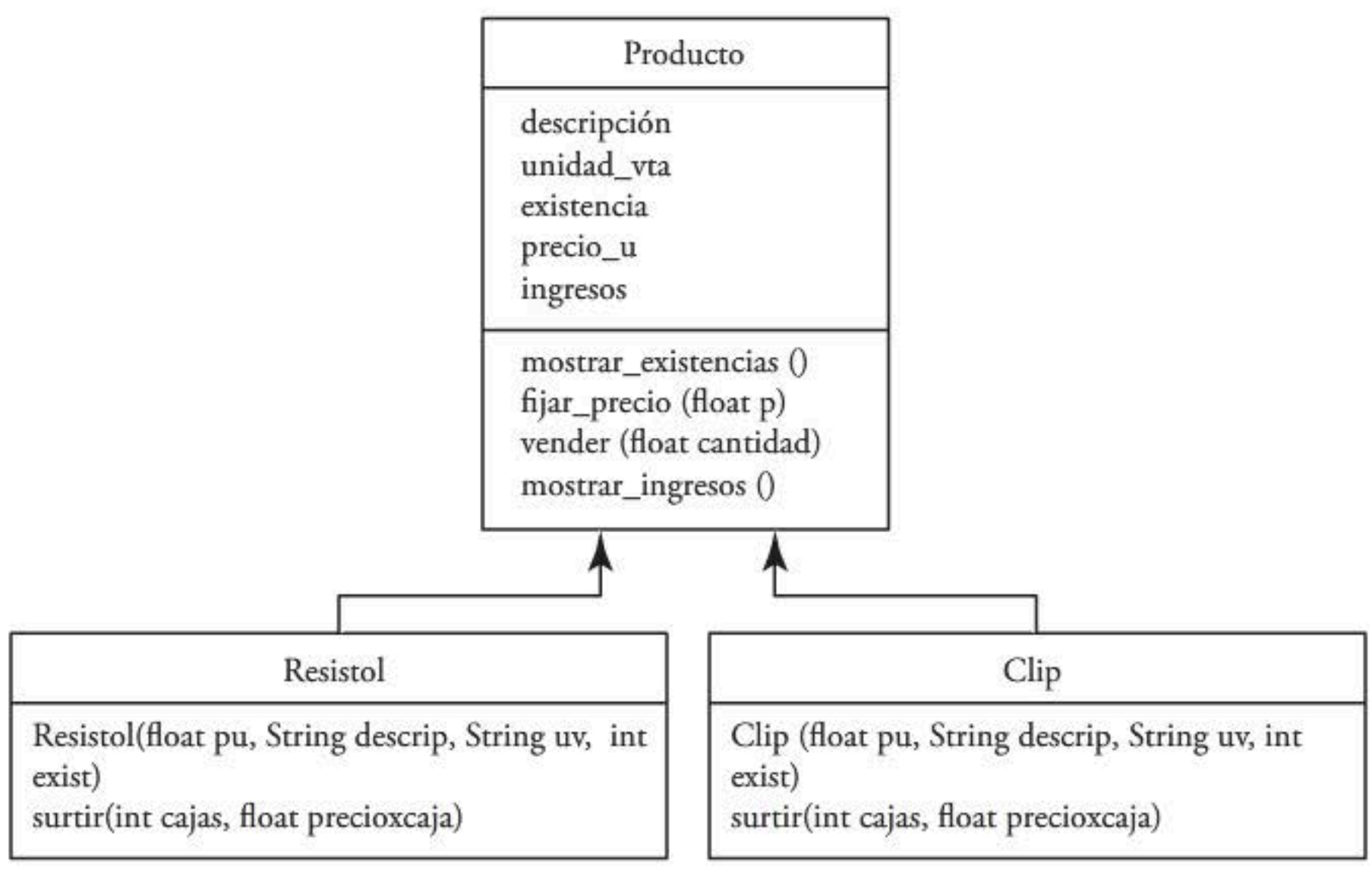
```
run:
DIMENSIONES Matriz1:3X2, Matriz2:2X4, Matriz3:3X4
Matriz 1
1  3
8  2
6  2

Matriz 2
3  2  4  3
4  7  5  5
```

(continúa en la siguiente página)

```
Matriz 3
15  23  19  18
32  30  42  34
26  26  34  28
GENERACIÓN CORRECTA (total time: 5 seconds)
```

(continúa de la página 276)

13. Utilizando herencia, elabore un programa con el paradigma orientado a objetos, considerando el siguiente esquema:



Los atributos y métodos de la clase padre (Producto) deben ser `protected` para que puedan ser visibles desde la clase derivada, pero no desde cualquier contexto. Los atributos descripción y unidad_vta deben ser de tipo `String`, existencia debe ser de tipo `int`, precio_u y ingresos deben ser de tipo `float`.

Los métodos deben funcionar de la siguiente manera:

- mostrar_existencias(): simplemente muestra la cantidad de artículos existentes en inventario.
- fijar_precio(**`float`** p): fija el precio unitario del producto utilizando el valor del parámetro p.
- vender(**`float`** cantidad): vende (saca de inventario) la cantidad de productos indicados por el parámetro *cantidad,* siempre y cuando haya las existencias correspondientes. Al hacer la venta los ingresos aumentan tomando en cuenta el precio unitario.
- mostrar_ingresos(): muestra la cantidad actual del producto e indica cuántos ingresos se han generado.

Para las clases Resistol y Clip, derivadas de la clase Producto considere lo siguiente:

Se debe tener un constructor con los parámetros necesarios para inicializar los atributos heredados (los parámetros corresponderían al precio unitario, descripción, unidad de venta y existencias).

Un método surtir, el cual debe llevar como parámetros las cajas que se surten y el precio de cada caja. Las sentencias de este método en cada clase derivada son diferentes porque en el caso de la clase Resistol se surten por cajas, pero las existencias son por cada tubo de resistol, considerando que cada caja lleva diez tubos. En el caso de la clase Clip se surten por cajas y se venden igual por cajas. En ambas clases derivadas debe verificarse si los ingresos alcanzan para hacer el surtido solicitado, si es así el método debe devolver verdadero(**true**), de lo contrario debe devolver falso (**false**).

A continuación se muestra la clase principal, la cual utilizará las clases que se deben crear.

```
public class Ventas {
public static void main(String[] args) {

/*Se crea un objeto de la clase Resistol, llamado pritt, con los
argumentos indicados */
     Resistol  pritt=new  Resistol(19.5f,"resistol  pritt  de
250ml","pieza",30) ;

// muestra existencias e ingresos
   pritt.mostrar_ingresos();

// Se establece el precio unitario
   pritt.fijar_precio(13.5f);

   pritt.vender(29);          // Se venden 29 unidades
   pritt.mostrar_ingresos();

/*Se intentan surtir 3 cajas de producto con un costo de $120 c/u
*/
  if(pritt.surtir(3, 120)==true)
    System.out.println("Se han surtido con éxito las cajas de"+ "
resistol");
else
  System.out.println("Error al surtir cajas de resistol");
 pritt.mostrar_ingresos();
 pritt.mostrar_existencias();

/* Se crea un objeto de la clase Clip, llamado cuadrito, con los
argumentos indicados */
   Clip cuadrito=new Clip(15.8f,"Clip metalico tamanio 2", "caja",
45);

// muestra existencias e ingresos
   cuadrito.mostrar_ingresos();
```

```
// Se establece el precio unitario
   cuadrito.fijar_precio(25.5f);

   cuadrito.vender(29);          // Se venden 29 unidades
   cuadrito.mostrar_ingresos();

/*Se intentan surtir 60 cajas de producto con un costo de $20.9
c/u */
  if(cuadrito.surtir(60, 20.9f)==true)
      System.out.println("Se han surtido con éxito las cajas de
Clip");
else
  System.out.println("Error al surtir cajas de Clip");
cuadrito.mostrar_ingresos();
cuadrito.mostrar_existencias();
}
}
```

```
run:
Hay 30 pieza resistol pritt de 250ml en inventario y un total de $0.0 en ingresos
Hay 1 pieza resistol pritt de 250ml en inventario y un total de $391.5 en ingresos
Se han surtido con éxito las cajas de resistol
Hay 31 pieza resistol pritt de 250ml en inventario y un total de $31.5 en ingresos
Hay 31 pieza resistol pritt de 250ml en existencia
Hay 45 caja Clip metalico tamanio 2 en inventario y un total de $0.0 en ingresos
Hay 16 caja Clip metalico tamanio 2 en inventario y un total de $739.5 en ingresos
No hay suficientes ingresos para surtir 60 caja de Clip metalico tamanio 2
Error al surtir cajas de Clip
Hay 16 caja Clip metalico tamanio 2 en inventario y un total de $739.5 en ingresos
Hay 16 caja Clip metalico tamanio 2 en existencia

GENERACIÓN CORRECTA (total time: 2 seconds)
```

14. En una empresa existen dos tipos de empleados: por horas o por comisión. Sin importar el tipo de trabajador, la empresa guarda la información de nombre, antigüedad y categoría. Sin embargo a los trabajadores por comisión se les paga un 10% de las ventas que realicen y a los trabajadores por horas se les paga a $25.75 cada hora acumulada de trabajo. Elabore un programa utilizando herencia, de manera que tenga una clase base (padre) llamada Empleado con los elementos comunes a los trabajadores y dos clases derivadas(hijas): Emp_comi con los elementos específicos para los trabajadores por comisión y Emp_horas con los elementos específicos para los trabajadores por horas. A continuación se muestra la clase principal que hará uso de la implementación que se hará:

```
//clase principal, donde inicia la ejecución
public class Pago {
public static void main(String[] args) {

/*Se crea el objeto trabajador1, de la clase Empleados por
comisión(Emp_comision) */
Emp_comi trabajador1=new Emp_comi(5,"Policarpo Alva", 'C');

//El trabajador 1 realiza 2 ventas
  trabajador1.vender(2000.89f);
  trabajador1.vender(1000);

// se muestra la información del trabajador1
   trabajador1.mostrar_info();

// Calcula e imprime el pago del trabajador1
   System.out.printf("Su pago es de  $%.2f \n", trabajador1.calc_
pago());

/* Se crea el objeto trabajador2, de la clase Empleados por horas
(Emp_horas) */
    Emp_horas  trabajador2=new Emp_horas(8,"Simplicio  Bejarano",
'B', 250,'N');

// El trabajador2 acumula horas trabajadas en 3 ocasiones
   trabajador2.acum_horas(5);
   trabajador2.acum_horas(10);
   trabajador2.acum_horas(3);

// se muestra la información del trabajador 2
trabajador2.mostrar_info();

/* Calcula e imprime el pago del trabajador2, con la primera ver-
sion de calc_pago() */
   System.out.printf("Su pago es de  $%.2f \n", trabajador2.calc_
pago());

/* Calcula e imprime el pago del trabajador2, con la segunda ver-
sion de calc_pago() */
    System.out.printf("Su pago sería de  $%.2f n", trabajador2.
calc_pago('A'));

/*Se crea el objeto trabajador3, de la clase Empleados por comisión
(Emp_comision) */
Emp_comi trabajador3=new Emp_comi(3,"Hortencia Ruiz", 'C');
```

```
// El trabajador3 realiza una venta de 5000.25
   trabajador3.vender(5000.25f);

// Se muestra la información del trabajador3
   trabajador3.mostrar_info();

// Calcula e imprime el pago del trabajador3
   System.out.printf("Su pago es de  $%.2f \n", trabajador3.calc_
pago());
 }
}// fin de la clase principal
```

```
run:
Nombre: Policarpo Alva, Antiguedad 5 años, Tipo de empleado: por comisión
Por alcanzar ventas de $3000.8901 Su pago es de  $300.09
Nombre: Simplicio Bejarano, Antiguedad 8 años, Tipo de empleado: por horas
Pago como empleado de base, con un salario base de 250.0 y 18 trabajadas, Su pago es de  $803.50
Pago como empleado sin base, solo las 18 trabajadas, Su pago sería de  $463.50
Nombre: Hortencia Ruiz, Antiguedad 3 años, Tipo de empleado: por comisión
Por alcanzar ventas de $5000.25 Su pago es de  $500.02

GENERACIÓN CORRECTA (total time: 2 seconds)
```

15. Elabore una clase sobrecargando el método sumar con 3 versiones, la primera versión devuelve la suma de 3 valores enteros (los 3 valores son los parámetros de la función), la segunda versión es sin parámetros y devuelve la suma de varios valores enteros dados por el usuario (todos mientras no sea cero), la tercera versión devuelve la concatenación de 4 parámetros de tipo `String`. Elabore también la clase principal que hará uso de la clase que contendrá la sobrecarga del método sumar.

```
run:
Versión para varios valores enteros
Dame un valor entero o cero para terminar: 9
Dame un valor entero o cero para terminar: -6
Dame un valor entero o cero para terminar: 0
La suma fue 3
Versión para 3 valores enteros, Se sumaron los valores 23, 45, 56 la suma fue 124
Versión para concatenar, Se sumaron los valores hola , soy , tu , amigo la suma fue hola soy tu amigo

GENERACIÓN CORRECTA (total time: 10 seconds)
```

16. Escribir un programa que contenga:

a) Una clase llamada "*Versiones_mayor*" en donde se tengan dos métodos sobrecargados, uno de ellos que reciba dos cantidades enteras y que encuentre el mayor de dichas cantidades. El otro método que reciba un conjunto de n números enteros y que encuentre el mayor de dicho conjunto. El nombre del método sobrecargado es "*mayor*".

b) Otra clase llamada "*El_mayor*" en donde esté contenido el programa principal, que permita el llamado a los métodos sobrecargados de la clase anterior "Versiones_mayor".

Salidas típicas se muestra a continuación:

```
Cuántos números son: 2
Dame los 2 números:
8
34
El mayor es =34
```

```
Cuántos números son: 4
Dame los 4 números:
13
−7
48
26
El mayor es =48
```

17. Escribir un programa que contenga:

a) Una clase llamada "*operaciones*" en donde se tenga un método sobrecargado llamado "operación" con seis versiones diferentes, de acuerdo a las siguientes características:

- Uno de ellos no recibe parámetros y solamente imprime la palabra "hola".
- El segundo recibe un solo parámetro e imprime solamente ese parámetro.
- El tercer método recibe dos parámetros, los suma, e imprime el resultado de dicha suma.
- El cuarto método recibe tres parámetros, suma los dos primeros, imprime el resultado y después le resta al resultado de la suma el tercer parámetro, e imprime el resultado.
- El quinto de los métodos recibe cuatro parámetros, suma los dos primeros, resta el segundo y multiplica dicho resultado por el cuarto de los parámetros.
- El sexto de los métodos recibe cinco parámetros, suma los dos primeros, resta el tercero, multiplica por el cuarto parámetro y finalmente divide el resultado entre el quinto de los parámetros.

Los parámetros que se mandan a los diferentes métodos son enteros. En caso de recibir parámetros lo primero que imprimen cada uno de los métodos son los propios parámetros que recibió y si se realiza alguna de las operaciones (suma, resta, multiplicación, división) se deberá imprimir el resultado después de llevar a cabo dicha operación aritmética.

a) Deberá contener además una clase llamada "*sobrecarga_operaciones*" que contiene el programa principal y que llama los distintos métodos sobrecargados enviándoles diferentes valores enteros como parámetros:

Una salida típica se muestra a continuación, en donde cada línea es la impresión de cada uno de los métodos a los cuales se les envió los parámetros que están al principio de la línea, a excepción de la línea con la palabra "Hola" cuyo método no recibió ningún parámetro.

```
12 34 40 6 Suma=46 Resta=6 multiplica=36
52 8 63 Suma=60 Resta=−3
15
60 3 71 14 20 Suma=63 Resta=−8 multiplica=−112 Divide=−5.60
Hola
28 17 Suma=45
53 5 13 9 16 Suma=58 Resta=45 multiplica=405 Divide=25.31
```

18. Escribir un programa que contenga:

a) Una clase llamada "*ordenalo*" que tiene un método sobrecargado llamado "*burbuja*" con dos versiones diferentes, una de ellas que permita ordenar un arreglo de números enteros en forma ascendente y la otra versión para ordenar un arreglo de nombres en forma alfabética usando para ello el sort de la burbuja.

b) Otra clase llamada "*Sort*" que contiene el programa principal y dos métodos sobrecargados, uno de ellos llamado "*leerlos*" para leer el conjunto de números o bien el conjunto de nombres almacenándolos en un arreglo. Otro método sobrecargado llamado "*imprime*" para imprimir el conjunto de nombres o bien el conjunto de números según corresponda.

Una salida típica se muestra a continuación:

```
Es un conjunto de:
[ 1 ] Números
[ 2 ] Nombres
Opción: 1
Cuantos datos son: 3
Dame los 3 datos:
47
23
58
Conjunto de datos ordenados:
23
47
58
```

```
Es un conjunto de:
[ 1 ] Números
[ 2 ] Nombres
Opción: 2
Cuantos nombres son: 4
Dame los 4 nombres:
María
Alicia
Pablo
Daniel
Conjunto de nombres ordenados:
Alicia
Daniel
María
Pablo
```

19. Escribir un programa que contenga:

Una clase llamada "*diferente*" que tiene un método sobrecargado llamado "*encuentra*" con cuatro versiones diferentes de acuerdo a lo siguiente:

- La primera versión del método sobrecargado podrá recibir un número flotante, ese valor es el radio del círculo que le permitirá encontrar el área de esa figura geométrica.

- Si solamente se recibe un número entero, entonces se deberá elevar al cubo dicha cantidad.
- Si recibe una cadena, deberá encontrar el número de letras vocales de dicha cadena.
- Si recibe una cadena y un número entero positivo y el número es menor que la longitud de la cadena, deberá extraer esa cantidad de caracteres de la cadena. En caso de que el número sea mayor que la longitud de la cadena, se deberá invertir la cadena.

En todos los casos se deberá imprimir la información recibida, así como el resultado de la operación correspondiente.

b) Deberá tener también otra clase que contenga el programa principal, desde donde se llama al método sobrecargado enviando la información respectiva.

Si la información que reciben los métodos es:

- "el león no es como lo pintan"
- 9
- "sistemas computacionales", 82
- 5.34
- "dime con quién andas y te diré quién eres", 17

La salida que corresponde es:

```
el león no es como lo pintan  TIENE 10 vocales
9 Al cubo=  729
sistemas computacionales  INVERTIDA= selanoicatupmoc sametsis
5.34  área = 89.58460896
dime con quién andas y te diré quién eres  SUBCADENA= dime con
```

INTRODUCCIÓN A LAS INTERFACES GRÁFICAS DE USUARIO

7

Antes de que un software sea reutilizable debería ser utilizable.

Ralph Johnson

Competencia de la unidad

Construir programas sencillos utilizando interfaces gráficas en Java.

- Conocer los conceptos centrales de las Interfaces Gráficas de Usuario.
- Aplicar los conceptos centrales de las Interfaces Gráficas de Usuario en el diseño y construcción de programas.

Control de flujo

Contenido

7.1. Introducción.
7.2. Definición y evolución de las Interfaces Gráficas.
7.3. Proceso de desarrollo de *software* con Interfaces Gráficas.
7.4. Paquetes gráficos.
7.5. Cuadros de diálogo.
7.6. Etiquetas.
7.7. Campos de texto.
7.8. Botones.
7.9. Resumen.
7.10. Un vistazo a JTextArea y JScrollPane.
7.11. Problemas.

7.1 Introducción

En los capítulos anteriores se aprendió a construir programas que funcionaban introduciendo y mostrando información mediante la consola del entorno de desarrollo NetBeans IDE. En este capítulo, se mejorará la apariencia y se facilitará el uso de los programas, porque se aprenderá a crear interfaces gráficas.

En Java existen muchos elementos gráficos; sin embargo, este capítulo tiene como objetivo introducir a la programación de interfaces gráficas, por lo que no se revisará con mucha profundidad la información que existe, pero con lo explicado se obtendrá la lógica básica de programación de interfaces, brindando así la oportunidad de entender con mayor facilidad y rapidez cualquier biblioteca gráfica en Java que se consulte más adelante.

Primeramente se explicará qué es una interfaz gráfica, cómo han evolucionado y cómo es que debe desarrollarse un programa que incluya una interfaz; para después explicar cómo programar interfaces en Java que hagan uso de cuadros de diálogo, etiquetas, campos de texto y botones.

7.2 Definición y evolución de las interfaces gráficas

Una GUI (*Graphical User Interface*, Interfaz Gráfica de Usuario), es un programa computacional que por medio de un conjunto de elementos gráficos, permite introducir y visualizar información. Esto hace que el *software* sea "amigable"; es decir, que sea fácil de utilizar y más agradable a la vista.

Las GUIs surgieron ante la necesidad de hacer más accesible a la gente el uso de la computadora, porque en un inicio sólo las personas con conocimientos técnicos podían manipularlas.

Algunos de los hechos importantes durante la evolución de las GUIs se resumen en la línea de tiempo de la Figura 6.1.

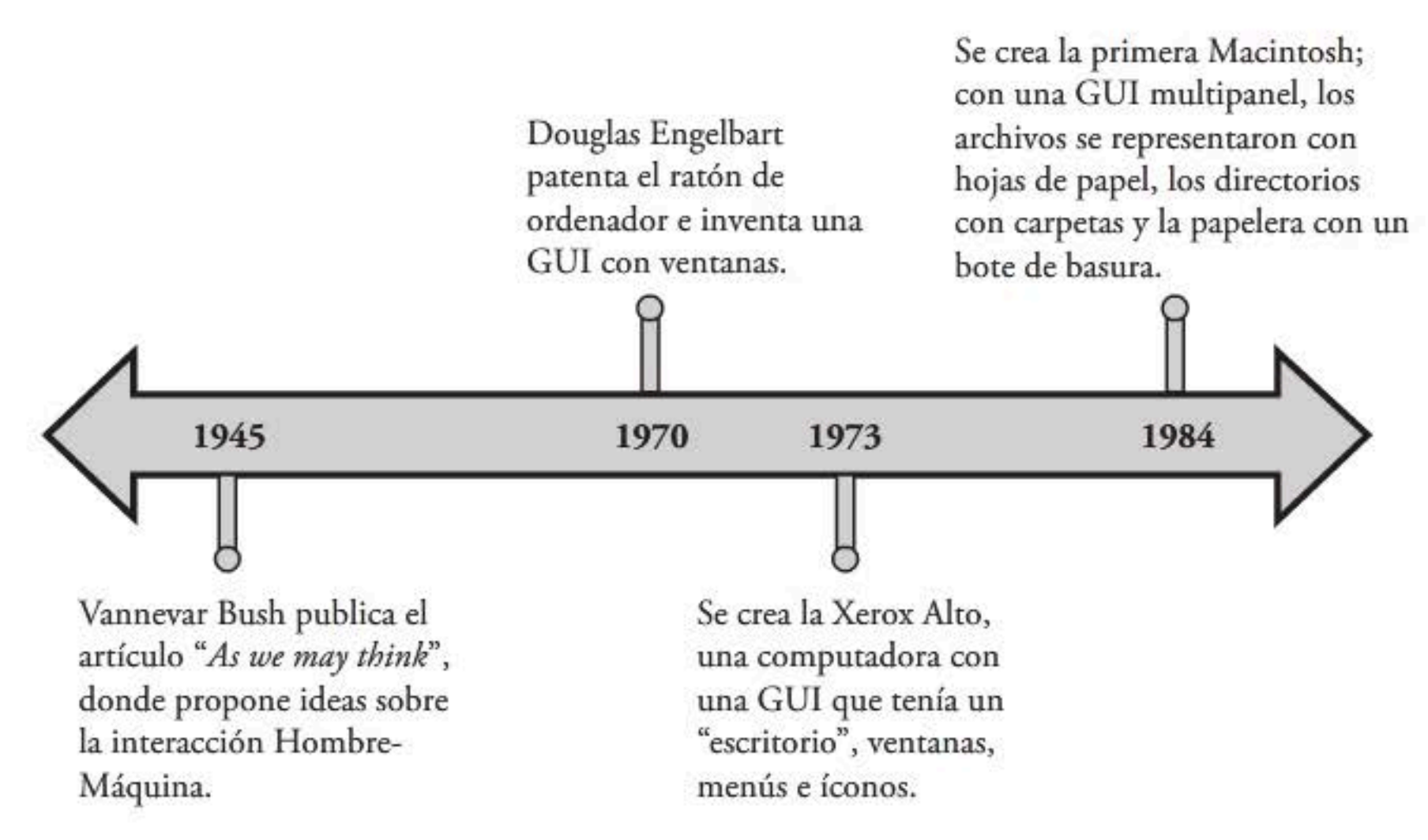


Figura 7.1. Evolución de las GUIs.

En 1945, Vanner Bush publicó el artículo "*As we may think*" (traducido en español: "como podríamos pensar"), en el cual planteaba varias ideas interesantes acerca de la interacción Hombre-Máquina; como el crear un dispositivo que permitiera realizar cálculos, que funcionara mediante el dictado y el hipertexto. La máquina se quedó solamente en un proyecto (porque nunca se construyó), se llamó "Memex" y era un dispositivo mecánico de almacenamiento de libros, grabaciones y películas, de búsqueda muy sencilla, rápida y no lineal.

En 1970, Douglas Engelbart patentó el ratón de ordenador e inventó la primera GUI basándose en las ideas de Vanner Bush. El mouse fue descrito como "un indicador de posición en el eje de coordenadas X-Y para un sistema de visualización", cuyo mecanismo consistía de una caja de madera con ruedas que movía un cursor en la pantalla. Engelbart nunca recibió ganancias por su invento porque la patente expiró en 1987, antes de que las computadoras personales utilizaran de manera indespensable el mouse.

En 1973, la compañía estadounidense Xerox desarrolló la primera computadora personal con una GUI llamada "Xerox Alto", esta computadora no fue un producto comercial pero se utilizó dentro de las oficinas de Xerox y en algunas universidades de ese país. La GUI de "Xerox Alto" contaba con ventanas, íconos y menús para realizar las acciones de abrir, borrar y mover archivos. Fue la primera en utilizar la metáfora de escritorio para ubicar las aplicaciones de trabajo.

Para 1984, se lanza al mercado la primera Macintosh creada por Steve Jobs y Jef Raskin, ésta fue la primera computadora comercial en tener una GUI de ventanas multipanel. Además presentó algunos elementos que se conservan a la fecha en los sistemas operativos de Mac; por ejemplo, los archivos se representaron por medio de íconos que tenían el aspecto de hojas de papel, los directorios se representaron por medio de carpetas, las cuales podían borrarse si se arrastraban a un ícono con la forma de un cesto de basura (papelera). También contaba con algunas aplicaciones (que también se conservan hasta ahora) como una calculadora, un reloj y un bloc de notas.

A medida que las GUIs han evolucionado, se ha contribuido en facilitar la manipulación de muchos dispositivos como computadoras, tablets y celulares; haciendo accesible su uso a diversos sectores de la población como los niños y personas mayores.

7.3 Proceso de desarrollo de *software* con interfaces gráficas

El desarrollo de *software* es una actividad no trivial y es de suma importancia conocer cuáles son los pasos mínimos necesarios para construir programas que incluyan interfaces gráficas, véase Figura 6.2.

Figura 7.2. Pasos para construir un programa con interfaz gráfica.

A continuación se describe cada fase:

1. Entender el problema a resolver.

En este paso, se deben obtener y comprender las especificaciones del cliente/desarrollador con las cuáles deberá cumplir el *software* a desarrollar.

Cuando se trabaja desarrollando *software*, esta fase se conoce como *obtención de requerimientos* y es muy compleja; porque no es sencillo entender lo que el cliente realmente desea. Muchas veces, no se habla en los mismo términos (por ejemplo, puede ser que el cliente sea un biólogo o un carpintero y no se manejen los tecnicismos); los clientes no tienen el tiempo para asistir a varias reuniones con el equipo de desarrollo y por lo tanto los desarrolladores no pueden obtener la información necesaria; los desarrolladores de *software* no quieren invertir tanto tiempo en entender al cliente y lo que desean es irse a programar cuanto antes; los desarrolladores de *software* no propician un ambiente de confianza para que el cliente pueda expresarse libremente, entre otras razones.

El no realizar un buen trabajo en esta fase ha sido la causa de muchos fracasos y pérdidas monetarias en muchos proyectos de *software*, porque tal como lo muestra la analogía del columpio con el *software* de la Figura 6.3, existe mucha diferencia entre lo que el cliente deseaba, lo que el desarrollador entendió y le entregó.

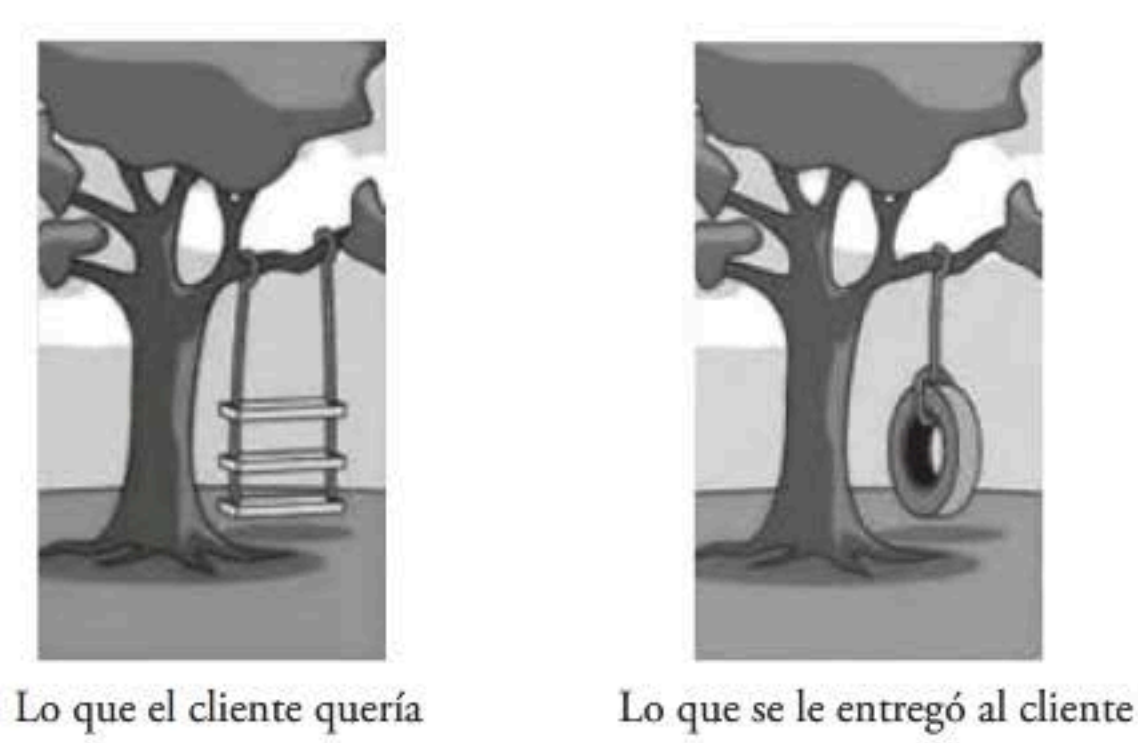


Figura 7.3. Analogía del desarrollo de *software*.

Por lo que, un buen desarrollo de *software* va encaminado a entregar un producto de calidad: que satisfaga las expectativas del cliente, cumpliendo con el presupuesto y tiempo acordado.

2. Diseñar la interfaz gráfica.

Es recomendable esbozar en hojas de papel la(s) pantalla(s) de la interfaz gráfica de usuario. Ésta deberá diseñarse tomando en cuenta los requerimientos obtenidos en la fase anterior.

Una buena práctica consiste en mostrar los diagramas de la interfaz al cliente antes de continuar con las demás fases del proceso de desarrollo de *software*; esto permite aclarar malos entendidos antes de codificar.

En la Figura 7.4 se muestra un ejemplo de cómo podrían realizarse los dibujos de las pantallas. En ellos se puede apreciar que se hace uso de círculos con números para indicar hacia dónde direcciona cada botón. Por ejemplo, si el usuario oprime en la "Interfaz de Inicio" el botón de "Sumar", el círculo indica que se presentaría la "Interfaz de Sumas" porque existe una correspondencia con el número 1.

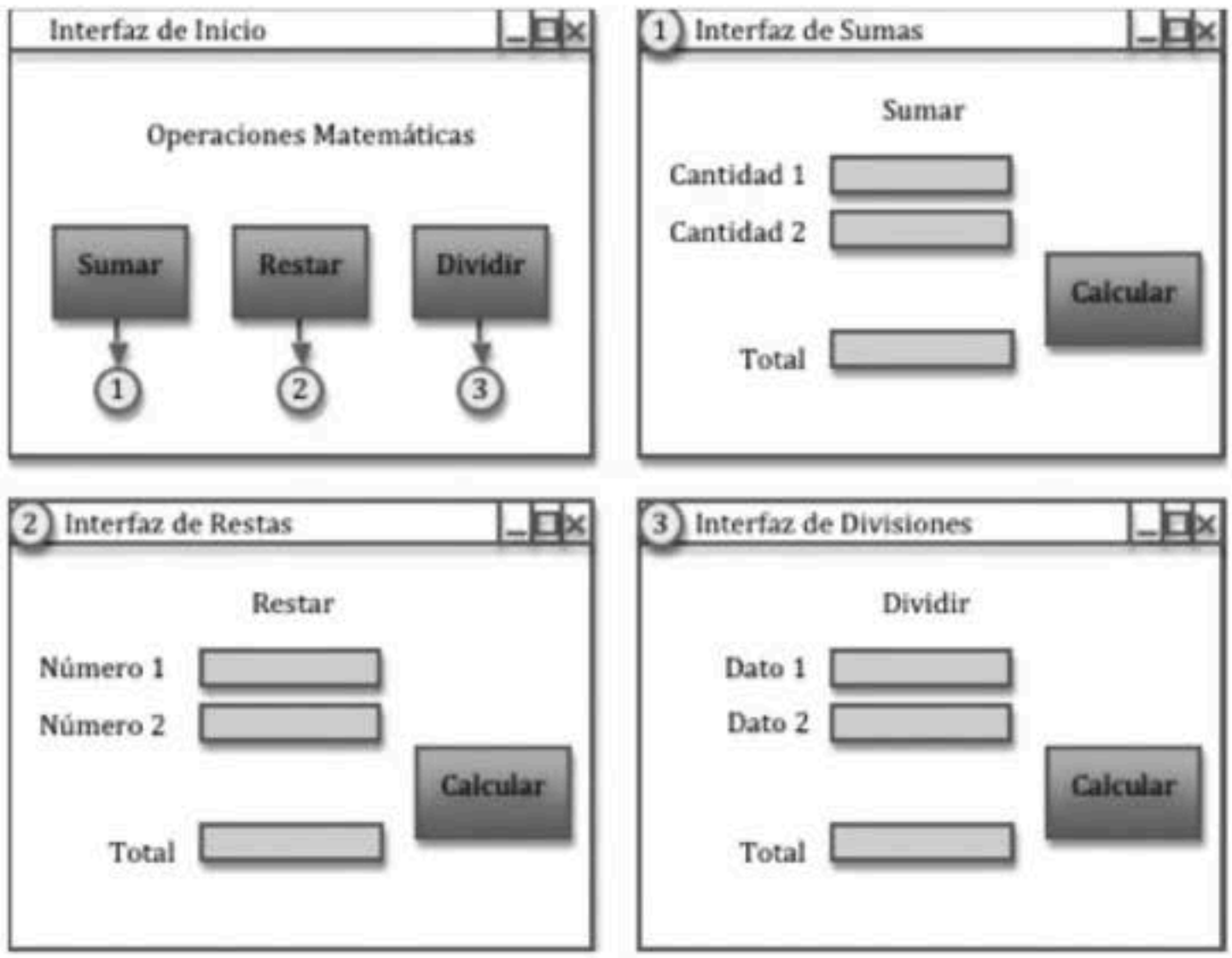


Figura 7.4. Ejemplo de esbozo de las pantallas de la GUI.

3. Diseñar la solución funcional.

En esta fase se diseña la parte funcional del *software*. Esto puede realizarse a través de Diagramas de Flujo, Diagramas N-S o Pseudocódigo.

Tanto el diseño de la interfaz gráfica como el de la solución funcional, se incluyen dentro de las fases conocidas como *Análisis y Diseño*. En las empresas de desarrollo de *software*, cuando se llega a esta etapa, los analistas y arquitectos de software se reúnen para plantear una solución al problema (una arquitectura), la cual servirá de guía durante la construcción.

El plantear la arquitectura de un sistema implica seleccionar la tecnología a usarse, el sistema operativo, el lenguaje de programación, entre otras cosas; además de considerar los posibles riesgos y problemas que podrían presentarse.

4. Codificar

Esta fase consiste en programar la solución planteada en las etapas anteriores.

Es aconsejable incluir comentarios, fechas, nombres de autores y versiones para documentar el código; de esta manera, en un futuro será más fácil comprender lo que se realizó y poder hacer modificaciones o darle mantenimiento a dicho *software*.

5. Realizar pruebas

Una vez que se tiene un programa, es necesario realizar pruebas para corroborar su buen funcionamiento.

Actualmente, los desarrollos de *software* no esperan a que se termine de construir el *software* para probarlo, sino que se realizan pruebas constantes; lo cual es una buena práctica.

Respecto a las pruebas, se puede decir que una buena prueba es aquella que detecta un *defecto* generado por un *error* (comúnmente introducido al momento de la codificación). Esto se hace para evitar *fallas* al momento de que el usuario haga uso del *software*.

Existen dos tipos de pruebas: de caja blanca y de caja negra.

Las pruebas de caja blanca se realizan sobre el código. Una de las pruebas comunes consiste en probar con valores límite las variables de los ciclos.

En cambio, las pruebas de caja negra se hacen sin ver el código. Estas se realizan sobre el *software* en ejecución. Una de las pruebas comunes consiste en "estresar al sistema"; es decir, introducir tantos datos como sean posibles o en introducir tipos de datos no esperados.

7.4 Paquetes gráficos

En Java existen dos paquetes gráficos: AWT (*Abstract Window Toolkit*, Conjunto de herramientas de Ventanas Abstractas) y Swing, los cuales forman parte de las JFC (*Java Foundation Classes*, Clases Fundamentales de Java).

AWT y Swing están compuestos por clases que sirven para crear interfaces gráficas y se diferencian por lo siguiente:

- AWT tiene componentes pesados que cada plataforma (sistema operativo) representa de manera determinada; por lo que no es portable. Muchas de las clases son obsoletas y ya no se pueden utilizar (en NetBeans IDE aparecerá el error en caso de presentarse esta situación).
- Swing tiene componentes ligeros que pueden tomar diferente aspecto y comportamiento dependiendo de la plataforma donde se ejecute un programa. Está escrito totalmente en Java y es el paquete más utilizado.

Para hacer uso de los paquetes sólo es necesario importarlos:

```
//Para utilizar todas las clases de Swing
  import javax.swing.*;

//Para utilizar todas las clases de AWT
  import java.awt.*;
```

En los ejemplos que se observarán más adelante, se mostrará que no es necesario importar todas las clases de los paquetes gráficos (para lo cual se utiliza un asterisco). Porque una vez que se adquiere experiencia en el uso de los paquetes gráficos, se puede seleccionar una clase específica para evitar cargar todas las clases en la memoria de la computadora.

7.5 Cuadros de diálogo

Los cuadros de diálogo son ventanas que permiten la interacción entre el usuario y el programa.

Existen dos tipos: de entrada y de salida. Para utilizarlos es necesario importar la clase `JOptionPane` del paquete `Swing`.

```
import javax.swing.JOptionPane;
```

Los *cuadros de diálogos de salida*, permiten mostrar información. El método que se llama de la clase `JOptionPane` es `showMessageDialog`, el cual está sobrecargado (existen varias versiones del

método con el mismo nombre pero se diferencian por el número de parámetros que pueden recibir); brindado así la opción de elegir el método más conveniente dependiendo de lo que se desee programar, los dos más utilizados son:

```
JOptionPane.showMessageDialog (componente, cadena_mensaje);
JOptionPane.showMessageDialog(componente,cadena_mensaje,
cadena_titulo, ícono);
```

En el parámetro que corresponde al **componente**, se le pondrá el valor de **null**, porque en este capítulo al ser introductorio a las GUIs no se hablará acerca de éstos. Para el parámetro **cadena_mensaje** se colocará la cadena que se desea mostrar como mensaje; en **cadena_título** se enviará la cadena que se desea que aparezca en la barra de título de la ventana del cuadro de diálogo; finalmente, en el **ícono**, se elegirá una constante de `JOptionPane` de la Tabla 7.1.

La sintaxis para indicar qué ícono se utlizará es:

`JOptionPane.`**Nombre_Constante_Icono**

Por ejemplo:

`JOptionPane.WARNING_MESSAGE`

Tabla 7.1. Tipos de íconos para cuadros de diálogo

Nombre de la constante	Ícono
QUESTION_MESSAGE	
WARNING_MESSAGE	
INFORMATION_MESSAGE	
ERROR_MESSAGE	
PLAIN_MESSAGE	Sin ícono

Ejemplo 7.1. En el siguiente programa con su respectiva imagen de la ejecución, se puede observar la implementación de un cuadro de diálogo de salida que hace uso del método `showMessageDialog` que recibe sólo dos parámetros: componente y cadena_mensaje.

```
import javax.swing.JOptionPane;

public class CuadroDialogoSalida_1 {
public static void main (String args []){
   JOptionPane.showMessageDialog(null, "Bienvenido a las interfa-
ces");
   }
}
```

Ejemplo 7.2. En el siguiente programa con su respectiva imagen de la ejecución, se puede observar la implementación de un cuadro de diálogo de salida que hace uso del método `showMessageDialog` que recibe cuatro parámetros: componente, cadena_mensaje, cadena_título e ícono.

```
import javax.swing.JOptionPane;

public class CuadroDialogoSalida_2 {
    public static void main (String args []){

JOptionPane.showMessageDialog(null, "Los programas con GUI son más
usables", "Ejemplo Interfaz 2", JOptionPane.INFORMATION_MESSAGE);

  }
}
```

Los *cuadros de diálogos de entrada*, permiten recibir información introducida por el usuario. Los datos recibidos son de tipo **`String`**; por lo que, en muchos casos será necesario hacer uso de lo visto en la Tabla 3.11 del tema "3.10 Conversión de Tipos de Datos" de este libro.

Para los cuadros de diálogos de entrada también se hace uso de la clase `JOptionPane` y de manera específica del método `showInputDialog`, el cual también está sobrecargado. Los dos métodos más utilizados son:

String cadenaRecibida = JOptionPane.showInputDialog (**cadena_mensaje**, **cadena_por_defecto**);
String cadenaRecibida = JOptionPane.showInputDialog(**componente**, **cadena_mensaje**, **cadena_título**, **ícono**);

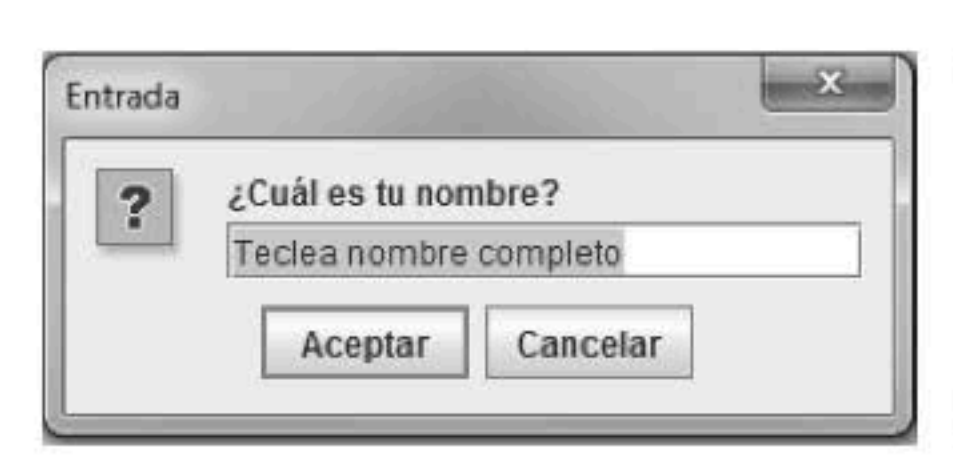
En el primer caso, en la **cadena_mensaje** se escribe la cadena que servirá de mensaje y en **cadena_por_defecto** una cadena que podría servir de ejemplo/instrucción para ayudar al usuario a introducir datos en el campo de texto.

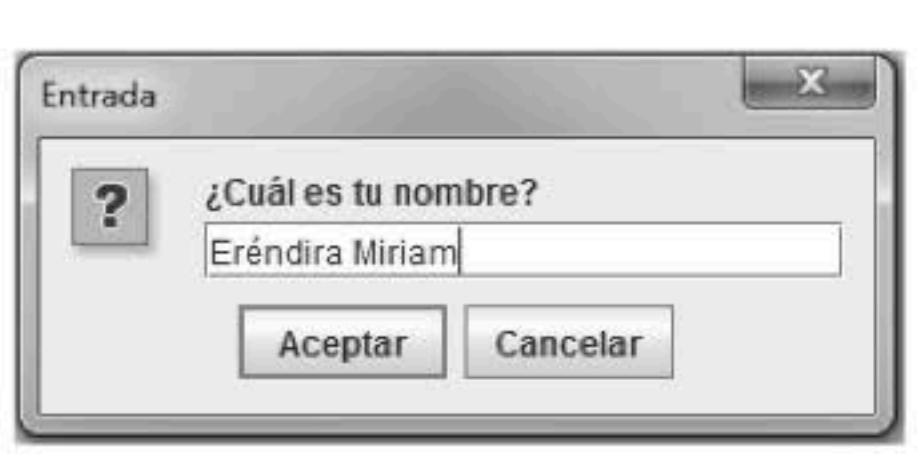
En el segundo caso, funciona al igual que en los cuadros de diálogos de salida; es decir, en el **componente** se pondrá **`null`**; en **cadena_mensaje** el mensaje principal de la ventana; en **cadena_título**, la leyenda que aparecerá en la barra de título de la ventana del cuadro de diálogo y en **ícono** también se seleccionará uno de la Tabla 7.1.

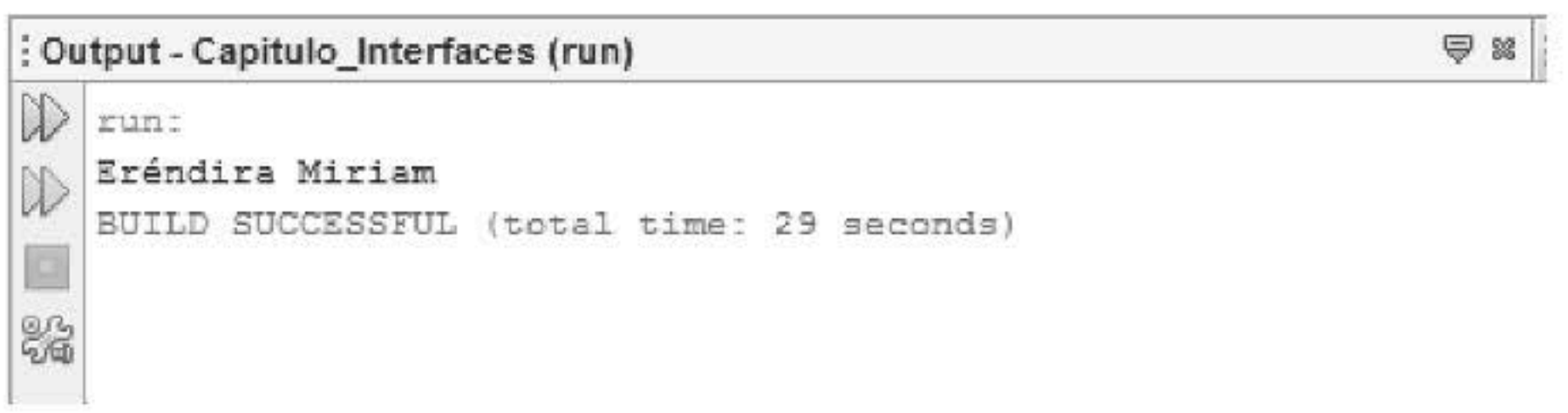
Y en ambos casos es necesario guardar el valor leído en una variable de tipo **`String`** (en este caso **`cadenaRecibida`**); o bien, en otro tipo de variable considerando que debe convertirse a ese tipo porque siempre se lee todo como **`String`**.

Ejemplo 7.3. En el siguiente programa con sus respectivas imagenes de la ejecución, se puede observar la implementación del primer caso de los cuadros de diálogo de entrada. Es decir, en la ventana aparece el mensaje "¿Cuál es tu nombre?" y en el campo de texto aparece por defecto la leyenda "Teclea nombre completo". Una vez que el usuario da clic sobre el campo de texto, puede teclear el nombre y al oprimir la tecla "enter" o clic en el botón "Aceptar"; para después mostrar en consola el nombre recibido.

```
import javax.swing.JOptionPane;

public class CuadroDialogoEntrada_1 {
    public static void main (String args []){

String nombre = JOptionPane.showInputDialog("¿Cuál es tu nombre?",
"Teclea nombre completo");

      System.out.println(nombre);
   }
}
```

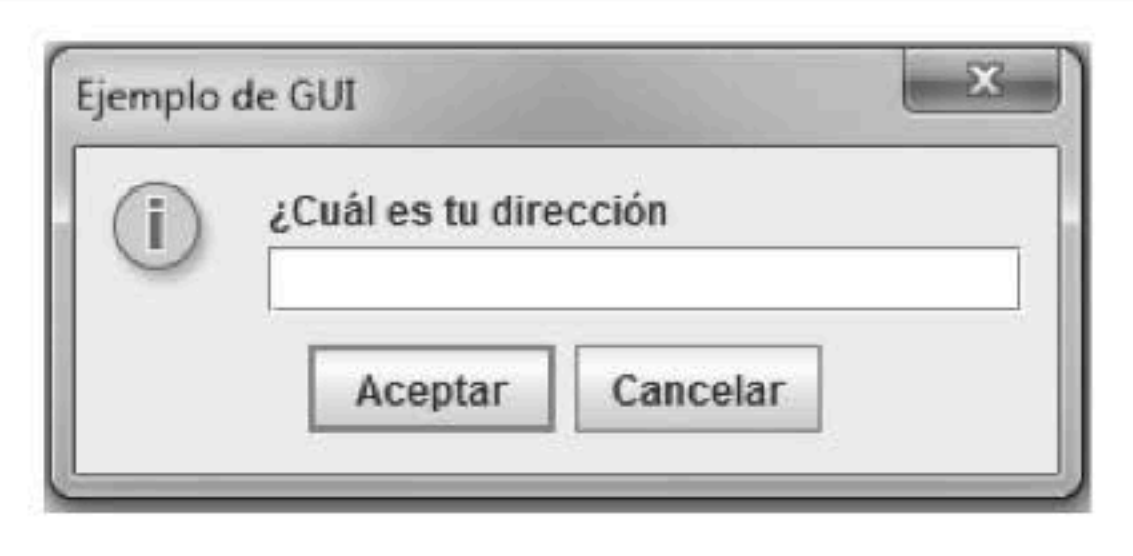


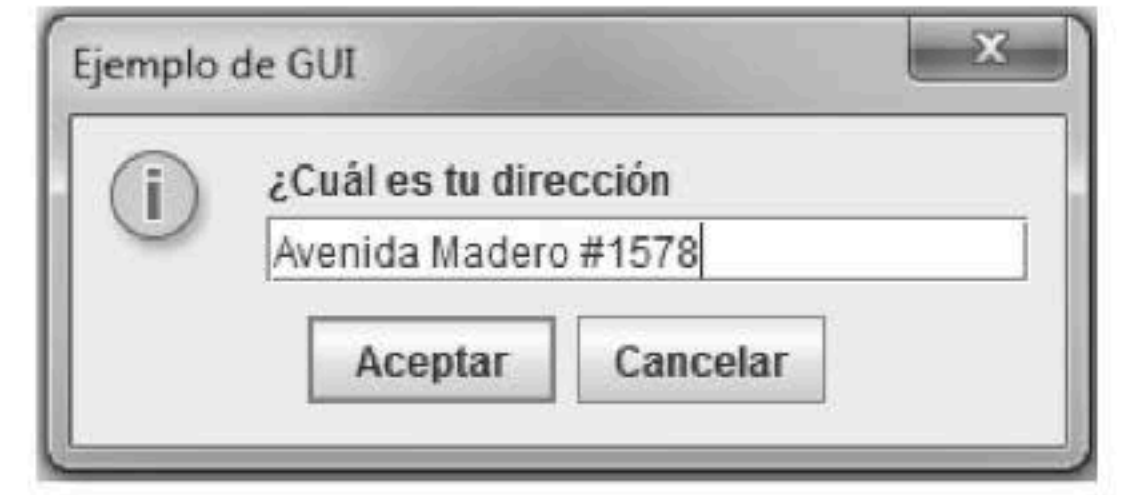




Ejemplo 7.4. En el siguiente programa con sus respectivas imagenes de la ejecución, se puede observar la implementación del segundo caso de los cuadros de diálogo de entrada. Es decir, en la ventana aparece el mensaje "¿Cuál es tu dirección?", el título de dicha ventana es "Ejemplo de GUI" y el tipo de ícono es `INFORMATION_MESSAGE`. Una vez que el usuario introduce la dirección, ésta se muestra en un cuadro de diálogo de salida.

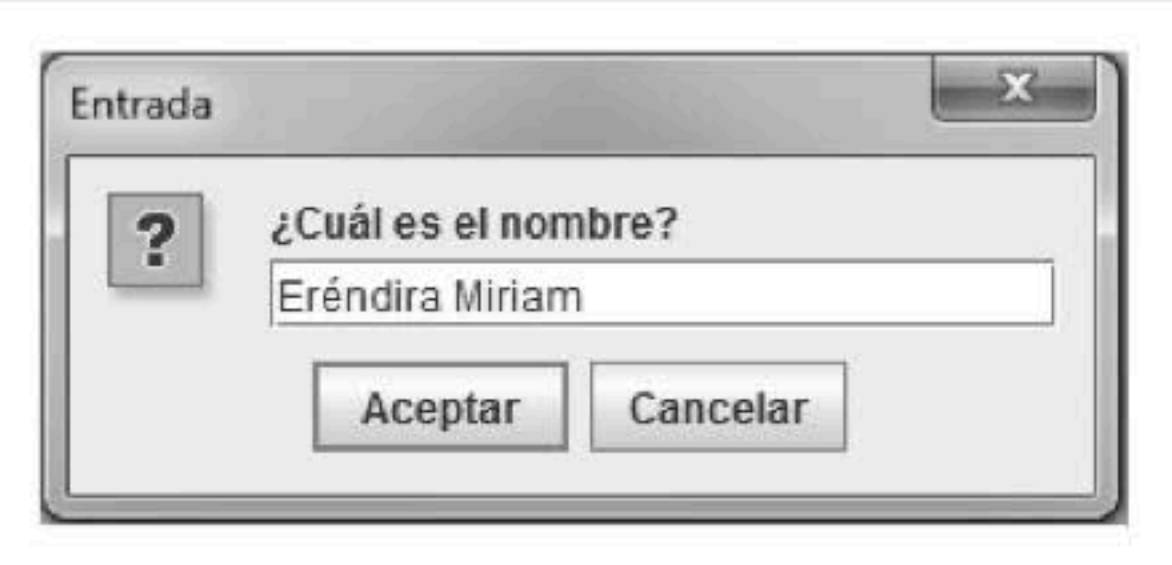
```
import javax.swing.JOptionPane;

public class CuadroDialogoEntrada_2 {
public static void main (String args []){

String direccion = JOptionPane.showInputDialog(null, "¿Cuál es tu
dirección", "Ejemplo de GUI", JOptionPane.INFORMATION_MESSAGE);

     JOptionPane.showMessageDialog(null, direccion);
   }
}
```

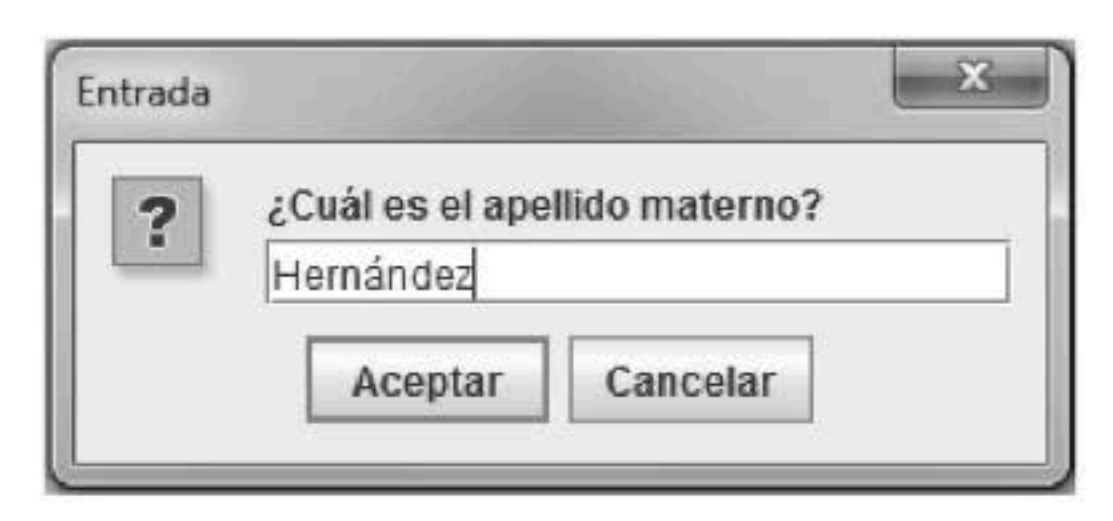


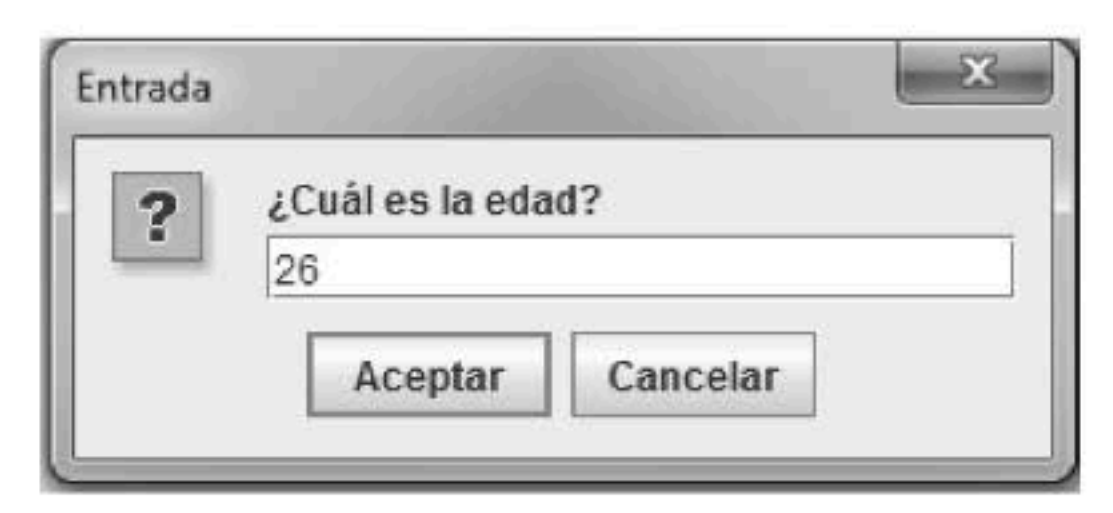


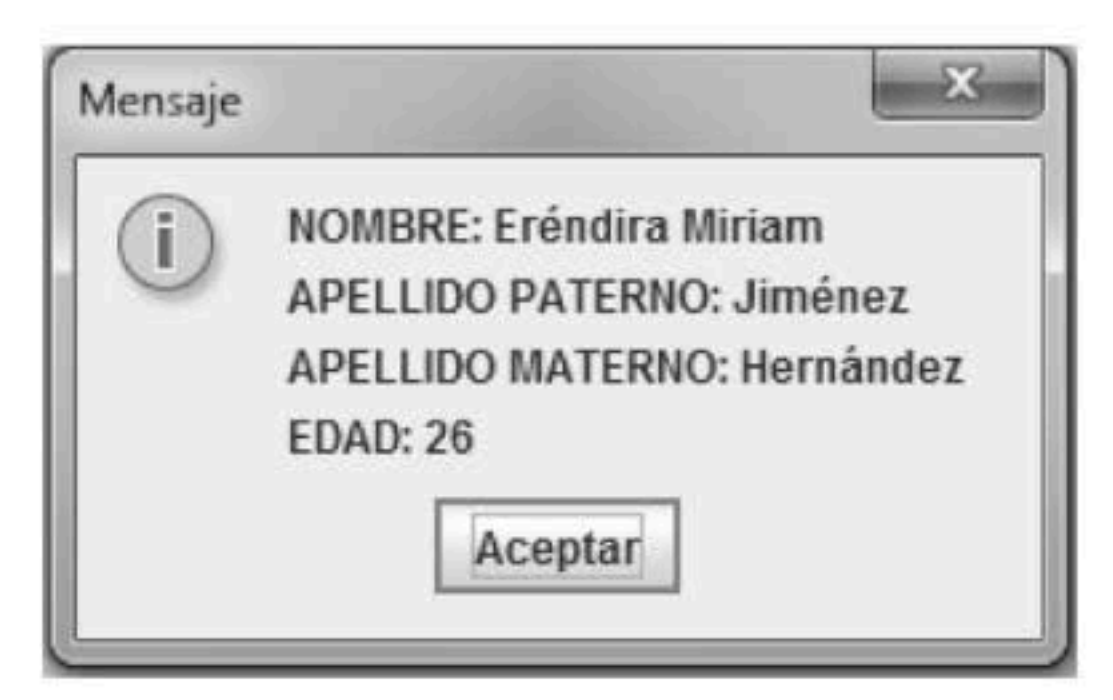
Ejemplo 7.5. En el siguiente programa se utilizan 4 cuadros de diálogo de entrada para solicitar al usuario su nombre, apellido paterno, apellido materno y edad. Los datos se van concantenando en una cadena para mostrarla en un cuadro de diálogo de salida.

```
import javax.swing.*;
public class datosPersona {
public static void main (String args []){
   String cadena = "";

cadena = cadena.concat("NOMBRE: "+
JOptionPane.showInputDialog( null, "¿Cuál es el nombre?"));

cadena = cadena.concat("\nAPELLIDO PATERNO: "+
JOptionPane.showInputDialog(null, "¿Cuál es el apellido pater-
no?"));

cadena = cadena.concat("\nAPELLIDO MATERNO: "+
JOptionPane.showInputDialog(null, "¿Cuál es el apellido mater-
no?"));

cadena = cadena.concat("\nEDAD: "+
JOptionPane.showInputDialog(null,  "¿Cuál es la edad?"));

    JOptionPane.showMessageDialog(null, cadena);
   }
}
```





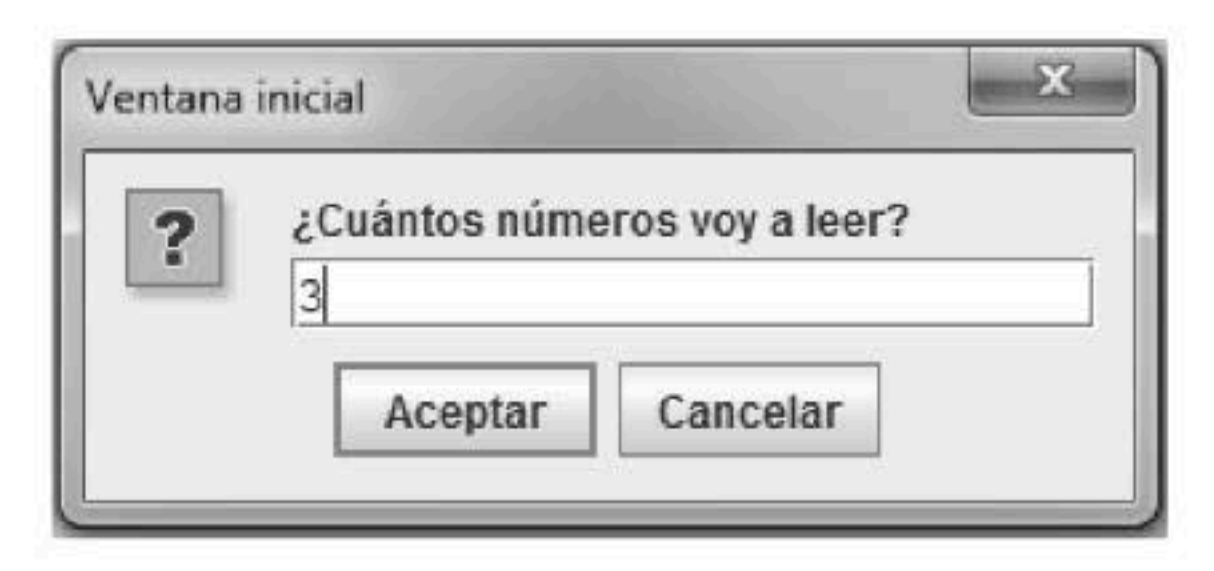


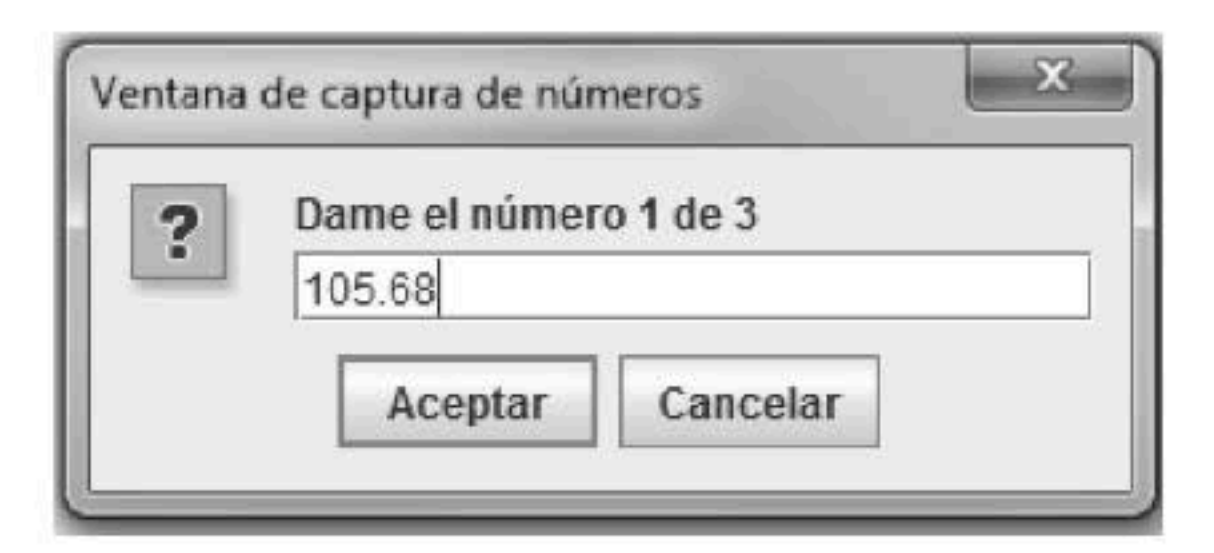


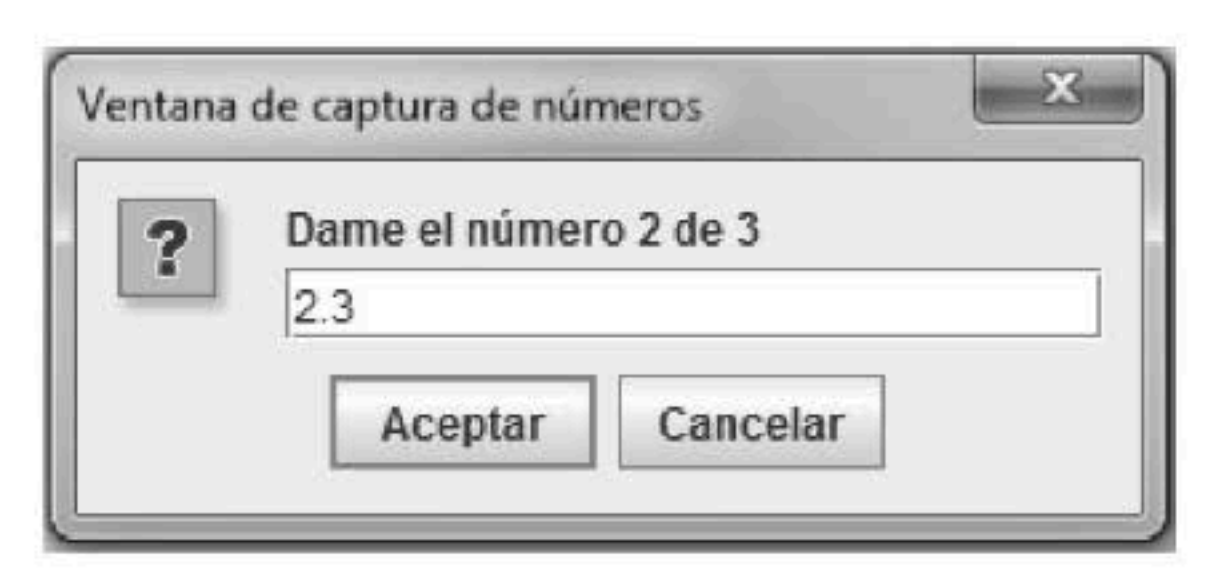
Ejemplo 7.6. En el siguiente programa se utiliza un cuadro de diálogo de entrada para pedirle al usuario que introduzca el número de números a leer para calcular el promedio de ellos. Por lo que, aparecerán n cuadros de diálogo de entrada para solicitar los números y finalmente mostrar en un cuadro de diálogo de salida el resultados del cálculo. Es importante notar que en este ejemplo se convierten las cadenas recibidas a números de tipo **double**.

```
import javax.swing.*;

public class promedio_grafico {
public static void main (String args []){

double suma = 0;
```

```
int n = Integer.parseInt(JOptionPane.showInputDialog(null, "¿Cuán-
tos números voy a leer?", "Ventana inicial", JOptionPane.QUESTION_
MESSAGE));

for (int i=1; i<=n; i++)
suma = suma + Double.parseDouble( JOptionPane.showInputDialog(null,
"Dame el número "+i+" de "+n,"Ventana de captura de números", JOp-
tionPane.QUESTION_MESSAGE));

JOptionPane.showMessageDialog(null, "El promedio de los números es
" + (suma/n),"Ventana de resultado", JOptionPane.INFORMATION_MES-
SAGE);
  }
}
```







7.6 Etiquetas

Las etiquetas son elementos gráficos que permiten mostrar texto no editable en una interfaz gráfica.

Para utilizar una etiqueta es necesario crear un objeto de la clase `JLabel`, la cual pertenece al paquete `swing`. Por lo que, primero es necesario importar el paquete `swing`:

```
import java.swing.*;
```

Para después crear el objeto de alguna de las dos maneras:

1. Declarando e inicializando el objeto en la misma línea.

```
JLabel nombre_etiqueta = new JLabel (cadena_etiqueta);
```

2. Declarando en una línea el objeto y después inicializándolo en otra línea.

```
JLabel nombre_etiqueta;
nombre_etiqueta = new JLabel (cadena_etiqueta);
```

Es importante saber que cuando se utilizan elementos gráficos (como las etiquetas, cuadros de texto o botones), éstos deben estar incluidos en un panel (`JFrame`). Esto sería como incluirlos en una ventana porque por sí solos no pueden aparecer de manera aislada en una pantalla.

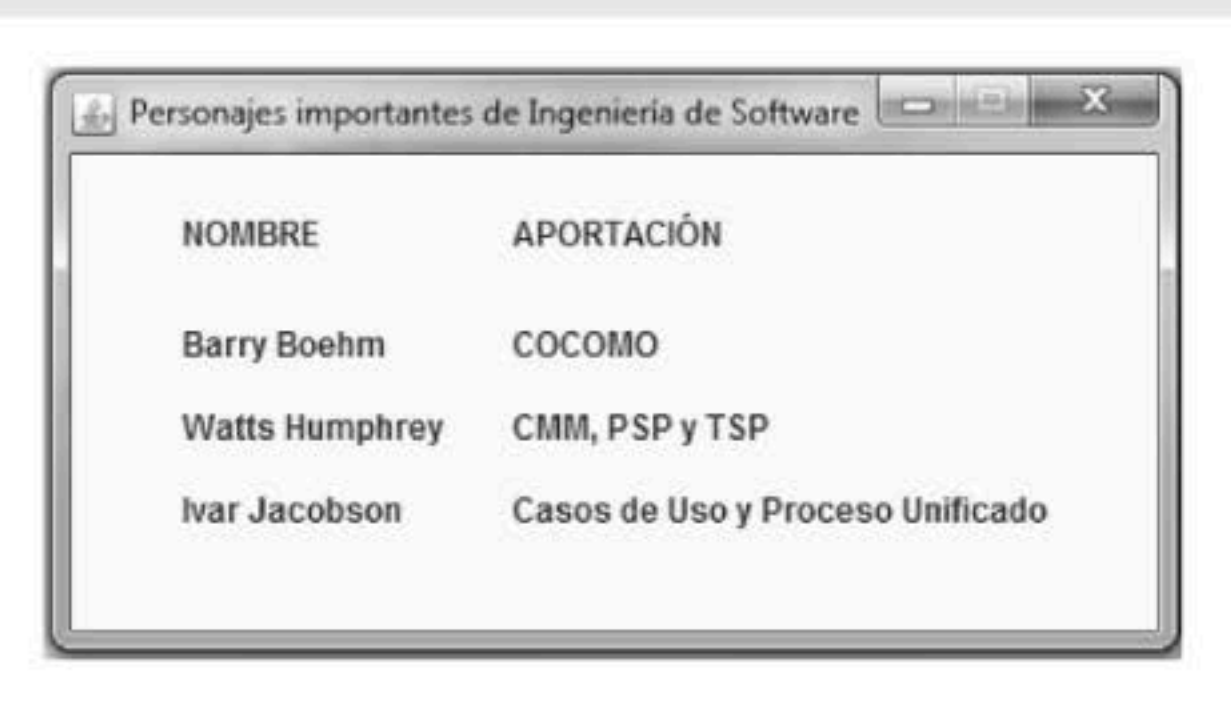
Una vez que se declara un objeto del tipo `JFrame`, se requiere utilizar un administrador de elementos gráfico, el cual permite indicar la forma en la cual se distribuirán o aparecerán los elementos gráficos al momento de incluirse en el panel.

Existen varios tipos de administradores gráficos, pero uno de los más eficientes es el `Absolute Layout`. Para utilizarlo es necesario incluir la biblioteca a la cual pertenece (véase Apéndice 3).

Ejemplo 7.7. En el siguiente programa, se puede observar cómo se deben declarar e inicializar las etiquetas para poder realizar una interfaz con ellas.

```
import javax.swing.*;
/* se incluyen las siguientes clases para utilizar el administrador
de elementos gráfico: Absolute Layout Recuerde agregar la librería
'Absolute Layout', con clic derecho en Librerías del proyecto */
import org.netbeans.lib.awtextra.AbsoluteConstraints;
import org.netbeans.lib.awtextra.AbsoluteLayout;

public class Etiquetas_1{
  public Etiquetas_1(){

/ *Se declara el objeto para crear un panel y se inicializa con la
cadena "Personales importantes de Ingeniería de Software", la cual
aparecerá en la barra de título de la ventana */
JFrame panel = new JFrame ("Personajes importantes de Ingeniería
de Software");

//se declaran e inicializan varias etiquetas
  JLabel nombres;
  nombres = new JLabel ("NOMBRE");
  JLabel aportaciones;
  aportaciones = new JLabel ("APORTACIÓN");

    JLabel nombre1 = new JLabel ("Barry Boehm");
    JLabel aportacion1 = new JLabel ("COCOMO");
    JLabel nombre2 = new JLabel ("Watts Humphrey");
    JLabel aportacion2 = new JLabel ("CMM, PSP y TSP");
    JLabel nombre3 = new JLabel ("Ivar Jacobson");
JLabel aportacion3 = new JLabel ("Casos de Uso y Proceso Unifica-
do");

/* se especifica que el administrador de elementos gráficos que
utilizará el panel será Absolute Layout */
   panel.setLayout(new AbsoluteLayout());

/* se especifica la ubicación de cada etiqueta en el panel
indicando las coordenadas en X y Y */
 panel.add(nombres, new AbsoluteConstraints(40,20));
 panel.add(nombre1, new AbsoluteConstraints(40,60));
 panel.add(nombre2, new AbsoluteConstraints(40,90));
 panel.add(nombre3, new AbsoluteConstraints(40,120));
```

```
 panel.add(aportaciones, new AbsoluteConstraints(160,20));
 panel.add(aportacion1, new AbsoluteConstraints(160,60));
 panel.add(aportacion2, new AbsoluteConstraints(160,90));
 panel.add(aportacion3, new AbsoluteConstraints(160,120));

//se especifica el tamaño de lo ancho y lo alto del panel
  panel.setSize(400,200);

//se indica que el panel será visible
  panel.setVisible(true);

/* Se indica que no podrá redefinirse el tamaño del panel.
Así que aunque el usuario intente modificar el tamaño de la ventana
al posicionar el cursor en una esquina, ésta conservará su tamaño
especificado siempre. */
   panel.setResizable (false);

/* se indica que al dar clic en el botón de cerrar la ventana se
terminará la ejecución */
    panel.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }

/* se declara la clase principal, se crea un objeto de la clase
Etiquetas_1 para poder crear la interfaz */
   public static void main(String[] args) {
        Etiquetas_1 objeto = new Etiquetas_1();
    }
}
```



7.7 Cuadros de texto

Los cuadros de texto son elementos gráficos que permiten al usuario ingresar información a través del teclado.

Los cuadros de texto funcionan de manera similar a las etiquetas, es decir se requiere importar el paquete `swing`:

```
import java.swing.*;
```

Adicionalmente deben importarse las clases `ActionEvent` y `ActionListener` para manejar los eventos de los cuadros de texto (los eventos son las acciones asociadas a los cuadros de texto):

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

En el caso de los cuadros de texto, cuando se escribe algo dentro de ellos y se oprime la tecla "enter", éste evento puede originar que se realicen ciertas acciones, como leer lo que el usuario introdujo para realizar determinados cálculos. Por eso se importan los paquetes anteriores.

Al igual que con las etiquetas, es necesario crear el objeto, en este caso de la clase `JtextField`, lo cual se puede hacer:

1- Declarando e inicializando el objeto en la misma línea.

```
JtextField nombre_campo1 = new JtextField (numero_columnas);
JtextField nombre_campo2 = new JtextField (cadena_predefinida);
```

2. Declarando en una línea el objeto y después inicializándolo en otra línea.

```
JtextField nombre_campo1;
nombre_campo1 5 new JtextField (numero_columnas);

JtextField nombre_campo2;
nombre_campo2 5 new JtextField (cadena_predefinida);
```

Si se decide crear un campo de texto con el constructor que recibe numero_columnas, será necesario especificar el número de columnas que tendrá el cuadro del campo de texto.

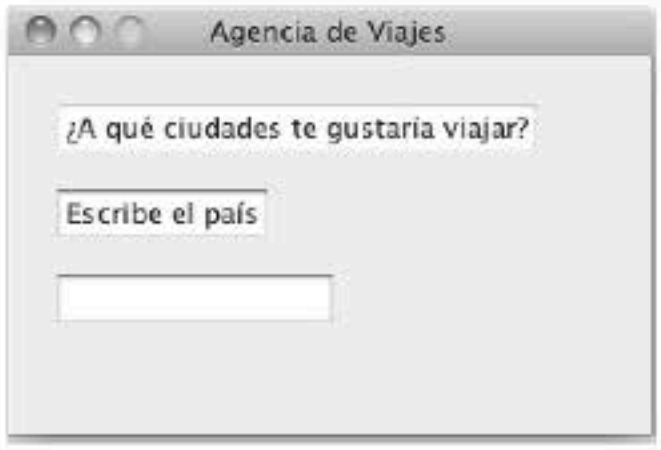
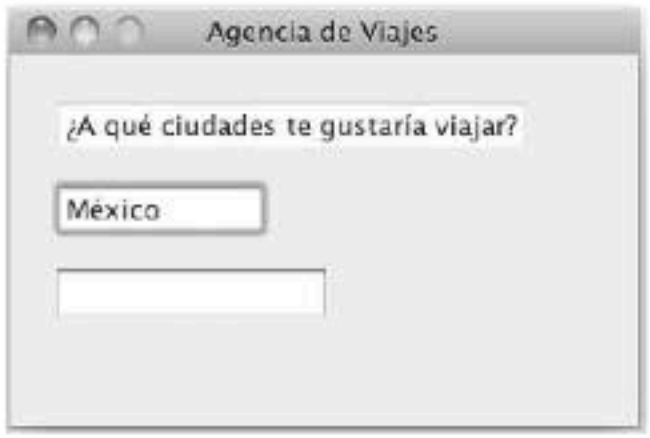
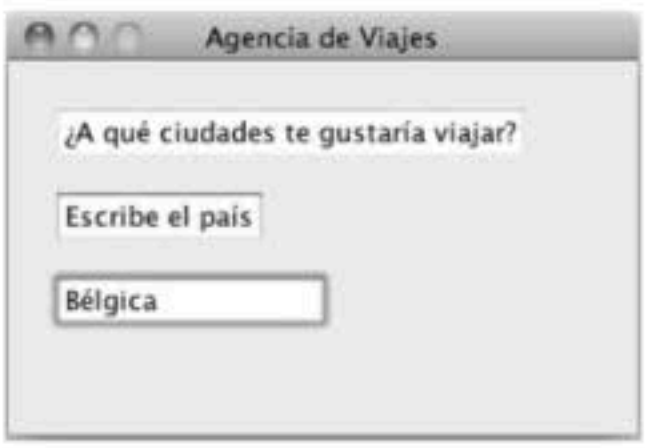
Y si se crea un campo de texto con una cadena_predefinida, deberá indicarse la cadena que aparecerá de manera inicial antes de que el usuario se posicione sobre el campo de texto y dé "enter".

Ejemplo 7.8. En el siguiente programa, se puede observar cómo se deben declarar e inicializar los cuadros de texto para poder realizar una interfaz con ellos. En este ejemplo se puede apreciar cómo manejar los eventos de dichos cuadros de texto.

```
/*se importa la clase ActionEvent y ActionListener de awt para po-
der manejar los eventos */
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
import org.netbeans.lib.awtextra.AbsoluteConstraints;
import org.netbeans.lib.awtextra.AbsoluteLayout;

public class CuadrosTexto_1 {
 public CuadrosTexto_1 (){

//se crea el panel que contendrá los cuadros de texto
  JFrame panel_2 = new JFrame ("Agencia de Viajes");
```

```
/* se declara e inicializa un cuadro de texto con la leyenda entre
paréntesis */
   JTextField campoNoEditable = new JTextField ("¿A qué ciudades
te gustaría viajar?");

//se especificará que no se puede editar
  campoNoEditable.setEditable(false);

/*se declara e inicializa un cuadro de texto con la leyenda entre
paréntesis se indica que su especificador de acceso será final para
poder manejar sus eventos */
  final JTextField campoTextoPredefinido = new JTextField ("Es-
cribe el país");

/*se declara e inicializa un cuadro de texto con un tamaño de 10
columnas se especifica que su especificar de acceso será final para
poder manejar sus eventos */
final JTextField campoTamanoDefinido = new JTextField (10);

/* se indica qué hará campoTextoPredefinido cuando el usuario dé
enter sobre él, para lo cual se maneja este evento */
   campoTextoPredefinido.addActionListener(
  new ActionListener(){
    public void actionPerformed(ActionEvent ae) {
      String entradaCampoTextoPredef;

/* se indica que se obtendrá lo escrito en el campo de texto y se
guardará en la cadena entradaCampoTextoPredef */
 entradaCampoTextoPredef = campoTextoPredefinido.getText();

/* se muestra en un cuadro de diálogo de salida lo que se introdujo */
   JOptionPane.showMessageDialog(null, entradaCampoTextoPredef);
   }
 }
);

/* se indica qué hará campoTamanoDefinido cuando el usuario dé en-
ter sobre él, para lo cual se maneja este evento */
   campoTamanoDefinido.addActionListener(
new ActionListener(){
   public void actionPerformed(ActionEvent ae) {
     String entradaCampoTamanoDefinido;

/* se indica que se obtendrá lo escrito en el campo de texto y se
guardará en la cadena entradaCampoTamanoDefinido*/
entradaCampoTamanoDefinido = campoTamanoDefinido.getText();
```

```
/* se muestra en un cuadro de diálogo de salida lo que se introdujo */
    JOptionPane.showMessageDialog(null, entradaCampoTamanoDefinido);
   }
  }
 );

/* se especifica que el administrador de elementos gráficos para panel_2 será AbsoluteLayout */
   panel_2.setLayout(new AbsoluteLayout());

/* se agrega campo por campo de texto al panel_2 con sus respectivas coordenadas */
   panel_2.add(campoNoEditable, new AbsoluteConstraints(20,20));
   panel_2.add(campoTextoPredefinido, new AbsoluteConstraints(20,60));
   panel_2.add(campoTamanoDefinido, new AbsoluteConstraints(20,100));

//se especifica el tamaño del panel_2
  panel_2.setSize(300,200);

//se indica que será visible el panel_2
panel_2.setVisible(true);

//se determina que el panel no podrá ser redimensionado
  panel_2.setResizable (false);

/* se indica que cuando se cierre el panel_2 se terminará la ejecución del programa */
panel_2.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

  public static void main(String[] args) {
/*se crea un objeto de la clase CuadrosTexto_1 por lo que automáticamente se llamará al constructor de esa clase */
  CuadrosTexto_1 objeto = new CuadrosTexto_1();
   }
}
```

Evento	Salida
Al ejecutarse el programa, se muestra la ventana principal, cuyo título es "Agencia de Viajes"	
Si se escribe algo sobre el campo de texto con el mensaje predefinido "Escribe el país" y se oprime la tecla "enter". En este caso, se escribió "México", por eso en el cuadro de diálogo de salida se mostró eso.	
Si se escribe algo sobre el campo de texto con el mensaje predefinido "Escribe el país" y se oprime la tecla "enter". En este caso, se escribió "México", por eso en el cuadro de diálogo de salida se mostró eso.	
Si se escribe algo sobre el campo de texto sin texto predefinido y con tamaño de 10 columnas. En este caso, se escribió Bélgica por lo que en el cuadro de diálogo de salida apareció ese país después de oprimir la tecla "enter"	

7.8 Botones

Los botones son elementos gráficos que permiten activar un evento para que se realicen ciertas acciones.

Para crear un botón en Java, es necesario importar el paquete `swing`, las clases `ActionEvent` y `ActionListener` para manejar sus eventos y crear el objeto de la clase `JButton`.

```
JButton calcular = new JButton(cadena_mensaje_boton);
```

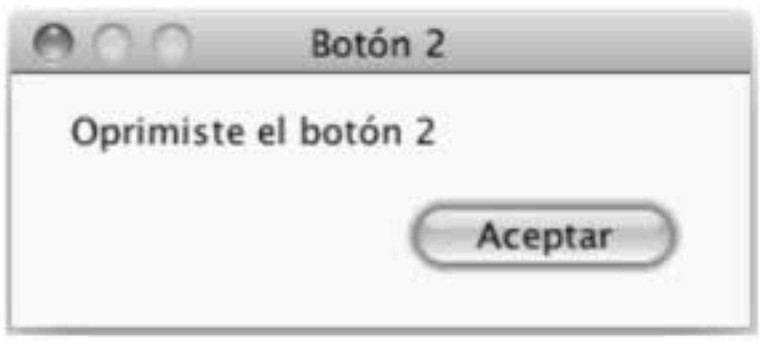
O bien:

```
JButton calcular;
calcular = new JButton(cadena_mensaje_boton);
```

Ejemplo 7.9. En el siguiente programa, se puede observar cómo se puede hacer uso de los botones en Java.

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import org.netbeans.lib.awtextra.AbsoluteConstraints;
import org.netbeans.lib.awtextra.AbsoluteLayout;

public class InterfazBotones {
    public InterfazBotones(){
        JFrame panel = new JFrame ("Botones");

        //se crean tres botones
        JButton botonUno = new JButton("Uno");
        JButton botonDos = new JButton("Dos");
        JButton botonTres;
        botonTres = new JButton("Tres");

        /*se especifica cómo se manejará el evento del
         *botonUno */
        botonUno.addActionListener(
         new ActionListener(){
          public void actionPerformed(ActionEvent ae) {
          JOptionPane.showMessageDialog(null, "Oprimiste el botón
1");
          }
         }
        );


        /*se especifica cómo se manejará el evento del
         *botonDos  */
```

```
        botonDos.addActionListener(
         new ActionListener(){
           public void actionPerformed(ActionEvent ae) {
JOptionPane.showMessageDialog(null, "Oprimiste el botón 2", "Botón
2", JOptionPane.PLAIN_MESSAGE);
           }
         }
        );

        /*se especifica cómo se manejará el evento del
         *botonTres */
        botonTres.addActionListener(
         new ActionListener(){
            public void actionPerformed(ActionEvent ae) {
                JOptionPane.showMessageDialog(null, "Oprimiste el
botón 3");
            }
         }
        );

        panel.setLayout(new AbsoluteLayout());
//se agregan los botones al panel
  panel.add(botonUno, new AbsoluteConstraints(50,25));
  panel.add(botonDos, new AbsoluteConstraints(50,75));
  panel.add(botonTres, new AbsoluteConstraints(50,125));

//se especifica el tamaño del panel
  panel.setSize(200,200);

//se indica que el panel será visible
  panel.setVisible(true);

//se indica que el panel no podrá redimensionarse
  panel.setResizable (false);

/*se indica que al cerrarse la ventana del panel se terminará la
ejecución del programa */
  panel.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 }

 public static void main(String[] args) {
 /*se crea un objeto de la clase InterfazBotones y automáticamente
se ejecuta lo que hay en el constructor de esa clase */
   InterfazBotones objeto_boton = new InterfazBotones();
    }

}
```

Evento	Salida
Al ejecutarse el programa, se muestra la ventana principal, cuyo título es "Botones"	
Si se oprime el botón con la leyenda "Uno"	
Si se oprime el botón con la leyenda "Dos"	
Si se oprime el botón con la leyenda "Tres"	

Ejemplo 7.10. En el siguiente programa, se muestra un ejemplo de cómo utilizar todos elementos gráficos vistos en este capítulo.

```
/*se importa el paquete swing para poder utilizar elementos grá-
ficos */
import javax.swing.*;
```

```
/*se importan las clases ActionEvent y ActionListener para manejar
los eventos */
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/*se importa la clase AbsoluteConstraints y AbsoluteLayout para
poder utilizar el administrador de elementos gráficos Abosulte La-
yout */
import org.netbeans.lib.awtextra.AbsoluteConstraints;
import org.netbeans.lib.awtextra.AbsoluteLayout;

public class Tienda{

  public Tienda(){

/*Se crea un objeto de la clase JFrame, éste será el panel que
contendrá todos los elementos gráficos de la interfaz. El título
de la ventana es "Calculadora para Tienda" */
  JFrame panel = new JFrame ("Calculadora para Tienda");

/*Se crean las etiquetas que aparecerán del lado izquierdo en la
ventana */
  JLabel productos = new JLabel ("Productos");

  JLabel producto1 = new JLabel ("Refrescos: ");
  JLabel producto2 = new JLabel ("Papas: ");
  JLabel producto3 = new JLabel ("Paletas: ");
  JLabel producto4 = new JLabel ("Agua: ");
  JLabel producto5 = new JLabel ("Pan: ");

/*se crea la etiqueta y los campos de texto que aparecerán en me-
dio de la ventana. Los cuadros de texto se inicializan con "0" */
  JLabel cantidades = new JLabel ("Cantidad");

  final JTextField campo1 = new JTextField (3);
  campo1.setText("0");
  final JTextField campo2 = new JTextField (3);
  campo2.setText("0");
  final JTextField campo3 = new JTextField (3);
  campo3.setText("0");
  final JTextField campo4 = new JTextField (3);
  campo4.setText("0");
  final JTextField campo5 = new JTextField (3);
  campo5.setText("0");
```

```
/*Se crean las etiquetas que aparecerán del lado derecho en la
ventana */
  JLabel precios = new JLabel ("Precios");

  JLabel precio1 = new JLabel ("X $5.50");
  JLabel precio2 = new JLabel ("X $8.00");
  JLabel precio3 = new JLabel ("X $1.30");
  JLabel precio4 = new JLabel ("X $6.00");
  JLabel precio5 = new JLabel ("X $15.50");

//se crea la etiqueta "TOTAL"
  JLabel total = new JLabel ("TOTAL");

/*se crea un campo de texto donde se mostrará el resultado del
cálculo, por eso se inhabilita */
  final JTextField pago = new JTextField (5);
  pago.setEnabled(false);

//se crea el botón "Calcular"
  JButton calcular = new JButton("Calcular");

/*se especifica que se manejarán los eventos del botón "Calcular".
Es decir, al dar clic sobre él, se obtendrán los valores introdu-
cidos en los campos de texto, para convertirlos en datos de tipo
int, multiplicarlos por la cantidad correspondiente y guardar los
resultados en variables de tipo double */
  calcular.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ae) {
double calculo1 = (double) Integer.parseInt(campo1.getText())*5.50;

double calculo2 = (double) Integer.parseInt(campo2.getText())*8.00;
double calculo3 = (double) Integer.parseInt(campo3.getText())*1.30;

double calculo4 = (double) Integer.parseInt(campo4.getText())*6.00;

double   calculo5   =   (double)   Integer.parseInt(campo5.get-
Text())*15.50;

//se realiza la suma del total
double calculoTotal = calculo1+calculo2+calculo3+calculo4+calculo5;

//se muestra en el campo de texto el resultado
  pago.setText(String.valueOf(calculoTotal));
   }
  }
 );
```

```
/*se especifica que el panel utilizará el administrador de elemen-
tos gráficos Absolute Layout */
  panel.setLayout(new AbsoluteLayout());

/*se indican las coordenadas del panel donde aparecerá cada ele-
mento gráfico */
  panel.add(productos, new AbsoluteConstraints(40,15));
  panel.add(cantidades, new AbsoluteConstraints(120,15));
  panel.add(precios, new AbsoluteConstraints(200,15));

  panel.add(producto1, new AbsoluteConstraints(40,50));
  panel.add(campo1, new AbsoluteConstraints(120,50));
  panel.add(precio1, new AbsoluteConstraints(200,50));

  panel.add(producto2, new AbsoluteConstraints(40,90));
  panel.add(campo2, new AbsoluteConstraints(120,90));
  panel.add(precio2, new AbsoluteConstraints(200,90));

  panel.add(producto3, new AbsoluteConstraints(40,130));
  panel.add(campo3, new AbsoluteConstraints(120,130));
  panel.add(precio3, new AbsoluteConstraints(200,130));

  panel.add(producto4, new AbsoluteConstraints(40,170));
  panel.add(campo4, new AbsoluteConstraints(120,170));
  panel.add(precio4, new AbsoluteConstraints(200,170));

  panel.add(producto5, new AbsoluteConstraints(40,210));
  panel.add(campo5, new AbsoluteConstraints(120,210));
  panel.add(precio5, new AbsoluteConstraints(200,210));

  panel.add(total, new AbsoluteConstraints(305,192));
  panel.add(pago, new AbsoluteConstraints(290,207));
  panel.add(calcular, new AbsoluteConstraints(280,242));

 //se especifica el tamaño del panel
   panel.setSize(400,300);

 //se indica que el panel será visible
    panel.setVisible(true);

 //se indica que el panel no podrá redimensionarse
   panel.setResizable (false);

/*se indica que al cerrarse la ventana del panel se terminará la
ejecucución del programa*/
```

```
  panel.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }

public static void main(String[] args) {
/*se crea un objeto de la clase Tienda y automáticamente se ejecuta
lo que hay en el constructor de esa clase*/
    Tienda objeto = new Tienda();
 }
}
```

Evento	Salida
Al ejecutarse el programa, se muestra la ventana principal, cuyo título es "Calculadora para Tienda". Los campos de texto están inicializados con el valor de "0".	Calculadora para Tienda Productos Cantidad Precios Refrescos: 0 X $5.50 Papas: 0 X $8.00 Paletas: 0 X $1.30 Agua: 0 X $6.00 Pan: 0 X $15.50 TOTAL Calcular
Si se escribe algo en los campos de texto y después se da clic en el botón "Calcular", se obtendrán los valores introducidos, se hará el cálculo y se mostrará el resultado en el campo de texto inhabilitado debajo de la etiqueta "TOTAL".	Calculadora para Tienda Productos Cantidad Precios Refrescos: 10 X $5.50 Papas: 2 X $8.00 Paletas: 0 X $1.30 Agua: 1 X $6.00 Pan: 0 X $15.50 TOTAL 77.0 Calcular

7.9 Un vistazo a JTextArea y JScrollPane

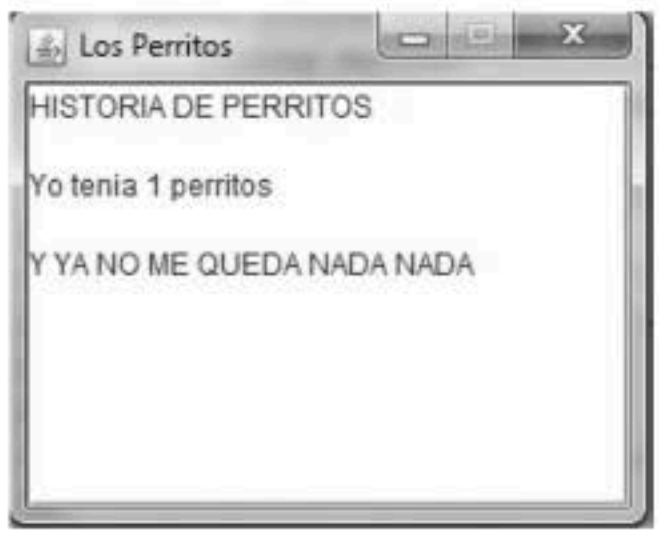
Los elementos de tipo `JTextArea` permiten contener texto plano en diferentes líneas, a diferencia de los `JTextFields` que solo puede contener una línea.

Para utilizar un objeto de área de texto es necesario crear un objeto de la clase `JTextArea`, la cual pertenece al paquete `swing`. Es importante mencionar que un objeto de tipo `JTextArea` no es capaz de manejar por si solo sus diferentes líneas a través del mecanismo de desplazamiento entre líneas, de tal forma que si la dimensión del área de texto no es suficiente para mostrar varias líneas de texto, parte del texto no quedará a la vista.

Para resolver la situación en que se desea mostrar un conjunto de líneas en un área de texto de dimensión menor a la necesaria para ciertas líneas, se puede hacer uso de un panel de desplazamiento u objeto de tipo `JScrollPane`, que también pertenece al paquete `swing`.

A continuación se muestra un ejemplo que el que se explican algunos aspectos importantes de estos dos tipos de objetos.

Ejemplo 7.11. Crear un programa que genere aleatoriamente un cierto número de líneas de texto a ser colocadas dentro de un objeto `JTextArea` susceptible a desplazamiento a través de un objeto `JScrollPane` en caso de que las líneas generadas no sean visibles dentro del tamaño del área de texto. El número de líneas es aleatorio.

Evento	Salida
Ejemplo de la primera ejecución, donde se generó aleatoriamente el número 37.	
Ejemplo de la primera ejecución, donde se generó aleatoriamente el número 1.	

```
import java.util.Random;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
```

```
public class TxtScroll {
TxtScroll(){

// Se crea la ventana donde aparecerán las salidas
   JFrame ventana= new JFrame();

// Se crea panel  que permitirá hacer scroll en la ventana
   JScrollPane scroll = new JScrollPane();

ventana.add(scroll);    // Se agrega el scroll a la ventana

// Se da título a la ventana
   ventana.setTitle("Los Perritos");

// Se indica terminar ejecución al cerrar ventana
   ventana.setDefaultCloseOperation(ventana.EXIT_ON_CLOSE);

// Se define tamaño de la ventana
ventana.setSize(250,200);

// Se indican coordenadas x,y donde aparecerá
   ventana.setLocation(400, 150);

// Se ejecuta el método crear texto
   String textoplano=creartexto();

/* Se crea un control de área de texto para poner en el la salida
resultante */
   JTextArea texto = new JTextArea(textoplano);

// Se determina que la ventana no podrá ser redimensionada
   ventana.setResizable (false);

// Se indica hacer la ventana visible
   ventana.setVisible(true);

// Se activa el scroll de acuerdo al control texto
   scroll.setViewportView(texto);
}

// método que genera el texto que irá en el JTextArea
   public String creartexto(){
    String salida=" HISTORIA DE PERRITOS \n\n";
    Random aleatorio=new Random();
    int n,c;
```

```
    n=aleatorio.nextInt(100);

    for (c=n; c>0; c--){
        salida+="Yo tenía "+c+" perritos\n"; }
    salida+="\nY YA NO ME QUEDA NADA NADA";
    return salida;
}

// parte principal de programa
   public static void main(String[] args){
// Creacion del objeto
   TxtScroll b=new TxtScroll();
}; // Termina main
}
```

A continuación se resaltan algunas líneas del código anterior:

```
JScrollPane scroll = new JScrollPane();
```

Permite crear el objeto cuyo identificador es scroll, permitirá hacer desplazamiento en las líneas que se encuentran en el área de texto, en caso necesario.

```
ventana.add(scroll);
```

Agrega el objeto scroll a la ventana, de lo contrario no se mostraría en ella.

```
JTextArea texto = new JTextArea(textoplano);
```

Crea el objeto área de texto, cuyo identificador es *texto* y al mismo tiempo se está indicando que el objeto de texto debe contener las líneas que están almacenadas en la variable `textoplano`.

```
scroll.setViewportView(texto);
```

Se indica el componente con el que trabajará el panel de desplazamiento, en este caso el componente u objeto `texto`.

Es importante destacar que los objetos de tipo `JTextArea` cuentan con el método `add` que sirve para adicionar texto al final del ya existente. Considere las siguientes líneas de código, que provocan que el objeto texto contenga finalmente 3 líneas de texto:

```
JTextArea texto = new JTextArea("Morelia\nMichoacán\n");
Texto.append("México");
```

7.10 Resumen

La GUI (*Graphical User Interface*, Interfaz Gráfica de Usuario) permite la captura y visualización de información en forma gráfica, facilitando de esta manera el desarrollo de programas y sistemas más amigables con mejor presentación.

Las interfaces gráficas tuvieron sus inicios en 1945 cuando Vannevar Bush publicó el artículo "*As we may think*", donde planteó la necesidad de crear programas amigables que permitieran una mejor

interacción hombre-máquina. Posteriormente, Douglas Engelbart inventó el ratón y uno de los primeros entornos gráficos de ventanas en 1970; más tarde la compañía Macintosh aprovecha el ratón y crea la GUI multipanel donde los archivos se representan como hojas de papel, dentro de carpetas que representan los subdirectorios y un bote de basura para guardar temporalmente los archivos y subdirectorios que ya no son de utilidad y que están en la antesala de desaparecer. Actualmente la GUI tiene amplias aplicaciones en computadoras y telefonía móvil, donde la navegación es en modo táctil. Para desarrollar una interfaz gráfica se realizan los siguientes pasos:

1.- **Entender el problema.** Que no es otra cosa que la obtención de requerimientos, o bien entender lo que el cliente desea que haga el sistema a desarrollar.
2.- **Diseño de la interfaz.** Que consiste en hacer un planteamiento gráfico en hojas de papel o pantallas para explicar en forma ilustrativa el funcionamiento del sistema, tomando en cuenta los requerimientos del cliente. Esto se hace por lo general antes de empezar la programación del sistema.
3.- **Diseño de la solución.** En esta etapa se diseña la parte funcional del *software* apoyándose en las diferentes representaciones algorítmicas como son: diagramas de flujo, diagramas N-S, Pseudocódigo, diagramas UML, entre otros. En este momento también se debe determinar la tecnología a usarse para el desarrollo del sistema (sistema operativo, lenguaje de programación, manejador de bases de datos, requerimientos mínimos del equipo para el funcionamiento del sistema, etc.).
4.- **Codificar**. Consiste en desarrollar los programas en el lenguaje de programación seleccionado anteriormente. Al hacer la codificación también se realiza paralelamente la documentación del código para explicar la finalidad de los diferentes partes del programa, citando fechas, fórmulas y recomendaciones, con la finalidad de facilitar el mantenimiento y actualización del *software*.
5.- **Realizar pruebas**. Una vez que se tiene el *software* es necesario llevar a cabo pruebas para asegurarse de que el funcionamiento es el correcto. Las pruebas se realizan saturando de información el sistema, probando datos extremos, buscando con ello fallas del nuevo sistema. En caso de encontrar algún error se deberá corregir y nuevamente realizar todas las pruebas necesarias para descartar cualquier tipo de error.

Para el manejo de interfaces graficas es necesario usar bibliotecas graficas del lenguaje de programación que se usará. En Java existen dos paquetes gráficos que son de utilidad en el desarrollo de las GUI. Estos paquetes gráficos son: AWT (*Abstract Window Toolkit*, Conjunto de herramientas de Ventanas Abstractas) y Swing, los cuales forman parte de las JFC (*Java Foundation Classes*, Clases Fundamentales de Java). De tal manera que es necesaria la importación de los mismos de la siguiente manera:

```
  //Para utilizar todas las clases de Swing
import javax.swing.*;

  //Para utilizar todas las clases de AWT
import java.awt.*;
```

La mayoría de los lenguajes de programación actuales tienen muchas herramientas que se pueden usar en la programación gráfica entre las cuales se pueden mencionar.

- **Cuadros de dialogo**. Son elementos gráficos que permiten la interacción entre el usuario y la computadora. Para usar los cuadros de dialogo en Java es necesario importar la clase `JOptionPane` de la clase `swing` de la siguiente manera.

import javax.swing.JOptionPane;

Los cuadros de dialogo de salida permiten mostrar información. El método que se usa es `showMessageDialog` de la clase `JOptionPane`, el cual está sobrecargado, brindado así la opción de elegir el método más conveniente dependiendo de lo que se desee programar.

- **Etiquetas.** Son elementos gráficos que permiten mostrar texto no editable. Para usar una etiqueta es necesario crear un objeto de la clase `JLabel` la cual es parte del paquete `swing`. El objeto se puede crear de la siguiente manera:´

```
JLabel nombre_etiqueta = new JLabel (cadena_etiqueta);
```

- **Cuadros de texto.** Permiten el ingreso de información de teclado. Requieren para su utilización importar el paquete `swing` y las clases `ActionEvent` y `ActionListener` para manejar los eventos de los cuadros de texto:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

Por medio de los cuadros de texto es posible realizar ciertas acciones cuando se presiona la tecla "enter". Para el uso de cuadros de texto es necesario crear un objeto. Ejemplo:

```
JtextField nombre_campo1 = new JtextField (numero_columnas);
JtextField nombre_campo2 = new JtextField (cadena_predefinida);
```

- **Botones.** Los botones son elementos gráficos que permiten ejecutar ciertas acciones al momento de presionarse. Para crear un botón en Java, es necesario importar el paquete `swing`, las clases `ActionEvent` y `ActionListener` para manejar sus eventos y crear el objeto de la clase `JButton`.

```
JButton calcular = new JButton(cadena_mensaje_boton);
```

Todos los elementos gráficos como son: cuadros de texto, botones, etiquetas, etc., deben ser parte de un panel `JFrame` porque no pueden aparecer de manera aislada en una pantalla. Cuando se declara el panel `JFrame` es necesario un administrador de elementos gráficos para distribuir y manejar adecuadamente cada uno de esos elementos gráficos. Hay varios elementos gráficos pero uno de los más utilizados es: `Absolute Layout`.

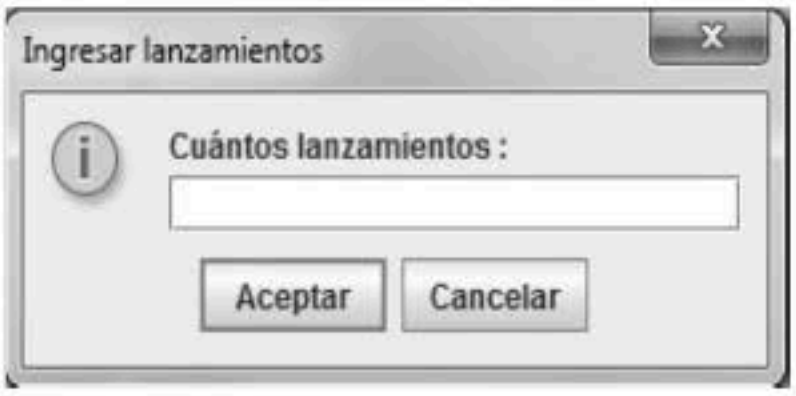
La finalidad de este libro es invitar al alumno para que use su razonamiento lógico en el desarrollo de algoritmos que permitan dar respuesta a problemas determinados, usando para ello diferentes formas de representación algorítmica como: diagramas de flujo, diagramas N-S y pseudocódigo. Se decidió ver el funcionamiento de esos algoritmos codificándolos en Java, pero principalmente en modo consola con la intención de que los conocimientos a desarrollar en el alumno sea el diseño de algoritmos y no los adornos que le pueda poner a los mismos. Con el objetivo de encausar al alumno en la programación orientada a objetos y en la programación gráfica, se agregaron las unidades de introducción a la programación orientada a objetos e introducción a las interfaces gráficas de usuario, pero solamente es una probadita de estos temas, ya que la POO y la programación de GUI son tan amplias que requieren un trato especial.

7.11 Problemas

1.- Escriba un programa para desplegar en la pantalla una lista de 4 países, cada uno de ellos asociado a un código. Se deberá ingresar el código del país en un `JTextField` y la computadora indicará cuál es la capital, continente, idioma y moneda oficiales de ese país. Utilice los controles gráficos de tipo `JFrame`, `JTextField`, `JButton`, así como `showMessageDialog`, considerando la siguiente ejecución:

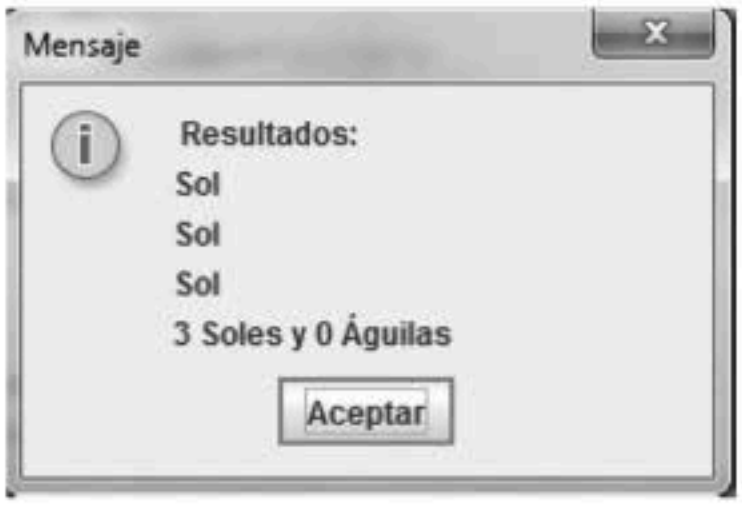
Evento	Salida
Al ejecutarse el programa, se debe mostrar la ventana principal, cuyo título es "LISTA DE PAÍSES". Un cuadro de texto editable aparece inicializado con el valor de "01". Incluir el botón "Mostrar Info"	
Al dar clic en el botón debe mostrarse en un cuadro de diálogo la información del país que corresponda al valor que se introdujo en el cuadro de texto editable. Considere que el título del cuadro de diálogo es el valor que se introdujo en el cuadro de texto *(En esta caso se considera que se introdujo 3).*	

2.- Escribir un programa para simular n lanzamientos de una moneda al aíre. Los resultados se generarán aleatoriamente. Imprimir cada uno de los resultados e imprimir los resultados obtenidos al final del evento, utilizando únicamente `showInputDialog` y `showMessageDialog` para entrada y salida respectivamente.

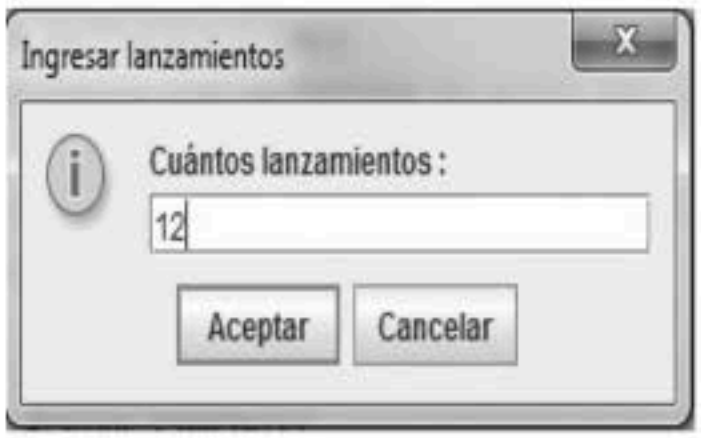
Evento	Salida
Al ejecutarse el programa, se debe mostrar un cuadro de diálogo para ingresar el número de lanzamientos.	

(*Continúa*)

(*Continuación*)

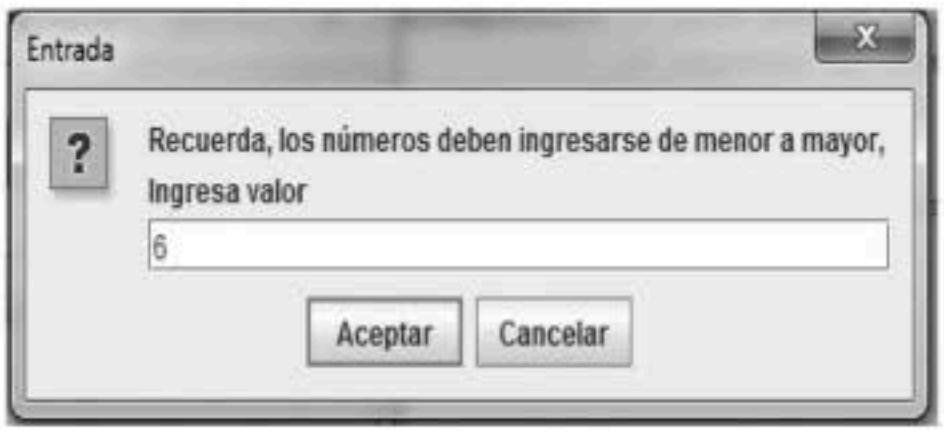
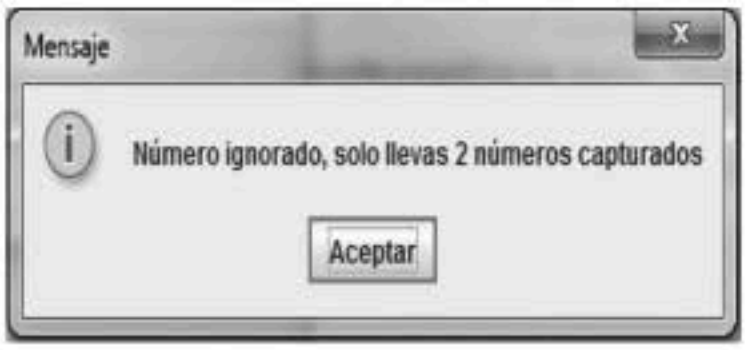
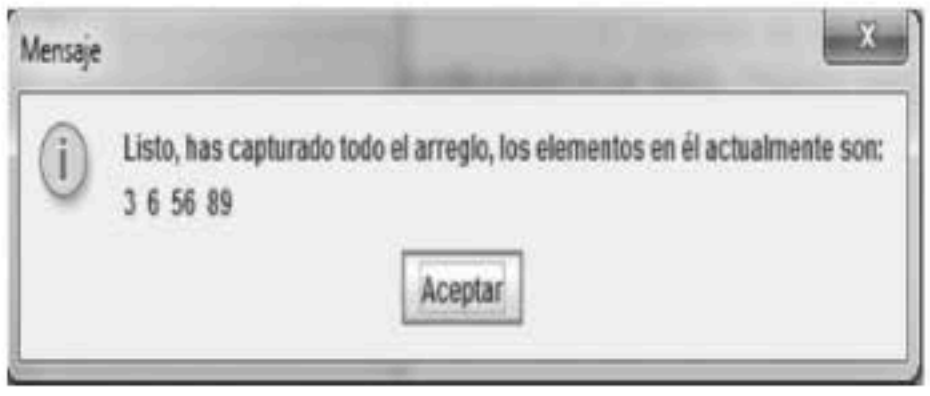
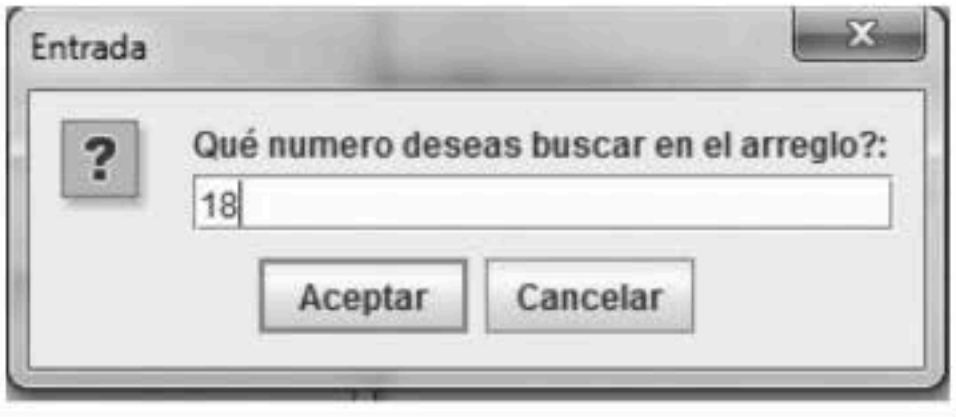
Evento	Salida
Después de haber ingresar un valor se mostrará el resultado como se muestra *(En esta caso se considera que se introdujo 3).*	

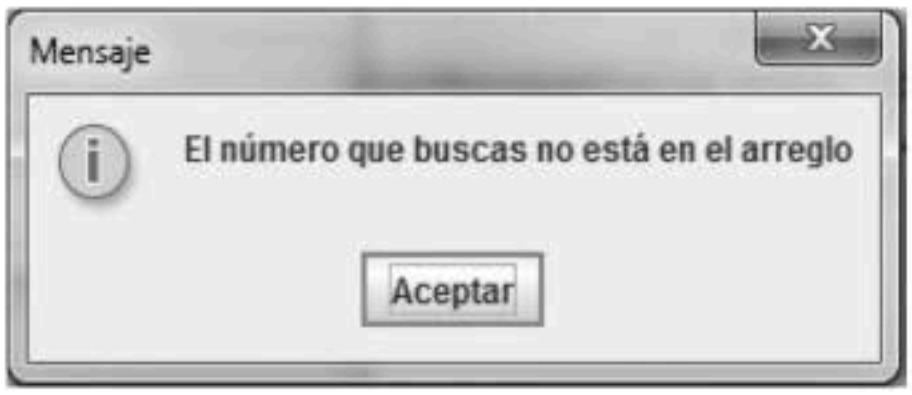
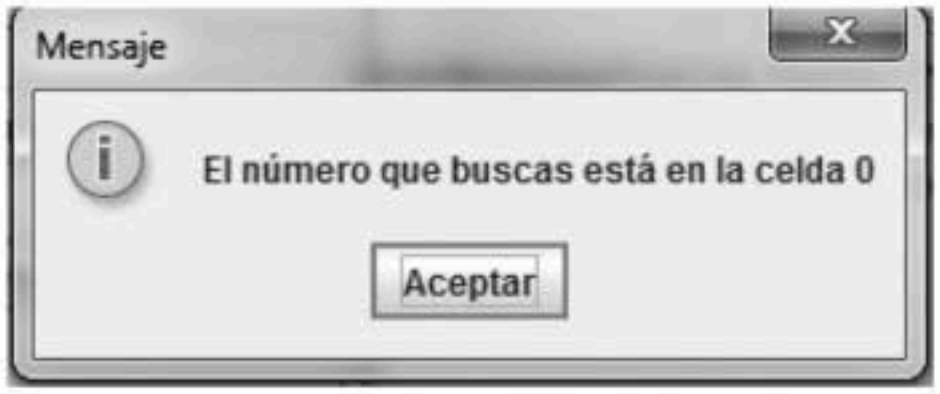
3.- Escribir el programa anterior de lanzamientos de moneda, pero para mostrar los resultados utilice una ventana con barra de desplazamiento, con el objetivo de poder ver todos los resultados aun cuando el programa se ejecute para una gran cantidad de lanzamientos.

Evento	Salida
Al ejecutarse el programa, se debe mostrar un cuadro de diálogo para ingresar el número de lanzamientos.	
Después de ingresar el número de lanzamientos se muestra el resultado, en una ventana con scroll. El texto quedará en un JTextArea *(En esta caso se considera que se introdujo 12).*	

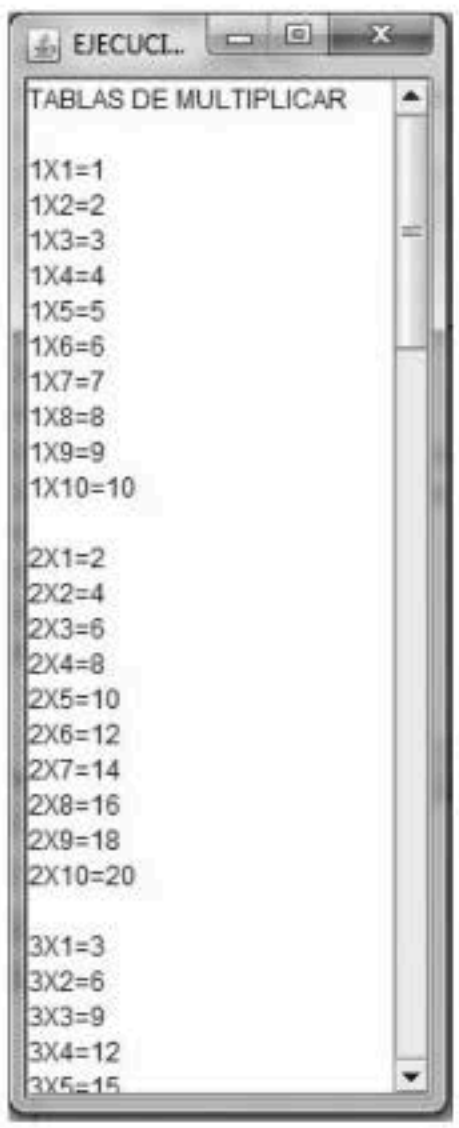
4.- Escribir un programa que muestre una ventana con 2 botones de opcion: **1)** Llenar el arreglo y **2)** Hacer Búsqueda binaria. La primera opción pedirá al usuario *4 números* enteros para ser almacenados en un arreglo, cuidando que los números se introduzcan de manera ascendente en el arreglo, en otras palabras, el número que se acaba de introducir debe ser mayor o igual que el anterior (la búsqueda binaria trabaja sobre arreglos ordenados). El botón **2)** solicitará al usuario un número para ser bus-

cado en el arreglo; si el número se encuentra en el arreglo el programa deberá mostrar en qué celda se encontró, de lo contrario mostrará el mensaje "el número que buscas no está en el arreglo". Para la búsqueda del elemento en el arreglo se aplicará la búsqueda binaria.

Evento	Salida
Al ejecutarse el programa, se debe mostrar una ventana con los botones que se muestran.	
Después de elegir el botón *ingresar números,* debe mostrarse un diálogo de entrada para pedir los números.	
Si los números no se ingresan en orden ascendente, un diálogo de mensaje deberá indicarlo *(Se supone que se ingresaron 3, 6 y luego se intentó ingresar un 4)*	
Al terminar de introducir los números, éstos deben mostrarse con un diálogo de mensaje.	
Después de elegir el botón *hacer búsqueda binaria,* debe preguntarse el número que desea buscarse.	

Evento	Salida
Si no existe en el arreglo el valor buscado, deberá indicarse con un mensaje	
Si el valor buscado sí está en el arreglo, deberá indicarse en qué celda se encontró.	

5.- Escribir un programa que muestre las tablas de multiplicar del 1 al 10, en una ventana que permita hacer scroll.

Evento	Salida
Al ejecutarse el programa, deben mostrarse en una ventana con scroll las tablas de multiplicar del 1 al 10, de manera que si el usuario desea tener a la vista las últimas talas, éste haga uso de la barra de desplazamiento.	

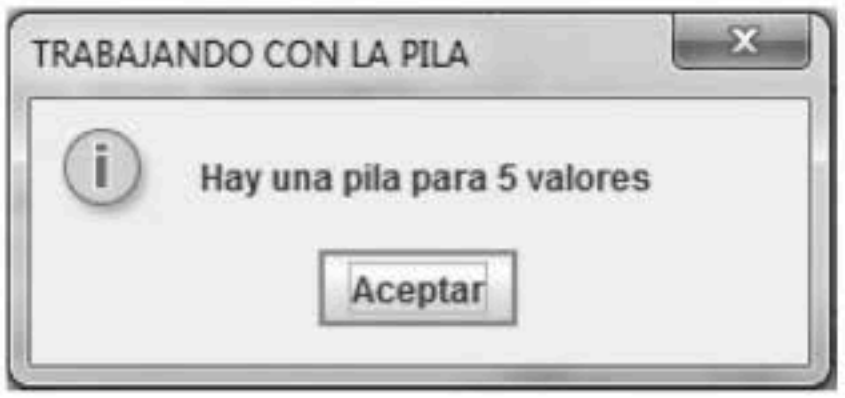
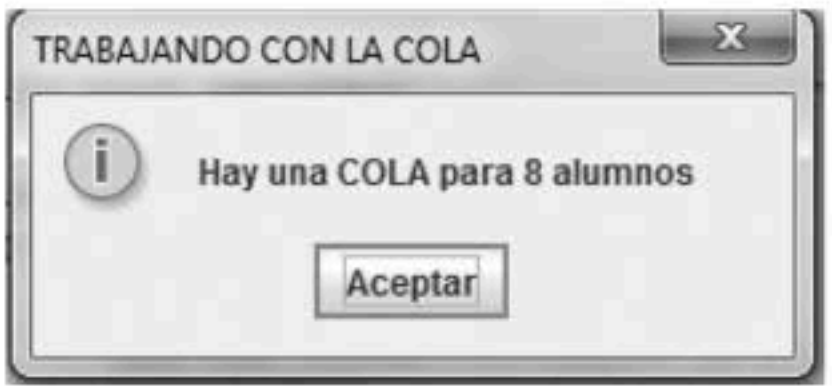
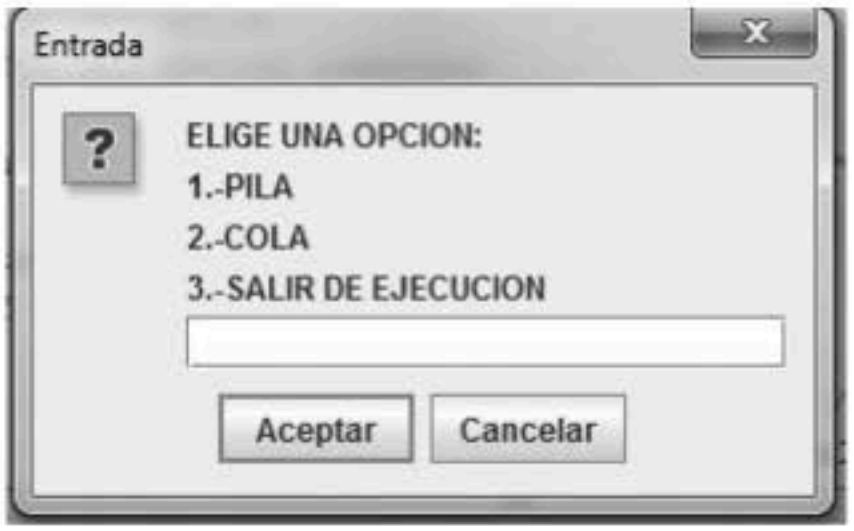
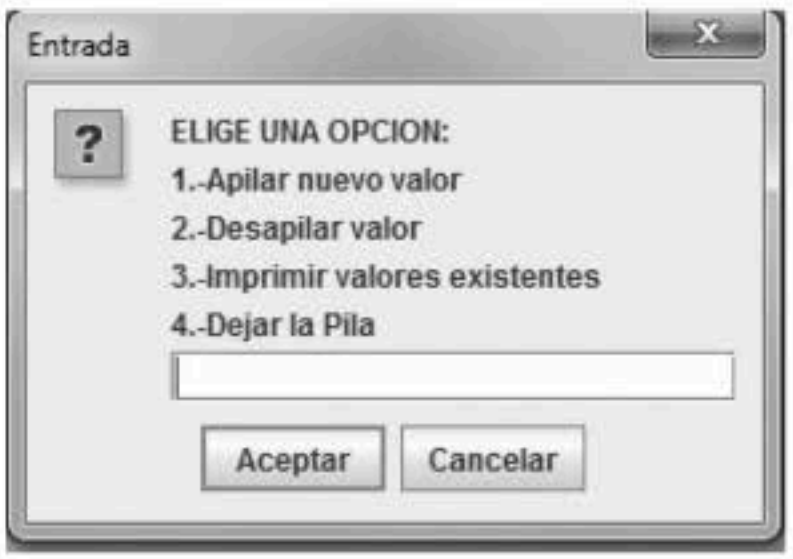
6.- Una lavandería compró equipo que se va a depreciar en cierto lapso de tiempo por el método de "Saldo de doble declinación"; es decir; la depreciación de cada año está dada por la siguiente fórmula:

$$d = \frac{2 * precio}{vida}$$

Siendo "*precio*" el valor inicial del equipo, y "*vida*" los años de vida útil en los cuales se deprecia el equipo. Escribir un programa para calcular la depreciación de un bien en base a su valor inicial y a su vida útil, considerando el siguiente ejemplo de ejecución.

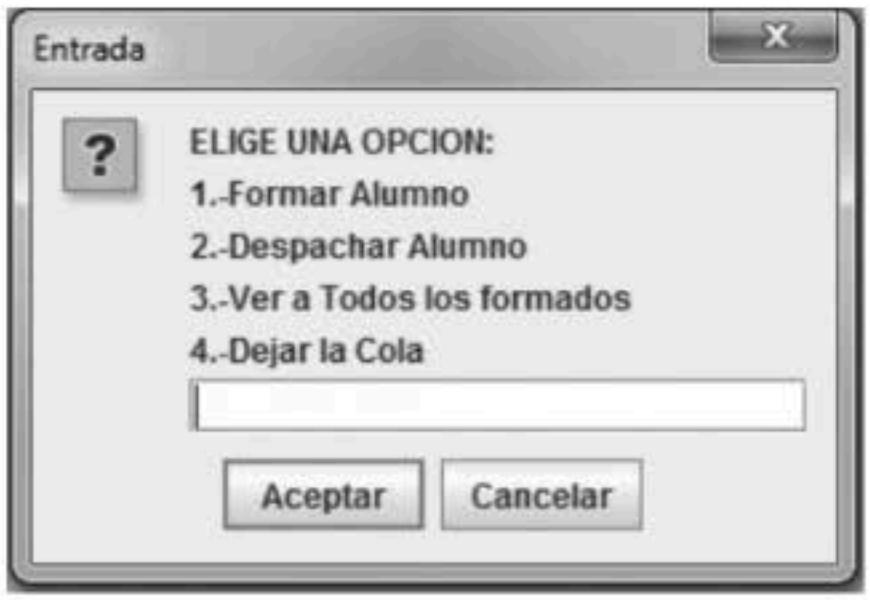
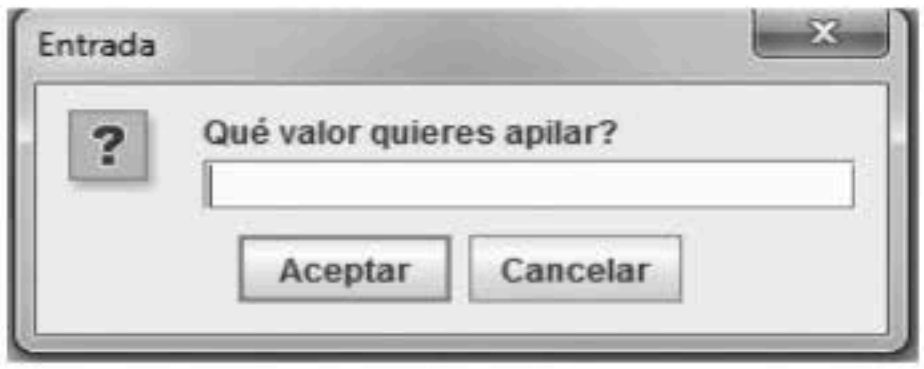
Evento	Salida
Al ejecutarse el programa, aparecerá una ventana con los JTextFields y el botón mostrados. Aun cuando los valores de valor de equipo y vida esperada están predeterminados, el usuario puede editarlos para calcular la depreciación.	
Al presionar el botón *depreciación,* aparecerá una ventana con barra de desplazamiento con los resultados año por año. En este momento el botón de la ventana inicial deberá desaparecer.	

7.- Escribir un programa que utilice dos arreglos unidimensionales: uno de tipo entero con 5 celdas que se manejará bajo el principio de pilas y otro de tipo String de 8 celdas que será manejado bajo el principio de colas. Diseñe la solución con dos clases, una para manejar un arreglo con el principio de pilas y otra para manejar un arreglo en base al principio de colas. Utilice cuadros de diálogo y de mensaje, considerando la ejecución como a continuación se muestra:

Evento	Salida
Al ejecutarse el programa, primero aparecerá una ventana indicando la capacidad del arreglo de enteros, después otra ventana indicando la capacidad del arreglo de cadenas.	 
A continuación aparecerá una ventana ofreciendo elegir entre trabajar con una pila (arreglo de enteros) o una cola (arreglo de cadenas)	
Dependiendo de la elección en la ventana anterior, se desplegará una de las dos ventanas que se muestran. En ellas el usuario podrá elegir la acción que desea realizar.	

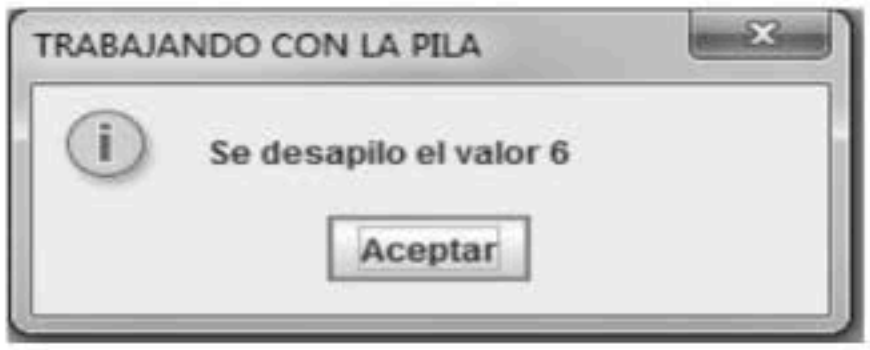
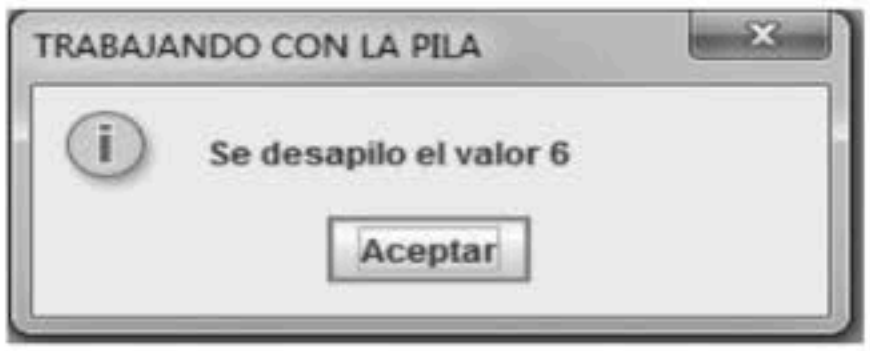
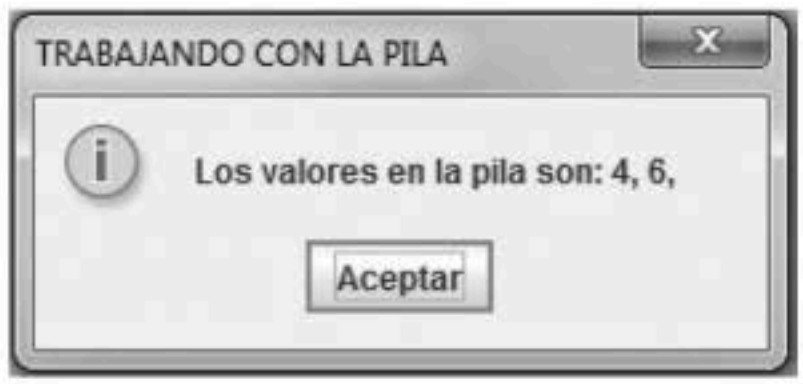
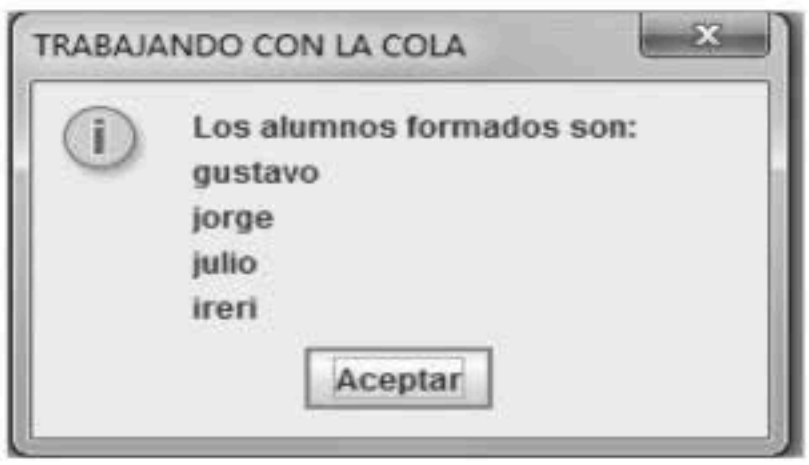
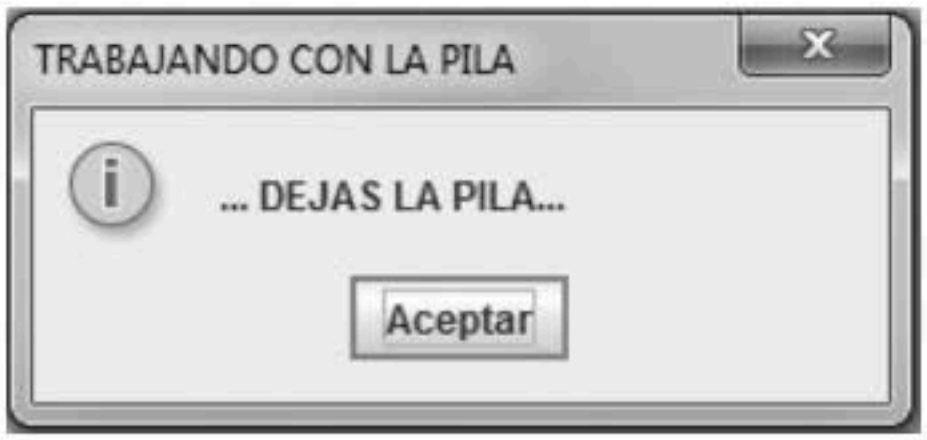
(*Continúa*)

(*Continuación*)

Evento	Salida
	
Al elegir la opción 1 en alguna de las ventanas anteriores, se desplegará una de estas ventanas, donde se tendrá oportunidad de ingresar un valor en el arreglo correspondiente.	
Al intentar ingresar un elemento cuando el arreglo correspondiente ya está lleno, deberá mostrarse un mensaje que así lo indique.	

(*Continúa*)

(*Continuación*)

Evento	Salida
Al elegir la opción Desapilar valor o Despachar alumno, debe mostrarse el elemento que se eliminó del arreglo correspondiente, de acuerdo a los principios de pilas o colar, según sea el caso. En caso de que se intente quitar un elemento e el arreglo deberá mostrarse el mensaje correspondiente.	 
Al elegir la opción 3 se mostrará una de estas ventanas visualizando lo almacenado en el arreglo con el que se esté trabajando actualmente.	 
Al elegir la opción 4 se mostrará un mensaje que indique que dejará de trabajar con ese arreglo (en este caso la pila). A continuación deberá regresar al menú principal (ventana de 3 opciones)	

(*Continúa*)

(*Continuación*)

Evento	Salida
Ventana que se mostrará al terminar la ejecución del programa (opción 3 del menú principal)	

RESPUESTAS A LOS PROBLEMAS NONES

Respuestas a capítulo 1

1.

	Definición	Concepto
1.	Equipo con el cual se puede obtener la información de la computadora que resulta de un procesamiento de datos o bien información que está almacenada en la computadora.	Sistema operativo (17)
2.	Es una persona o sujeto que utiliza una computadora, sistema operativo o cualquier sistema computacional.	Medios de Entrada (14)
3.	Programas que permiten administrar los recursos de la computadora	*Hardware* (20)
4.	Máquina electrónica que recibe datos de un medio de entrada, procesa dichos datos de una manera rápida y exacta, para posteriormente enviar la información resultante a un medio de salida.	*Software* de sistemas (3)
5.	Conjunto finito de pasos, precisos y ordenados para resolver un problema.	Lenguajes formales (8)
6.	Conjunto de programas y procedimientos necesarios para que la computadora lleve a cabo alguna tarea específica.	Windows NT (16)
7.	Conjunto de instrucciones basadas en un lenguaje de programación que una computadora ejecuta para resolver un problema o realizar una función específica.	Medios de salida (1)
8.	Lenguajes de menor capacidad para simular y modelar lenguajes naturales.	Paradigma de programación (30)
9.	Tiene una amplia gama de posibilidades para el manejo y la presentación del texto, entre ellas: Tipo y tamaño de letra, Formateo de párrafos, Efectos artísticos, Corrector de ortografía, Diccionario en varios lenguajes, Intercalado de imágenes, entre otras.	Programación (18)

(*Continúa*)

(*Continuación*)

	Definición	Concepto
10.	Programas para llevar a cabo tareas específicas como: Edición de textos, Graficación, Cálculos, Diseños, Simulación, entre otras tareas.	Lenguaje máquina (28)
11.	Analiza el programa fuente y lo traduce a otro equivalente llamado lenguaje objeto.	Computadora (4)
12.	Programa o archivo que contiene instrucciones en código máquina y que puede ejecutarse en una plataforma determinada.	Lenguaje de programación (19)
13.	Lenguajes orientados a la inteligencia artificial. Estos lenguajes todavía están poco desarrollados. Ejemplo de ellos son: LISP, Prolog, OPS5 y Mercury.	Windows (26)
14.	Herramientas utilizadas para ingresar todo tipo de datos a la computadora.	Programa (7)
15.	Sistema operativo de código libre, portable, multiusuarios, escrito en lenguaje C, que fue lanzado al mercado a principios de la década de los noventas tomando en cuenta las sugerencias de programadores.	*Software* (6)
16.	Permite trabajar con un *software* al mismo tiempo en una red de usuarios aprovechando de esa manera los beneficios de Windows en redes de computadoras.	Editor de texto (27)
17.	Programa cuya finalidad es organizar el trabajo de la computadora.	Algoritmo (5)
18.	Sistema de escritura para la descripción precisa de algoritmos o programas informáticos.	*Software* de aplicación (10)
19.	Conjunto de reglas sintácticas y semánticas que permiten la comunicación con una computadora.	Compilador (11)
20.	Tangibles de un sistema computacional.	Usuario (2)
21.	Permite al programador escribir las instrucciones de un programa utilizando palabras de un idioma.	Intérprete (29)
22.	Lenguaje que la computadora puede entender en el momento de ejecutar un programa.	Linux (15)
23.	Salió a principios de la década de los 80s, fue desarrollado por Microsoft y se hizo popular debido a la gran cantidad de *software* que se podía ejecutar con él.	Ejecutable (12)
24.	Sistema operativo multiusuario y multitarea desarrollado por los laboratorios Bell de AT&T que corre en diferentes computadoras: personales, minicomputadoras, supercomputadoras y redes de computadoras.	Lenguajes de quinta generación (13)

(*Continúa*)

(*Continuación*)

	Definición	Concepto
25.	Derivado del lenguaje máquina y está formado por abreviaturas de palabras y números llamados mnemotécnicos.	Procesador de texto (9)
26.	Sistema operativo desarrollado con la finalidad de tener una interfaz gráfica más amigable con íconos que representan las diferentes operaciones que se pueden ejecutar.	Lenguaje de bajo nivel (22)
27.	Programa que permite crear y modificar archivos digitales compuestos únicamente por texto plano sin formato.	Unix (24)
28.	Las instrucciones de este lenguaje están integradas por 0´s y 1´s.	Lenguajes de alto nivel (21)
29.	Analiza cada línea del programa fuente que se va escribiendo y lo traduce directamente sin generar ningún código equivalente.	MS−DOS (23)
30.	Propuesta tecnológica que es adoptada por una comunidad de programadores cuyo núcleo central es incuestionable en cuanto a que unívocamente trata de resolver uno o varios problemas claramente delimitados.	Lenguaje ensamblador (25)

3.

Software	Sistema operativo	Lenguaje	Intérprete	Compilador	Hoja electrónica	Procesador de texto	Monousuario	Multiusuario	Monotarea	Multitarea	*Software* libre	De alto nivel	De bajo nivel	Lenguaje máquina
MS−DOS	*						*		*			*		
Pascal		*		*			*		*			*		
Basic		*	*				*		*			*		
Unix	*						*	*	*	*		*		
Excel				*			*	*	*			*		
JSP		*		*			*	*	*	*	*	*		
Linux	*						*	*	*	*	*	*		
Java		*		*			*	*	*			*		
Windows NT	*						*	*	*	*		*		
Ada		*		*			*	*	*		*	*		
Word						*	*	*	*			*		

(*Continúa*)

(*Continuación*)

Software	Sistema operativo	Lenguaje	Intérprete	Compilador	Hoja electrónica	Procesador de texto	Monousuario	Multiusuario	Monotarea	Multitarea	*Software* libre	De alto nivel	De bajo nivel	Lenguaje máquina
PHP		*	*				*	*	*			*		
C++		*		*			*	*	*			*		
MacOS	*						*	*	*	*		*		
Binario		*												*
Windows	*						*		*	*		*		
Ensamblador		*	*				*		*				*	
Word Perfect						*	*		*			*		

5.

Algoritmo para sumar dos números A y B e imprimir el resultado

Inicio
Leer el número A
Leer el número B
Hacer C=A+B
Imprimir C
Fin

Algoritmo para leer dos números A y B e imprimir el mayor de ellos

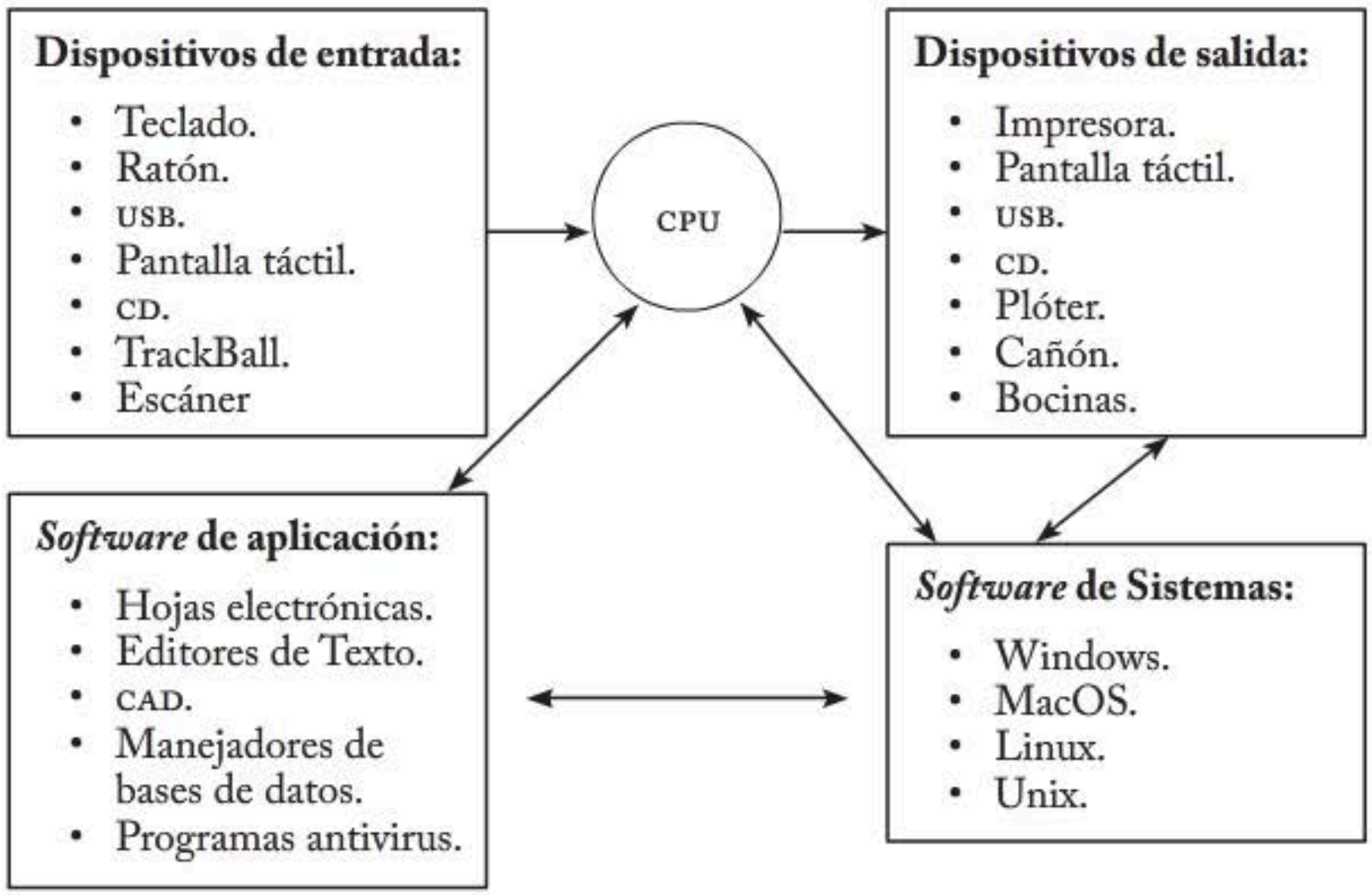
Inicio
Leer A
Leer B
Si A>B entonces imprimir A
Sino imprimir B
Fin

Algoritmo para ingresar como alumno al Tecnológico de Morelia

Inicio
Solicitar informes
Obtener ficha
Obtener recibo de pago de examen de admisión
Si tiene ficha y tiene recibo de pago Entonces
Inicio
Presentar examen de admisión
Si aprobó examen de admisión Entonces
Ingresa a Tecnológico de Morelia.
Fin
Fin

7.

El mapa conceptual puede variar, pero una respuesta posible es:



9.

Década	Características del *hardware* y/o *software*
80	Las computadoras usaban cintas y discos magnéticos de $5\frac{1}{4}$ pulgadas para guardar y leer información.
00	Se usan con frecuencia para programar los lenguajes C++ y Java.
70	Las computadoras eran de grandes dimensiones y utilizaban para leer información tarjetas y/o cintas perforadas.
90	Surgen las computadoras portátiles "laptop" y las USB como dispositivos de almacenamiento secundario.
70	Los lenguajes más utilizados en esta década fueron Basic y Fortran IV.
90	Se empezaron a utilizar los sistemas operativos Windows, Linux y Unix.
00	El ipod, cámaras digitales y robots son muy comunes en esta década.
90 y/o 00	Sistemas operativos multitarea y multiusuario son comunes.
80	Los lenguajes más utilizados eran Basic, Pascal y Cobol.
80	MS−DOS fue el sistema más utilizado en esta década.

Respuestas a capítulo 2

1.

a) Pintar un carro

- inicio
- Lavar el coche
- Lijar el auto
- Limpiar superficie
- Empapelar (Poner papel a cristales, faros, calaveras)
- Preparar pintura.
- Colocar pintura en recipiente de pistola.
- Conectar pistola a compresora.
- Pintar superficie.
- Dejar secar superficie pintada
- Desconectar equipo de pintado.
- Lavar equipo de pintado.
- Pulir superficie pintada
- fin

b) Preparar un café con leche.

- Inicio
- Poner a en una olla
- Encender la estufa.
- Poner olla en la estufa.
- Colocar café en agua
- Dejar hervir el café
- Apagar el fuego de estufa.
- Servir el café.
- Agregar leche.
- Agregar azúcar al gusto
- Agitar la mezcla de café, leche y azúcar.
- Fin

c) Enviar una fotografía y un mensaje por medio de Hotmail.

- Inicio
- Encender computadora.
- Ingresar a Internet
- Accesar página de Hotmail.
- Teclear usuario y clave de usuario.
- Seleccionar escritura de correo electrónico.
- Escribir el mensaje.
- Adjuntar la fotografía.
- Enviar correo con archivo adjunto.
- Salir de Hotmail

- Salir de Internet.
- Apagar la computadora.
- Fin

d) Ingresar al cine para ver una película en el cine.

- Inicio
- Trasladarse a al cine.
- Seleccionar la película en la cartelera.
- Comprar el boleto
- Comprar palomitas, café y refresco.
- Ingresar al cine
- Disfrutar de la película.
- Salir del cine
- Trasladarse a la casa
- Fin

3.

a) $y = (a*c-5)/(7*b)-9*a$

b) $y = b*((a+7*f)/5)\ ^\wedge 3+4$

c) $y = -5*x^\wedge 3+2*x-9$

d) $y = ((2*a-b^\wedge 3)/7)/((5*e+3)*c)$

e) $y = d*(3*c+e)-((f-2*a)/(b+6))/(7*(4*d-a)\ ^\wedge 2)$

f) $y = (3*a-c) \bmod ((b^\wedge 3+7)/(d+4*c))-a*b$

g) $y = ((c^\wedge 4+3*b)/((d+5*a) \bmod 7))/(b+8*c)$

h) $y = (5*a^\wedge 2+(4*a*b-c)/(d-3*b)\ ^\wedge 3)\ ^\wedge 7+5/((a+d*c)*b^\wedge 2)$

5.

a) $y = \dfrac{a-b}{3(5c+b)} + \dfrac{4(d+3c)}{2a}$

b) $y = 5c^2 - d(a+bc)^{2/5}$

c) $y = \dfrac{(4+8b)\left(\dfrac{3a}{4}-9c\right)}{\dfrac{6b}{e}+7}$

d) $y = 5c+3b^2-\left(a+\dfrac{b}{c}\right)^7$

e) $y = \dfrac{b}{\dfrac{(2c-e)^4}{a}} - \dfrac{\dfrac{5c+6}{4d}}{e}$

f) $y = \dfrac{(a+3e-7c^2b)4}{5(b-c)^2}$

g) $y = \dfrac{3-de}{4-\dfrac{5a}{d}} + \dfrac{7}{a-b^2}$

h) $y = 8ab^2 - \dfrac{2c+3}{(d+7)^4} + \dfrac{1}{3}$

7. Para $a = 1$, $b = -2$, $c = 3$, $d = 4$.

a) $y = b + 3*(c - b*(5 + a\wedge 2) - 3*d) + d$

$y = -2 + 3*(3 -(-2)*(5 + 1\wedge 2)-3*4) + 4$ *Sustituyendo valores*

$y = -2 + 3*(3 -(-2)*(5 + 1) - 3*4) + 4$ *Paréntesis interno*

$y = -2 + 3*(3 -(-2)*6 - 3*4) + 4$

$y = -2 + 3*(3 + 12-12) + 4$

$y = -2 + 3*(15 - 12) + 4$

$y = -2 + 3*3 + 4$
$y = -2 + 9 + 4$
$y = 7 + 4$
$y = 11$

b) $y = (a + c\text{^}3)/b + 5*c + (c + (4*b + 7)\text{ ^}2)$
$y = (1 + 27)/(-2) + 5*3 + (3 + (4*(-2) - 7)\text{ ^}2)$ *Sustituyendo valores*
$y = 28/(-2) + 5*3 + (3 + (4*(-2) - 7)\text{ ^}2)$
$y = 28/(-2) + 5*3 + (3 + (4*(-2) -7)\text{ ^}2)$
$y = 28/(-2) + 5*3 + (3 + (-8 -7)\text{ ^}2)$
$y = 28/(-2) + 5*3 + (3 + (-15)\text{ ^}2)$
$y = 28/(-2) + 5*3 + (3 + 225)$
$y = 28/(-2) + 5*3 + 228$
$y = -14 + 5*3 + 228$
$y = -14 + 15 + 228$
$y = 1 + 228$
$y = 229$

c) $y = 5*d/(22 - c\text{^}3) + a*b - (-32/(2*b - d))\text{ ^}2$
$y = 5*4/(22-3\text{^}3) + 1*(-2) - (-32/(2*(-2) - 4))\text{ ^}2$ *Sustituyendo valores*
$y = 5*4/(22-27) + 1*(-2) - (-32/(2*(-2)-4))\text{ ^}2$ *Paréntesis izquierdo*
$y = 5*4/(-5) + 1*(-2) - (-32/(2*(-2)-4))\text{ ^}2$
$y = 5*4/(-5) + 1*(-2) - (-32/(-4-4))\text{ ^}2$
$y = 5*4/(-5) + 1*(-2) - (-32/-8)\text{ ^}2$
$y = 5*4/(-5) + 1*(-2) - 4\text{^}2$
$y = 5*4/(-5) + 1*(-2) - 16$
$y = 20/(-5) + 1*(-2) - 16$
$y = -4 + 1*(-2) - 16$
$y = -4 - 2 - 16$
$y = -6 - 16$
$y = -22$

d) $y = (4*c - b*d/\mathrm{a} + 3 + b\text{^}2)/c$
$y = (4*3 - (-2)*4/1 + 3 + (-2)\text{^}2)/3$ *Sustituyendo valores*
$y = (4*3 - (-2)*4/1 + 3 + 4)/3$
$y = (12 - (-2)*4/1 + 3 + 4)/3$
$y = (12 + 8/1 + 3 + 4)/3$
$y = (12 + 8 + 3 + 4)/3$
$y = (20 + 3 + 4)/3$
$y = (23 + 4)/3$
$y = 27/3$
$y = 9$

9. Sean a, b, c y x variables enteras cuyo valor es $a = -1$, $b = 7$, $c = 3$, $d = 5$.

a) $x = (b * c - d \bmod a) \bmod (a - b)$
$x = (7 * 3 - 5 \bmod -1) \bmod (-1 - 7)$ *Sustituyendo valores*
$x = (21 - 5 \bmod -1) \bmod (-1 - 7)$
$x = (21 + 0) \bmod (-1 - 7)$
$x = 21 \bmod -8$
$x = 5$

b) $x = (((a - b) \bmod c) \bmod (c / a)) \bmod (b * c)$
$x = (((-1 - 7) \bmod 3) \bmod (3 / -1)) \bmod (7 * 3)$ *Sustituyendo valores*
$x = ((-8 \bmod 3) \bmod (3 / -1)) \bmod (7 * 3)$
$x = (-2 \bmod (3 / -1)) \bmod (7 * 3)$
$x = (-2 \bmod -3) \bmod (7 * 3)$
$x = -2 \bmod (7 * 3)$
$x = -2 \bmod 21$
$x = -2$

c) $x = ((d + \text{a} \bmod c)* b - (d \bmod a\text{^}3)) \bmod c$
$x = ((5 - 1 \bmod 3)* 7 - (5 \bmod (-1)\text{^}3)) \bmod 3$ *Sustituyendo valores*
$x = ((5 - 1)* 7 - (5 \bmod (-1)\text{^}3)) \bmod 3$
$x = (4* 7 - (5 \bmod (-1)\text{^}3)) \bmod 3$
$x = (4* 7 - (5 \bmod -1)) \bmod 3$
$x = (4* 7 - 0) \bmod 3$
$x = (28 - 0) \bmod 3$
$x = 28 \bmod 3$
$x = 1$

d) $x = a \bmod \text{d} \bmod b - 3*\text{c} \bmod \text{d} \wedge 2 + a/c \bmod b$
$x = -1 \bmod 5 \bmod 7 - 3*3 \bmod 5 \wedge 2 - 1/3 \bmod 7$ *Sustituyendo valores*
$x = -1 \bmod 5 \bmod 7 - 3*3 \bmod 25 - 1/3 \bmod 7$
$x = -1 \bmod 7 - 3*3 \bmod 25 - 1/3 \bmod 7$
$x = -1 - 3*3 \bmod 25 - 1/3 \bmod 7$
$x = -1 - 3*3 \bmod 25 - 1/3 \bmod 7$
$x = -1 - 9 \bmod 25 - 1/3 \bmod 7$
$x = -1 - 9 - 1/3 \bmod 7$
$x = -1 - 9 + 0 \bmod 7$
$x = -1 - 9 + 0$
$x = -10$

11. Si $a = 2$, $b = -1$, $c = 4$ y $d = -3$.

Respuestas:

a) si (($a > b$) *and* (*not*(($c <= d$) *or* (d ! = a))))
imprimir "Manzana";
sino
imprimir "Pera", a*b;

Evaluación de la proposición:
((a > b) and (not((c <= d) or (d != a))))
((2 > −1) and (not((4 <= −3) or (−3 != 2))))
(1 and (not(0 or 1)))
(1 and (not(1)))
(1 and 0)
0
El mensaje impreso es: Pera −2

b)
```
si (not((a = = b) or (c < d) and (c > = a)))
    imprimir "Morelia ", (a + b);
   sino
    imprimir "Tlaxcala ", d;
```

Evaluación de la proposición:
(not((a = = b) or (c<d) and (c >=a)))
(not((2 = = −1) or (4<−3) and (4 >=2)))
(not(0 or 0 and 1))
(not(0 or 0))
(not(0))
1
Mensaje: Morelia 1

c)
```
si (a < d or not(c != b) and c >= a*3)
   imprimir "Hola: ", 5*c;
  sino
   imprimir b," Adios";
```

Evaluación de la proposición:
(a < d or not(c != b) and c >= a*3)
(2 <− 3 or not(4 != −1) and 4 >= 2*3)
(2 <− 3 or not(4 != −1) and 4 >= 6)
(0 or not(1) and 0)
(0 or 0 and 0)
(0 or 0)
0
Mensaje: Adios

d)
```
si (c >= b*a and ((d != a) or (4*a >= c)) and not (b > c))
   imprimir "Oaxaca= ", 3*d;
 sino
   imprimir "Michoacán= ", (c − d*a);
```

Evaluación de la proposición:
(c >= b*a and ((d != a) or (4*a >= c)) and not (b > c))
(4 >= −1*2 and ((−3 != 2) or (4*2 >= 4)) and not (−1 > 4))
(4 >= −2 and ((−3 != 2) or (8 >= 4)) and not (−1 > 4))

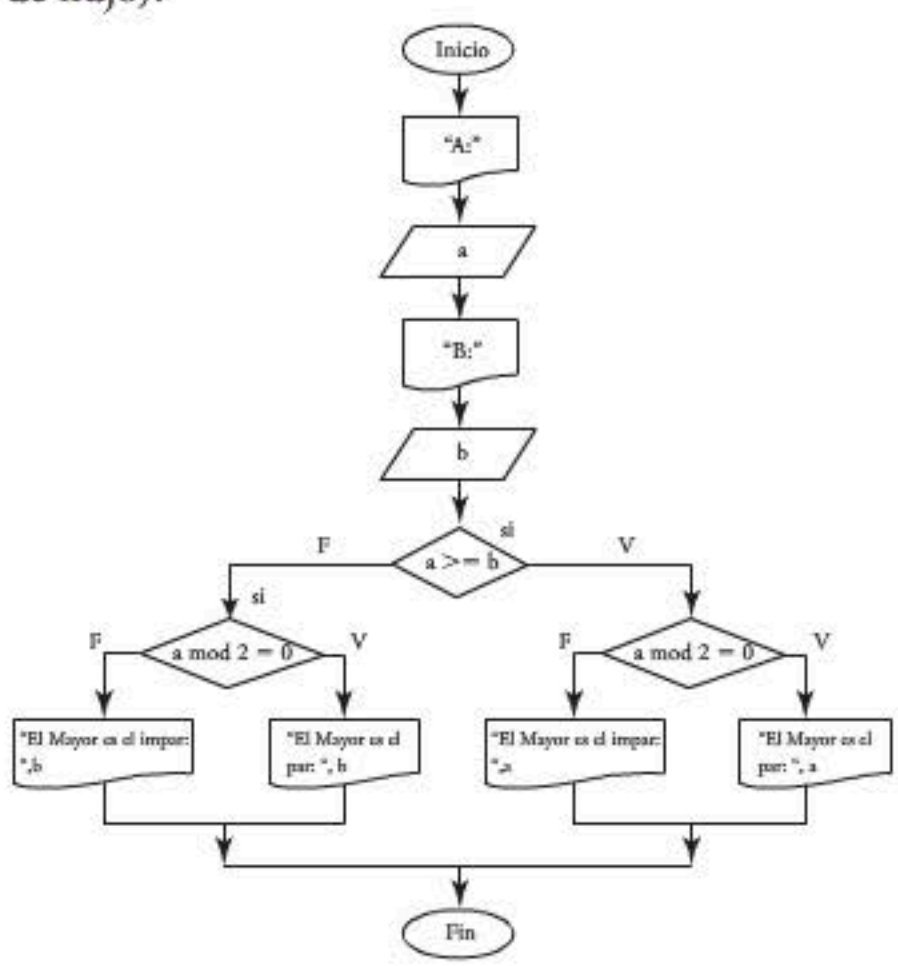
(1 and (1 or 1) and not (0))
(1 and 1 and 1)
(1 and 1)
1
Mensaje: Oaxaca = −9

13.

Respuesta (Pseudocódigo).

```
inicio
entero a,b;
  imprimir "A: ";
  leer a;
  imprimir "B: ";
  leer b;
  si (a>= b)
     si (a mod 2 = = 0)
          imprimir "El mayor es el par: ",a;
         sino
          imprimir "El mayor el impar: ",a;
      sino
        si (b mod 2 = = 0)
           imprimir "El mayor es el par: ",b;
           sino
             imprimir "El mayor es el impar: ", b;
     fin
```

Respuesta (Diagrama de flujo).

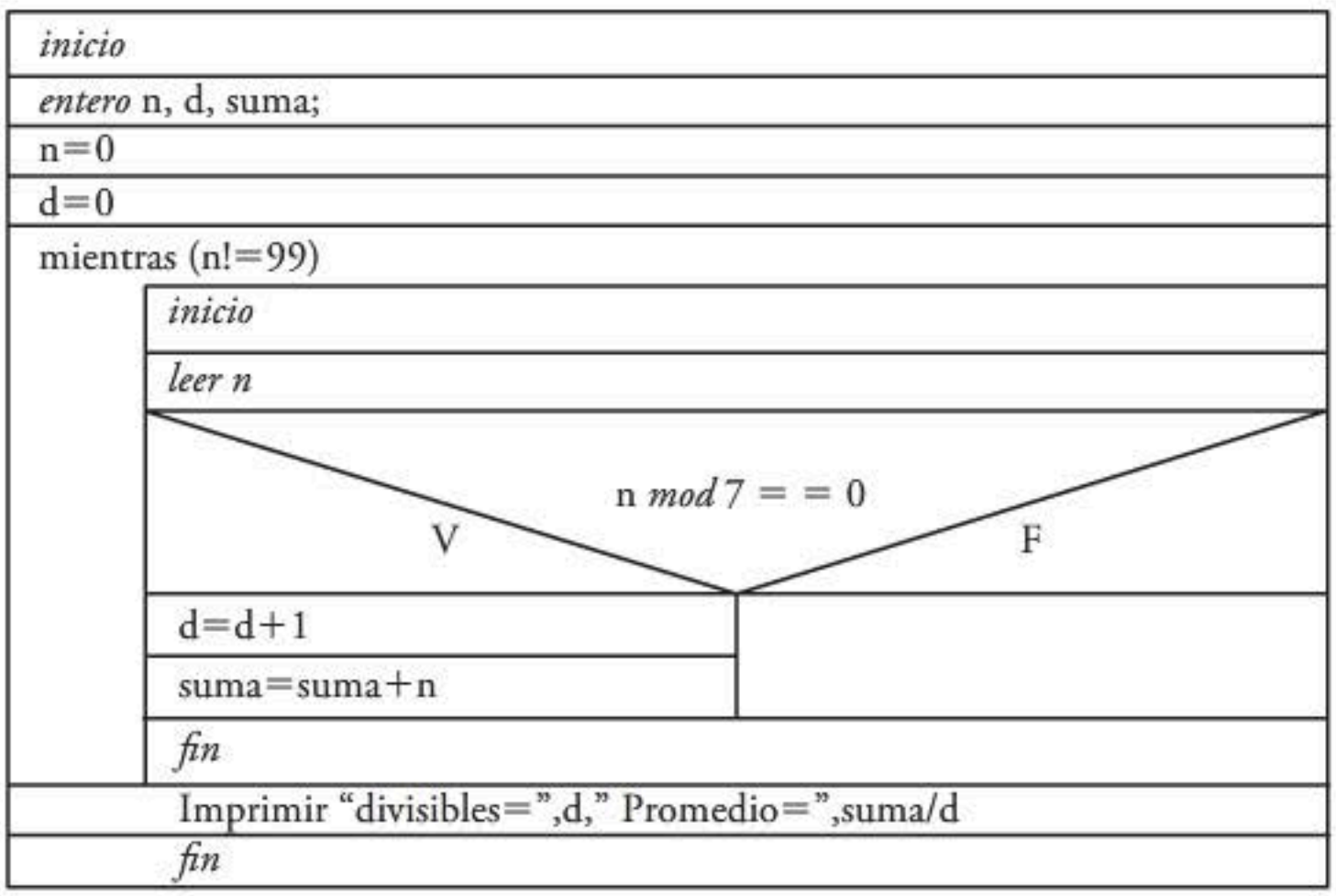


15.–

Respuesta: (En pseudocódigo)

```
inicio
entero n, d, suma;
 imprimir "Dame datos";
   n = 0;
   d = 0;
   mientras (n! = 99)
       inicio
          leer n;
          si (n mod 7 = = 0)
              inicio
                 d = d +1;
                  suma = suma + n;
              fin
         fin
  imprimir "divisibles =", d, ",Promedio = ",suma/d;
 fin
```

Respuesta: (En diagrama N–S)

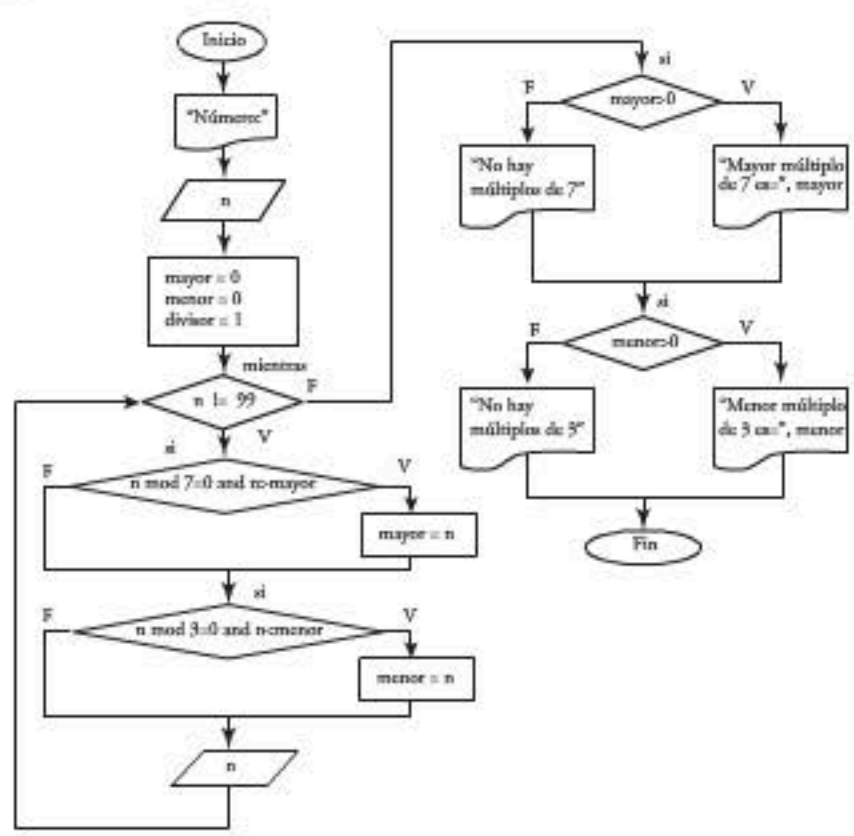


17.–

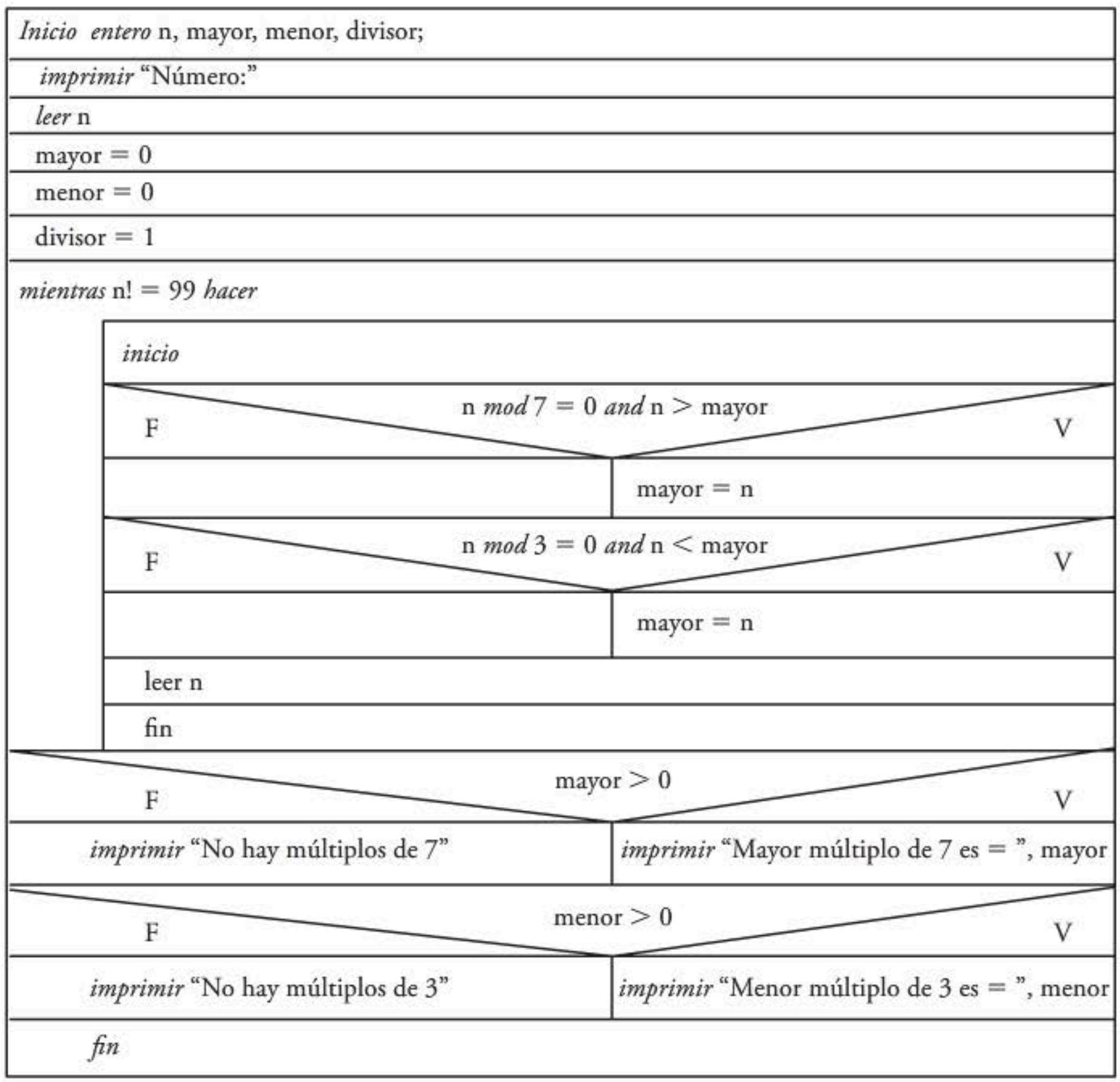
Respuesta (Usando pseudocódigo)

```
inicio
  entero n, mayor, menor, divisor;
  imprimir "Número: ";
  leer n;
  mayor = 0;
  menor = 0;
  divisor =1;
  mientras (n != 99)
    inicio
       si (( n mod 7 = = 0) and (n > mayor))
                 mayor = n;
       si ((n mod 3= = 0) and (n < menor))
                 menor = n;
        leer n;
      fin
       si (mayor > 0)
          imprimir "Mayor múltiplo de 7 es = ", mayor;
        sino
          imprimir "No hay  múltiplos de 7";
       si menor > 0 entonces
          imprimir " Menor múltiplo de 3 es = ",menor;
         sino
           imprimir "No hay múltiplos de 3";
fin
```

Respuesta (Usando diagrama de flujo)



Respuesta (Usando diagramas N−S)



19.

Respuesta (Con el ciclo Mientras)

```
inicio
  entero m, n, men, may, suma, x;
  imprimir "m: ";
  leer m;
  imprimir "n: ";
  leer n;
```

```
    si (m>n)
    inicio
       men=n;
       may = m;
    fin
    sino
     inicio
       men = m;
       may = n;
      fin
     suma = 0;
     x = men + 1;
    mientras (x <= (may - 1))
      inicio
       suma = suma +x*x;
       x = x + 1;
      fin
   imprimir "Suma = ", suma;
  fin
```

Respuesta (Con el ciclo Hacer_mientras)

```
   inicio
   entero m, n, men, may, suma, x;
   imprimir "m: ";
   leer m
   imprimir "n: ";
   leer n;
     si (m>n)
     inicio
        men=n;
        may = m;
     fin
     sino
      inicio
        men = m;
        may = n;
       fin
      suma = 0;
      x = men + 1;
     hacer
       suma = suma +x*x;
        x = x + 1;
       mientras (x > (may-1))
   imprimir "Suma = ", suma;
  fin
```

Respuesta (Con el ciclo Desde)

```
inicio
  entero m, n, men, may, suma, x;
  imprimir "m: ";
  leer m;
  imprimir "n: ";
  leer n;
    si m>n entonces
    inicio
       men=n;
       may = m;
    fin
    sino
     inicio
       men = m;
       may = n;
      fin
     suma = 0;
    desde (x = (men + 1), x<= (may - 1), x=x+1)
      suma = suma +x*x;
   imprimir "Suma = ", suma;
 fin
```

21.–

Respuesta (Usando para ello el ciclo Mientras)

```
inicio
  entero n, divisor, suma;
  imprimir "Número: ";
  leer n;
  suma = 0;
  divisor =1;
  mientras (divisor < n)
    inicio
       si (n mod divisor == 0)
                  suma = suma + divisor;
             divisor = divisor +1;
         fin
       si (suma = = n)
            imprimir "El ", n, " es perfecto";
          sino
             imprimir "El ", n, "no es perfecto";
 fin
```

Respuesta (Usando para ello el ciclo Hacer–Mientras)

```
inicio
  entero n, divisor, suma;
  imprimir "Número: ";
  leer n;
  suma = 0;
  divisor =1;
  hacer
       si (n mod divisor = = 0)
                   suma = suma + divisor;
              divisor = divisor +1;
    mientras (divisor != n)
         si (suma = = n)
            imprimir "El ", n, " es perfecto";
          sino
             imprimir "El ", n, "no es perfecto";
 fin
```

Respuesta (Usando para ello el ciclo Desde)

```
inicio
  entero n, divisor, suma;
  imprimir "Número: ";
  leer n;
  suma = 0;
  desde (divisor = 1; divisor<= (n-1); divisor = divisor+1)
     si (n mod divisor = = 0)
                   suma = suma + divisor;

        si (suma = = n)
            imprimir "El ", n, " es perfecto";
          sino
             imprimir "El ", n, "no es perfecto";
 fin
```

23.

Respuesta (Usando condicionales anidadas)

```
inicio
  entero op, n1, n2;
  imprimir "Operaciones: ";
  imprimir "1.- Suma ";
```

```
  imprimir "2.- Resta ";
  imprimir "3.- Multiplicación ";
  imprimir "4.- División ";
  imprimir "5.- Módulo ";
  imprimir "                    99.- Salir ";

op=0;
Mientras (op!= 99)
 inicio
    imprimir "Operación: ";
    leer op;
    imprimir "a:  ";
    leer n1;
    imprimir "b:  ";
     leer n2;
si (op = =1)
     imprimir "Suma = ", n1 + n2;
      sino
        si (op = = 2)
           imprimir "Resta = ", n1 - n2;
        sino
          si (op = = 3)
            imprimir "Multiplicación = ", n1 * n2;
         sino
           si (op = = 4)
             imprimir "División = ", n1 / n2;
         sino
           si (op = = 5)
              imprimir "Módulo = ", n1 mod n2;
     fin
fin
```

Respuesta (Usando condicionales de opción múltiple)

```
inicio
  entero op, n1, n2;
  imprimir "Operaciones: ";
  imprimir "1.- Suma ";
  imprimir "2.- Resta ";
  imprimir "3.- Multiplicación ";
  imprimir "4.- División ";
  imprimir "5.- Módulo ";
  imprimir "                    99.- Salir ";
```

```
op=0
Mientras (op!= 99)
 inicio
    imprimir "Operación: ";
    leer op;
    imprimir "a:  ";
    leer n1;
    imprimir "b:  ";
     leer n2;
    seleccionar (op)
     inicio
         1: imprimir "Suma = ", n1 + n2;
             romper;
         2: imprimir "Resta = ", n1 - n2;
             romper;
         3: imprimir "Multiplicación = ", n1 * n2;
             romper;
         4: imprimir "División = ", n1 / n2;
             romper;
         5: imprimir "Módulo = ", n1 mod n2;
             romper;
      fin           /* seleccionar */
    fin
fin
```

25.

```
inicio
  real c, incre, fi;
  real pulg, pie, yard;
  imprimir "Inicio: ";
  leer c;
   imprimir "Incremento: ";
   leer incre;
   imprimir "Final: ";
   leer fi;
   imprimir "Centímetros     Pulgadas     Pies     Yardas ";
  mientras (c <= fi)
      inicio
         pulg =  c / 2.54;
         pie = pulg / 12;
         yard = pie / 3;
         imprimir c, pulg, pie, yard;
```

```
        c = c + incre;
      fin
   fin
```

27.

Respuesta (Usando una función para calcular el pago)

```
Inicio   /* Programa principal */
  entero ht;
real ph;

  imprimir "H. Trabajo: ";
  leer ht;
  imprimir "Pago por H.: ";
  leer ph;
    imprimir "Pago de semana = ",sueldo(ht, ph);
 fin

real sueldo(entero ht, real ph)
 val
  real pago;
  inicio
    si (ht > 40)
            pago = 40*ph+(ht - 40)*1.6;
       sino
         pago=ht*ph;
     retornar pago;
  fin
```

Respuesta (Sin utilizar funciones)

```
inicio
  entero ht;
 real, ph;
  imprimir "H. Trabajo: ";
  leer ht;
  imprimir "Pago por H.: ";
  leer ph;
  si (ht > 40)
            pago = 40*ph+(ht - 40)*1.6;
       sino
         pago=ht*ph;
    imprimir "Pago de semana = ",pago;
 fin
```

29.

Respuesta (usando procedimientos)

```
inicio  /* Programa principal */
entero  n, digito;
  lee ( );
  imprimelos ( );
fin

lee ( )   /* Procedimiento lee */
 inicio
    imprimir "Cantidad: ";
    leer n;
 fin

imprimelos ( )  /* Procedimiento imprimelos */
 inicio
      imprimir "Dígitos: ";
      mientras (n > 0)
        inicio
          digito = n mod 10;
          imprimir digito;
          n = n /10;
       fin
  fin
```

Respuestas a capítulo 3

1.

a) Es portable.
b) Gratuito.
c) *Open source* (código abierto).
d) Sencillo para aprender a programar.

3.

a) y = b%(c−5*d)+32/(c%7);
b) y = ((8*b−7%5)/(a*c−d))/((4*c−3)%a);
c) y = (3%(4+b%2)−8)/(9*c−a)+d%3;

5.

a) $y = \dfrac{3 \bmod b - 5}{4(7 \bmod b + 6) + ac} + c \bmod 6$

b) $y = (c * d + 5 \bmod b) \bmod (7a + 8) + \dfrac{3 \bmod c}{2}$

c) $y = \dfrac{4 \bmod a - 3\left(\dfrac{b \bmod 5 + 8}{c + b \bmod 5}\right)}{ac}$

7.

a) y = (3%b−5)/(4*(7%b+6)+a*c)+c%6;
y = (3%4−5)/(4*(7%4+6)+2*(−3))+(−3)%6;
y = (4−5)/(4*(3+6)−6))−3;
y = −1/(4*9−6)−3;
y = −1/(36−6)−3;
y = −1/30−3;
y = 0−3;
y = −3;

b) y = (c*d+5%b)%(7*a+8)+3%c/a;
y = (−3*8+5%4)%(7*2+8)+3%(−3)/2;
y = (−24+1)%(14+8)+0/2;
y = −23%22+0;
y = −1;

c) y = (4%a−3*((b%5+8)/(c+b%5)))/(a*c);
y = (4%2−3*((4%5+8)/(−3+4%5)))/(2*(−3));
y = (0−3*((4+8)/(−3+4)))/(−6);
y = (0−3*(12/1))/(−6);
y = (0−3*12)/(−6);
y = (0−36)/(−6);
y = −36/(−6);
y = 6;

9.

a) y = (4*e−7)/**Math.pow**((3*a*(5−b)),2/3.0)+w/(4−d)−1;
b) y = **Math.sqrt**(3*x−2)/**Math.cos**(40*3.1416/180)−1/(5*x+4);
c) y = **Math.pow(Math.pow**(7*a−b,2),1/3.0)*((1/**Math.sin**(70*3.1416/180))/(b−4*e));

11.

a) $y = 3a(b - 4(c - 2d)) + \frac{5e - 8}{7a}$

b) $y = \frac{\frac{4x^2 - 8}{7}}{6(2x - 14)} + \frac{5 - x^3}{8}$

c) $y = \cos(60^\circ) - \frac{\sqrt{x^2 - 4}}{|7x - 8|}$

13.

```
import java.util.Scanner;

public class ec_segundo_grado {
public static void main(String[] args){

    Scanner leer = new Scanner(System.in);

    double a, b, c, x1, x2;

    System.out.print("a: ");
    a=leer.nextDouble();
    System.out.print("b: ");
    b=leer.nextDouble();
    System.out.print("c: ");
    c=leer.nextDouble();
    x1=(-b+Math.sqrt(Math.pow(b,2)-4*a*c))/(2*a);
    x2=(-b-Math.sqrt(Math.pow(b,2)-4*a*c))/(2*a);
    System.out.printf("x1 = %.4f\n",x1);
    System.out.printf("x2 = %.4f\n",x2);
}
}
```

15.

```
import java.util.Scanner;

public class Eva_polinomio {
public static void main(String[] args){
Scanner leer = new Scanner(System.in);

    double y, x;

        System.out.print("x: ");
    x=leer.nextDouble();
    y=Math.pow(((5*Math.pow(x,4)-6*Math.pow(x,2)+8),1/3.0)/9;
    System.out.printf("y = %.4f\n",y);
}
}
```

17.

```
import java.util.Scanner;
public class funciones_trigonometrica {
public static void main(String[] args){

    Scanner leer = new Scanner(System.in);
    final double PI=3.1416;
```

```
    int ang;
    double x;

    System.out.print("Ángulo (en grados): ");
    ang=leer.nextInt();
    x=ang*PI/180;  //Convierte ang a radianes
    System.out.printf("sen (%d) = %.4f\n", ang,Math.sin(x));
    System.out.printf("cos (%d) = %.4f\n", ang,Math.cos(x));
    System.out.printf("tan (%d) = %.4f\n", ang,Math.tan(x));
    System.out.printf("cot (%d) = %.4f\n", ang,1/Math.tan(x));
    System.out.printf("sec (%d) = %.4f\n", ang,1/Math.cos(x));
    System.out.printf("csc (%d) = %.4f\n", ang,1/Math.sin(x));
}
}
```

Respuestas a capítulo 4

1.

```
import java.util.Scanner;

public class triangulo {
   public static void main(String[] args){
   int a,b,c;

   Scanner leer = new Scanner(System.in);

   System.out.print("a: ");
   a=leer.nextInt();
   System.out.print("b: ");
   b=leer.nextInt();
   System.out.print("c: ");
   c=leer.nextInt();

   if (c>=(a+b))
       System.out.println ("No es un triángulo.");
   else
    if (Math.pow(c,2)==Math.pow(a,2)+Math.pow(b,2))
        System.out.println ("Es un triángulo rectángulo");
    else
    if (Math.pow(c,2) > Math.pow(a,2)+ Math.pow(b,2))
        System.out.println ("Es un triángulo obtusángulo");
    else
    if (Math.pow(c,2) < Math.pow(a,2)+ Math.pow(b,2))
        System.out.println ("Es un triángulo acutángulo");
   }
  }
```

3.

a) con `ifs` anidados:

```
import java.util.Scanner;
public class copias {
  public static void main(String[] args){
   int ncopias;
   double costo;

   Scanner leer = new Scanner(System.in);

   System.out.print("Número de copias: ");
   ncopias=leer.nextInt();

   if (ncopias <= 100)
       costo=ncopias*0.50;
   else
    if (ncopias>100 && ncopias<=200)
       costo=100*0.50+(ncopias-100)*0.35;
   else
    if (ncopias>200 && ncopias<=500)
       costo=100*0.50+100*0.35+(ncopias-200)*0.25;
   else
       costo=100*0.50+100*0.35+300*0.25+(ncopias-500)*0.20;

    System.out.printf ("Importe = $%.2f\n",costo);
    }
   }
```

5.

```
import java.util.Scanner;

public class descuento_tienda {
 public static void main(String[] args){

   double compra, descuento=0, importe;

   Scanner leer = new Scanner(System.in);

   System.out.print("Monto de la compra: ");
   compra =leer.nextFloat();

   if (compra>500 && compra<=1000)
       descuento = (compra-500)*0.05;
   else
```

```
    if (compra > 1000 && compra<=2000)
       descuento = 500*0.05+(compra-1000)*0.08;
   else
     if (compra > 2000)
       descuento = 500*0.05+1000*0.08+(compra-2000)*0.1;

   System.out.printf ("Descuento = $%.2f\n",descuento);
   importe=compra-descuento;
   System.out.printf ("Pago = $%.2f\n",importe);
   }
  }
```

7.

a) Adios manzana.
b) Hola pera.

9.

```
import java.util.Scanner;

public class dias {
   public static void main(String[] args){
   int mes=0,año;

   Scanner leer = new Scanner(System.in);
   while (mes!=999){
   System.out.print("Mes (mm): ");
   mes=leer.nextInt();
   if (mes!=999){
   System.out.print("Año (aaaa): ");
   año=leer.nextInt();

  if (mes==2 && año%4==0)
       System.out.println("Febrero tiene 29 dias");
   else{
       switch (mes){
           case 1: System.out.println("Enero tiene 31 dias");
                   break;
           case 2: System.out.println("Febrero tiene 28 dias");
                   break;
           case 3: System.out.println("Marzo tiene 31 dias");
                   break;
```

```
            case 4: System.out.println("Abril tiene 30 dias");
                    break;
            case 5: System.out.println("Mayo tiene 31 dias");
                    break;
            case 6: System.out.println("Junio tiene 30 dias");
                    break;
            case 7: System.out.println("Julio tiene 31 dias");
                    break;
            case 8: System.out.println("Agosto tiene 31 dias");
                    break;
           case 9: System.out.println("Septiembre tiene 30 dias");
                    break;
            case 10: System.out.println("Octubre tiene 31 dias");
                    break;
           case 11: System.out.println("Noviembre tiene 30 dias");
                    break;
           case 12: System.out.println("Diciembre tiene 31 dias");
                    break;
            default: System.out.println("Mes inexistente");
                    break;
            }
           }
          }
         }
        }
       }
```

11.

```
import java.util.Scanner;

public class sumalos {
 public static void main(String[] args){

   int n, m, sum, tot=0;

   Scanner leer = new Scanner(System.in);

   System.out.print("De: ");
   m=leer.nextInt();
   System.out.print("A: ");
   n=leer.nextInt();
   System.out.print("Suma: ");
   sum=leer.nextInt();
```

```
    System.out.println("Pares cuya suma es "+sum);
    for (int i=m; i<=n; i++){
     for (int j=m; j<=n; j++){
       if (sum==i+j){
         System.out.println(i+"  "+j);
         tot++;
       }
      }
     }
    System.out.println("Total de pares = "+tot);
   }
}
```

13.

a) Respuesta sin usar métodos

```
public class Tablas_de_multiplicar {
public static void main(String[] args){

   int a, b, c;

  System.out.printf("Tabla del 1    Tabla del 2     Tabla del 3");
  System.out.printf("     Tabla del 4     Tabla del 5\n");
  for (b=1; b<=10; b++){
    for (a=1; a<=5; a++){
      c=a*b;
      System.out.printf("%d × %d = %d\t",a,b,c);
    }
    System.out.println();
   }

  System.out.println();
  System.out.printf("Tabla del 6    Tabla del 7     Tabla del 8");
  System.out.printf("     Tabla del 9     Tabla del 10\n");
  for (b=1; b<=10; b++){
    for (a=6; a<=10; a++){
      c=a*b;
      System.out.printf("%d × %d = %d\t",a,b,c);
    }
    System.out.println();
   }
  }
 }
```

b) Respuesta usando un método para desplegar las tablas

```
public class con_metodos {
public static void main(String[] args){
  System.out.printf("Tabla del 1    Tabla del 2     Tabla del 3");
  System.out.printf("     Tabla del 4     Tabla del 5\n");
  pintalas(1,5);
  System.out.println();
  System.out.printf("Tabla del 6    Tabla del 7     Tabla del 8");
  System.out.printf("     Tabla del 9     Tabla del 10\n");
  pintalas(6,10);
  }
   public class con_metodos {
  static void pintalas(int ini, int fin){
    for (int b=1; b<=10; b++){
    for (int a=ini; a<=fin; a++){
      int c=a*b;
      System.out.printf("%d × %d = %d\t",a,b,c);
    }
    System.out.println();
   }
  }
 }
```

15.

a) Respuesta sin métodos.

```
import java.util.Scanner;

public class amigos {
public static void main(String[] args){
Scanner leer = new Scanner(System.in);
 int b, n, c=0, a, i;
 System.out.print ("n: ");
 n=leer.nextInt();

 System.out.println ("Amigos entre 1 y "+n);
 for (a=1; a<n; a++){
   b=0;
   for (i=1; i<a; i++){
    if(a%i==0){
     b=b+i;
    }
   }
```

```
//Divisores de b
   c=0;
   for (i=1; i<b; i++){
    if(b%i==0)
     c=c+i;
    }
    if(a==c)
    System.out.println (a+" y "+b);
  }
 }
}
```

b) Respuesta usando una función para sumar los divisores:

```
import java.util.Scanner;

public class num_amigos {
public static void main(String[] args){
Scanner leer = new Scanner(System.in);
 int b, n, c=0, a, i;

 System.out.print ("n: ");
 n=leer.nextInt();

 System.out.println ("Amigos entre 1 y "+n);
 for (a=1; a<n; a++){
   b=divisores(a);
   c=divisores(b);
   if(a==c)
    System.out.printf ("%6d    y %6d\n",a,b);
 }
}

//Función para sumar los divisores de los números a y b
static int divisores(int x){
   int w=0;
   for (int i=1; i<x; i++){
    if(x%i==0){
     w=w+i;
    }
   }
   return w;
 }
}
```

17.

```
import java.util.Scanner;

public class raros {
public static void main(String[] args){
Scanner leer = new Scanner(System.in);
 int a, n, s, m, res, i;

 System.out.print ("n: ");
 n=leer.nextInt();

 System.out.println ("Números raros entre 1 y "+n);
 for (i=1; i<=n; i++){
   a=i;
   m=inviertelo(a);
   s=sumadigitos(a);

   res=3*m+s;
   if (res==i)
     System.out.println(i);
   }
  }

//Función para invertir el número
static int inviertelo(int a){
 String cad1=String.valueOf(a);   //convierte n a cadena
 StringBuilder invierte=new StringBuilder(cad1);
 String cad2=invierte.reverse().toString();  //invierte cad1
 int w=Integer.parseInt(cad2);// convierte cad2 a numero
 return w;
}

//Función para sumar los digitso
static int sumadigitos(int a){
  int suma=0;
  while (a!=0){
     int digito=a % 10;
      a=a/10;
      suma=suma+digito;
  }
 return suma;
}
}
```

19.

```
import java.util.Scanner;

public class factores {
public static void main(String[] args){
    Scanner leer = new Scanner(System.in);

    int a, b;
    int dosa,tresa,cincosa,sietesa,oncesa,trecesa;
    int dosb,tresb,cincosb,sietesb,oncesb,trecesb;
    System.out.print("a: "); a=leer.nextInt();
    System.out.print("b: "); b=leer.nextInt();

    dosa=factor(a,2);   //Determina si tiene como factor al 2
    tresa=factor(a,3);  //Determina si tiene como factor al 3
    cincosa=factor(a,5);
    sietesa=factor(a,7);
    oncesa=factor(a,11);
    trecesa=factor(a,13);

    System.out.println("Factores: ");
    System.out.print(a+" = ");
    imprimelos(dosa,2);
    imprimelos(tresa,3);
    imprimelos(cincosa,5);
    imprimelos(sietesa,7);
    imprimelos(oncesa,11);
    imprimelos(trecesa,13);
    System.out.println();

    dosb=factor(b,2);
    tresb=factor(b,3);
    cincosb=factor(b,5);
    sietesb=factor(b,7);
    oncesb=factor(b,11);
    trecesb=factor(b,13);

    System.out.print(b+" = ");
    imprimelos(dosb,2);
    imprimelos(tresb,3);
    imprimelos(cincosb,5);
    imprimelos(sietesb,7);
    imprimelos(oncesb,11);
    imprimelos(trecesb,13);
    System.out.println();
```

```
    int mcm=1;
    System.out.print("mcm("+a+","+b+")=");
    mcm=mcm*mincomun(dosa,dosb,2);
    mcm=mcm*mincomun(tresa,tresb,3);
    mcm=mcm*mincomun(cincosa,cincosb,5);
    mcm=mcm*mincomun(sietesa,sietesb,7);
    mcm=mcm*mincomun(oncesa,oncesb,11);
    mcm=mcm*mincomun(trecesa,trecesb,13);
    System.out.println("="+mcm);
    int mcd=1;
    System.out.print("mcm("+a+","+b+")=");
    mcd=mcd*maxcomun(dosa,dosb,2);
    mcd=mcd*maxcomun(tresa,tresb,3);
    mcd=mcd*maxcomun(cincosa,cincosb,5);
    mcd=mcd*maxcomun(sietesa,sietesb,7);
    mcd=mcd*maxcomun(oncesa,oncesb,11);
    mcd=mcd*maxcomun(trecesa,trecesb,13);
    System.out.println("="+mcd);
  }

  //Función para encontrar los factores w(2,3,7..) del número a
  static int factor(int a, int w){
    int result=0, factor=0;
    while (result==0){
      result=a%w;
      if (result==0){
        factor++;
        a=a/w;
      }
    }
    return factor;
  }

  //Método para imprimir los factores de cada número
  static void imprimelos(int x, int f){
    while (x>0){
        System.out.print(f);
        x--;
    }
  }

 //Función para multiplicar los x factores w
  static int acumula(int x, int w){
    int prod=1;
    while (x!=0){
```

```
      prod=prod*w;
      x--;
    }
    return prod;
  }
//Función que determina que factores pertenecen al mcm
 static int mincomun(int x, int y, int f){
    int mcm=1;
      if (x!=0 && y!=0)
            if(x>y){
               imprimelos(x,f);
               mcm=mcm*acumula(x,f);
               }
            else {
                imprimelos(y,f);
                mcm=mcm*acumula(y,f);
                }
       else {
               if (y!=0){
                imprimelos(y,f);
                mcm=mcm*acumula(y,f);
                }
               else if(x!=0){
                imprimelos(x,f);
                mcm=mcm*acumula(x,f);
               }
              }
     return mcm;
}
//Función para determinar que factores pertenecesn al mcd
static int maxcomun(int x, int y, int f){
  int mcd=1;
    if (x!=0 && y!=0)
          if(x<y){
             imprimelos(x,f);
             mcd=mcd*acumula(x,f);
             }
          else {
              imprimelos(y,f);
              mcd=mcd*acumula(y,f);
              }

    return mcd;
 }
}
```

21.

```
import java.util.Scanner;

public class palindroma {
public static void main(String[] args){
Scanner leer = new Scanner(System.in);

 String frase;
 System.out.print ("Dame frase: ");
 frase=leer.nextLine();

 while (!(frase.equals("finalizar"))){
  String frase1=quitaespacios(frase);
  String b=inviertela(frase);
  String frase2=quitaespacios(b);

   if (frase1.equals(frase2))
    System.out.println(b+" => es frase palíndroma");
   else
    System.out.println(b+" => no es frase palíndroma");

  System.out.print ("Dame frase: ");
   frase=leer.nextLine();
   }
  }

//Función usando para invertir la cadena
static String inviertela(String b){

 StringBuilder invierte=new StringBuilder(b);
 String cad2=invierte.reverse().toString();  //invierte la cadena b
 return cad2;
}
//Función para quitar los espacios de la cadena
static String quitaespacios(String a){
 String sinesp="";
 a=a.trim();  //quita los espacios del incio y al final
 int longitud=a.length();  //calcula la lomgitud
 int i=0;
 while(i<longitud-1){
  String letra=a.substring(i,i+1);  //Extrae letra de la cadena
  if (!(letra.equals(" ")))
    sinesp=sinesp.concat(letra); //concatena la letra extraída
```

```
  i++;
 }
 return sinesp;
}
}
```

23.

```
import java.util.Scanner;

public class promedio {
public static void main(String[] args){

   int a,b,c;
   double prom;

   Scanner leer = new Scanner(System.in);

   System.out.print("A: "); a=leer.nextInt();
   System.out.print("B: "); b=leer.nextInt();
   System.out.print("C: "); c=leer.nextInt();

   prom=promedialos(a,b,c);
   System.out.printf("Promedio = %.2f \n",prom);
  }
   //Función para encontrar el promedio de a, b y c
    static double promedialos(int a, int b, int c){
    return (a+b+c)/3.0;
    }
  }
```

25.

```
public class elipse {
 public static void main(String[] args){
 Scanner leer = new Scanner(System.in);

     int suma, puntos=0;

 System.out.print("Imprimir puntos donde (x+y) es: ");
 suma=leer.nextInt();

  System.out.println("Puntos dentro de la elipse cuya suma es
x+y="+suma);
```

```
    for (int x=-4; x<=4; x++)
     for (int y=-5; y<=5; y++){
       if((x*x/16+y*y/25)<=1){
         if (x+y==6){
            System.out.printf("(%d,%d)\n",x,y);
         }
        puntos++;
       }
     }
     System.out.printf("Total dentro de la elipse = %d\n",puntos);
  }
 }
```

27.

```
import java.util.Scanner;

public class aproxima_coseno {
public static void main(String[] args){

  Scanner leer = new Scanner(System.in);

double x, error;

System.out.print("x(grados): ");
x= leer.nextFloat();
x=(Math.PI*x/180);  //Convierte el angulo x a radianes
System.out.print("Error permitido: ");
error= leer.nextFloat();
double resultado=Math.cos(x);
System.out.printf("Coseno (x) real = %.4f\n",resultado);
tabula(x,error,resultado);
}

//Función para tabular la tabla
 static void tabula(double x, double err, double res){
   System.out.println("  n\tCos(x) aprox.\tDiferencia");
   double suma=1, erra;
   int n=2, itera=0;
   erra=Math.abs(res-suma);
   while (erra>=err){
    System.out.printf(" %3d \t %.6f \t %.6f \n",itera,suma,erra);
     if (n%4==0)
       suma+=Math.pow(x,n)/factorial(n);
     else
        suma+=-Math.pow(x,n)/factorial(n);
```

```
      erra=Math.abs(res-suma);
      itera++;
      n+=2;
    }
    System.out.printf(" %3d \t %.6f \t %.4f \n",itera,suma,erra);
}

 //Función para encontrar el factorial de un número
 static int factorial(int n){
      int f=1;
      for (int i=1; i<=n; i++)
        f*=i;
      return f;
 }
}
```

29.

```
import java.util.Scanner;
import java.util.Random;

public class juego_restale {
public static void main(String[] args){

  Scanner leer = new Scanner(System.in);
  Random aleatorio = new Random();

String nombre;
int x, n, c;

System.out.print("Nombre:  ");
nombre= leer.next();
n=aleatorio.nextInt(41)+10;
System.out.println("Número generado: "+n);
while (n!=0){
 System.out.print(nombre+" resta: ");
 x= leer.nextInt();
 while(x<1 || x>3){
  System.out.print("No válido "+nombre+" resta: ");
  x= leer.nextInt();
 }
 n-=x;
 System.out.println("Número= "+n);
 if (n==0)
  System.out.println("GANÓ "+nombre);
```

```
else{
      c=n%4;
      if(c==0)
         c=2;
      n-=c;
      System.out.println("Compu. resta: "+c);
      System.out.println("Número= "+n);
      if(n==0)
        System.out.println("GANÓ la computadora");
     }
    }
   }
  }
```

31.

a) Respuesta: 1 + 2 + 3 +...+ n

```
import java.util.Scanner;

public class sumatoria1 {
public static void main(String[] args){

  Scanner leer = new Scanner(System.in);
 int n, resultado;
System.out.print("Terminos a sumar (n): ");
n=leer.nextInt();
resultado=sumalos(n);
System.out.println("Resultado = "+resultado);
}

//Función recursiva para encontrar la sumatoria.
static int sumalos(int n){
 int w;
  if (n<=1)
    w=1;
  else
    w=sumalos(n-1)+n;
  return w;
}
     }
```

b) Respuesta: $-1 + 2 - 3 + 4 - 5 + \ldots + (-1)n\ n$.

```
import java.util.Scanner;

public class sumalos {
public static void main(String[] args){

  Scanner leer = new Scanner(System.in);

int n, resultado;
System.out.print("Terminos a sumar (n): ");
n=leer.nextInt();
resultado=sumatoria(n);
System.out.println("Resultado = "+resultado);
}

//Función para encontrar n terminos de la sumatoria
static int sumatoria(int n){
 int w;
  if (n<=1)
    w=-1;
  else
    w=sumatoria(n-1)+(int)Math.pow(-1,n)*n;

  return w;
}
}
```

c) Respuesta: $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n$

```
import java.util.Scanner;

public class fac_recursivo {
public static void main(String[] args){
    Scanner leer = new Scanner(System.in);
int n;
System.out.print ("Valor de n: ");
 n=leer.nextInt();
 System.out.printf ("El factorial de %d es= %d\n",n,factorial(n));
}  // Fin del programa principal

//Método para encontrar el factorial de n
    static int factorial(int n){
     int w;

      if (n<=1)
          w = 1;
```

```
      else
          w = factorial(n-1)*n;

        return w;
    }
}
```

d) Respuesta: $1 + 4 + 9 + \ldots + n^2$

```
import java.util.Scanner;
public class suma {
public static void main(String[] args){
  Scanner leer = new Scanner(System.in);
int n, resultado;
System.out.print("Terminos a sumar (n): ");
n=leer.nextInt();
resultado=suma(n);
System.out.println("Resultado = "+resultado);
}
//Función para encontrar n terminos de la sumatoria
static int suma(int n){
 int w;
  if (n<=1)
    w=1;
  else
    w=suma(n-1)+n*n;
  return w;
}
}
```

Respuestas a capítulo 5

1.

```
import java.util.Random;
public class tablita_prob1 {
public static void main(String[] args) {
    Random aleatorio=new Random();
    int numeros[][]=new int[6][5];
    int r,c;
```

```
//  Se llena el arreglo
 System.out.println("Por columna");
for (c=0;c<numeros[0].length;c++){
  for(r=0;r<numeros.length;r++){
    // Se llena columna por columna
    numeros[r][c]=100+aleatorio.nextInt(801);
    System.out.print(numeros[r][c]+"  ");
    }
     System.out.println();
  }

//Impresion por renglón
System.out.println("Por renglon");
for (r=0;r<numeros.length;r++){
 for(c=0;c<numeros[0].length;c++)
    //Se imprime renglón por renglón
    System.out.print(numeros[r][c]+"  ");
    System.out.println();
    }
 }
```

3.

```
public class TriangPascal {
   public static void main(String[] args) {
   //se declara e inicializa el arreglo con ceros
   int cuad[][]=new int [7][13];
   int c=6,r,c1,i;
   cuad[0][c]=1;//Se comienza con 1 en el primer renglón, columna
central
   for(r=1;r<cuad.length;r++){
      c1=c-1; //una columna antes del comienzo en renglon anterior
      i=1;  //contador de números por renglón
      do{
         if(c1>0)  //para no salir de los límites del arreglo
            cuad[r][c1]=cuad[r-1][c1-1];
         if (c1<cuad[r].length-1) //para no salir de los límites
           cuad[r][c1]+=cuad[r-1][c1+1];
         i++;    //contador de numeros por renglon
         c1+=2;  //se avanza  2 celdas en el mismo renglón
      }while(i<=r+1);
      c--;
    }
for(r=0;r<cuad.length;r++){   //IMPRESION DEL ARREGLO
```

```
    for(c=0;c<cuad[0].length;c++)
       if (cuad[r][c]==0)
          System.out.print("   ");
       else if(cuad[r][c]<10)
          System.out.print(" "+cuad[r][c]+" ");
       else
          System.out.print(cuad[r][c]+" ");
    System.out.println();
    }
  }
}
```

5.

```
import java.util.Scanner;
import java.util.Random;
public class Recorrido_Arreglo {
   ////////   METODO MAIN O PRINCIPAL  ////////
    public static void main(String[] args) {
    int numeros[][]=new int[4][7];
    // Llamadas a los métodos
    generar(numeros);
    imprimir_arreglo(numeros);
    imprimir_recorrido(numeros);
   }

 //////// METODO GENERAR ////////
  static void generar(int numeros[][])
  {   // Se crea el objeto con el que se generarán numeros alea-
torios
  Random aleatorio=new Random();
  int r,c;
  for (c=0;c<numeros[0].length;c++)
     for(r=0;r<numeros.length;r++)
        numeros[r][c]=10+aleatorio.nextInt(89);
  }

 ////////  METODO IMPRIMIR_ARREGLO  ////////
  static void imprimir_arreglo(int numeros[][])
  { int r,c;
   System.out.println("Los números generados en el arreglo son: ");
    for (r=0;r<numeros.length;r++){
        for(c=0;c<numeros[0].length;c++)
            //Se imprime renglón por renglón
           System.out.print(numeros[r][c]+"    ");
```

```
        System.out.println();
      }
   }

//////// METODO IMPRIMIR_RECORRIDO  ////////
   static void imprimir_recorrido(int numeros[][])
   {
    Scanner leer=new Scanner(System.in);
    int r,c,veces,i;
    do{// Ciclo asegura un renglón inicial válido
      System.out.print("Renglón inicial? ");
      r=leer.nextInt();
    }while(r<0 || r>numeros.length-1);
    do{  // Ciclo asegura una columna inicial válida
       System.out.print("Columna inicial? ");
       c=leer.nextInt();
      }while(c<0 || r>numeros[0].length-1);
    do{ // Ciclo asegura un valor mayor que cero en veces
       System.out.print("Cuantas veces hacer el recorrido? ");
       veces=leer.nextInt();
      }while(veces<0);
    System.out.println("Los números por los que se pasó son: ");
    for(i=0;i<veces;i++){
       //Se imprime la celda actual
       System.out.print(numeros[r][c]+"  ");
       r++; // para referirnos al renglón de abajo
       //Si el renglón rebasa el valor máximo permitido
       if(r>numeros.length-1)
          r=0;
       c++; // para referirnos a la columna de la derecha
       // Si la columna rebasa el valor máximo permitido
       if(c>numeros[0].length-1)
          c=0;
     }
    System.out.println();
   }
}
```

7.

Escribir un programa que imprima los valores menor y mayor de cada renglón de una matriz irregular previamente inicializada.

9.

```
import java.util.Scanner;
public class Triang_aster {
```

```
 public static void main(String[] args) {
 Scanner leer=new Scanner(System.in);
 int reng,r,c;
 System.out.print("De cuantos renglones quieres la figura de trián-
gulo? ");
 reng=leer.nextInt();
 char T[][]=new char [reng][]; //Se crea la matriz con los renglo-
nes indicados
 for(r=1;r<=T.length;r++){
     T[r-1]=new char[r];  // Se crea la dimensión diferente para
cada renglón
      for (c=0;c<T[r-1].length;c++)  //Se mete '*' en cada celda
existente
        T[r-1][c]='*';
  for (c=0;c<T[r-1].length;c++)  //Se imprime la mtriz
        System.out.print(T[r-1][c]+" ");
     System.out.println();
   }
  }
 }
```

11.

```
import java.util.Scanner;
public class Ordena_nombres {
public static void main(String[] args){
    Scanner leer = new Scanner(System.in);
    int n;

     System.out.print("n: "); n=leer.nextInt();
     String a[]=new String[n]; //define arreglo
     leerlos(a);    //Lee los nombres y los coloca en el arreglo
     ordenalos(a); //Ordena el arrgelo
     System.out.println("Ordenados ");
     imprime(a);   //Imprime el arreglo ordenado
     }

  //Método para leer los elementos y colocarlos en el arreglo
   static void leerlos(String [] a){
    Scanner leer = new Scanner(System.in);
    String nombre;
    System.out.println("Dame los "+a.length+" nombres: ");
```

```
    for (int i=0; i<a.length; i++){
       nombre=leer.nextLine();       //Lee el nombre
       nombre=nombre.toUpperCase(); //lo convierte a mayúsculas
       a[i]=nombre;                  //Lo guarda en el arreglo
       }
    System.out.println();
    }

   //Método ordenar el arreglo en forma ascendente
   static void ordenalos(String [] a){
   int i=1, c, x;
   String t;
   c=a.length;
   while (i>0){
     i=0; c=c-1; x=0;
     while (x<c){
       if (a[x+1].compareTo(a[x])<=0){
         t=a[x];
         a[x]=a[x+1];
         a[x+1]=t;
         i++;
         }
       x++;
     }
    }
   }

   //Método para imprimir el arreglo
   static void imprime(String [] a){
     for (int i=0; i<a.length; i++)
        System.out.println(a[i]);
      }
   }
```

13.

```
import java.util.Scanner;
public class Diferencias {
   public static void main (String args []){
Scanner leer = new Scanner(System.in);
int a=0, b=0;
System.out.print("n: ");
int n = leer.nextInt();
int arreglo[] = new int [n];
```

```
System.out.println("Dame los "+n+" números:");
for (int i=0; i<n; i++)
arreglo [i] = leer.nextInt();

int diferencia = 0;

System.out.println("Diferencia entre consecutivos: ");
for (int x=0; x<(n-1); x++){
System.out.println(Math.abs(arreglo[x]-arreglo[x+1]));
int calculoDiferencia = Math.abs(arreglo[x]-arreglo[x+1]);
if (calculoDiferencia>diferencia)
diferencia = calculoDiferencia;
}
for (int x=0; x<(n-1); x++){
int calculoDiferencia2 = Math.abs(arreglo[x]-arreglo[x+1]);
if (calculoDiferencia2==diferencia){
a = arreglo[x];
b = arreglo[x+1];
}
}
System.out.println("La diferencia mayor es "+diferencia+" y está
entre el "+a+" y el "+b);
    }
}
```

15.

```
import java.util.Random;
import java.util.Scanner;
public class Matriz_transpuesta {
public static void main(String[] args){

 Scanner leer = new Scanner(System.in);
 int fa, ca;

 System.out.print("Filas de A: "); fa=leer.nextInt();
 System.out.print("Columnas de A: "); ca=leer.nextInt();
 int a[][]=new int[fa][ca];  //Crea la matriz
genera(a);      //Genera elementos y los deposita en las celdas
 System.out.println("Matriz A ");
 imprimemat (a);  //Imprime la matriz

 System.out.println("Transpuesta ");
 transp (a,fa,ca);
```

```
    if(fa==ca){
       System.out.println("Matriz cuadrada ");
       diagonal(a);
       }
       periferia(a);
    }
  //Método para generar los datos de la matriz
   static void genera(int [][] x){
   Random aleatorio=new Random();

   for (int i=0; i<x.length; i++)
     for (int j=0; j<x[i].length; j++)
       x[i][j]=aleatorio.nextInt(18)-5;
   }

   //Método para imprimir la matriz
   static void imprimemat (int [][] x){
    for (int i=0; i<x.length; i++){
     for (int j=0; j<x[i].length; j++)
      System.out.printf("%4d",x[i][j]);
      System.out.println();
     }
    System.out.println();
   }

    //Método para imprimir la transpuesta de la matriz
   static void transp (int [][] x, int fx, int cx){
    for (int i=0; i<cx; i++){
     for (int j=0; j<fx; j++)
      System.out.printf("%4d",x[j][i]);
      System.out.println();
     }
    System.out.println();
   }

   //Método para sumar los elementos impares de la diagonal
   static void diagonal(int [][] x){
    int i,suma=0;
    for (i=0; i<x.length; i++){
      if(x[i][i]%2!=0)
        suma=suma+x[i][i];
        }
       System.out.println("Suma de elementos impares en diagonal
="+suma);
    }
```

```
//Método para sumar los elementos multiplos de 7 de la periferia
static void periferia(int [][] x){
int i,suma=0;
int fa=x.length;  //Número de filas de la matriz
 int ca=x[0].length;  //Número de columnas de la matriz
 for (i=0; i<fa; i++){
   //Suma elementos múltiplos de 7 de la columna cero
  if(x[i][0]%7==0)
     suma=suma+x[i][0];

   //Suma elementos múltiplos de 7 de la última columna
   if(x[i][ca-1]%7==0)
     suma=suma+x[i][ca-1];
 }

   for (int j=1; j<fa-1; j++){
   //Suma elementos de primera fila multiplos de 7
   if(x[0][j]%7==0)
     suma=suma+x[0][j];
   //Suma elementos de última fila multiplos de 7
   if(x[fa-1][j]%7==0)
     suma=suma+x[fa-1][j];
   }
     System.out.println("Suma de multiplos de 7 en periferia
="+suma);
 }
}
```

17.

```
import java.util.Scanner;
public class alumnos {
public static void main(String[] args){
    Scanner leer = new Scanner(System.in);
    int n, m, fb, cb, pgpo;

    System.out.print("No. de alumnos: "); n=leer.nextInt();
     System.out.print("No. de materias por alumno: "); m=leer.ne-
xtInt();
    int a[][]=new int[n][m+1];  //Arreglo de calificaciones y pro-
medios
    String nom[]=new String[n]; // Arreglo de nombres
    int tem[]=new int[m+1]; //Arreglo para guardar nombres tempo-
ralmente
                            //mientras se hace el intercambio
```

```
 leedatos(nom,a,n,m);  //Lee la información
pgpo=calcula(a,n,m);  //Calcula promedios y los guarda al final
 ordenalos(nom,a,tem,n,m);  //Ordena la información
 imprimemat (nom,a,n,m,pgpo); //Imprime la información
 }  //fin de programa principal


//Método para leer la información
 static void leedatos(String [] nom, int [][] a, int n, int m){
 Scanner leer = new Scanner(System.in);
 for (int i=0; i<n; i++){
   System.out.print("Nombre: ");
   nom[i]=leer.next();
   for (int j=0; j<m; j++){
     System.out.print("Calif "+(j+1)+":");
     a[i][j]=leer.nextInt();
     }
  }
 }

//Método para calcular el promedio de los alumnos
 static int calcula (int [][] a, int n, int m){
  int sgpo=0, pgpo=0;
  int salum, palum=0;
  int i,j=0;
  for (i=0; i<n; i++){
     salum=0;
   for (j=0; j<m; j++)
    salum+=a[i][j];

    a[i][j]=salum/m;  //Guarda el promedio de cada alumno
    sgpo+=a[i][j];
   }
  pgpo=sgpo/n;
  return pgpo;  //Regresa el promedio del grupo
 }

 //Método para imprimir la tabla
 static void imprimemat (String [] nom,int [][] a,int n,int m,int
pgpo){
  int i,j;
  System.out.print("Nombre          ");
  for (int k=1; k<=m; k++)
    System.out.print("Calif "+k+"  ");
  System.out.println("Promedio");
  for (i=0; i<n; i++){
```

```
      System.out.printf("%-10S\t",nom[i]);
      for (j=0; j<m; j++)
       System.out.printf("%5d\t",a[i][j]);
       System.out.printf("%5d\n",a[i][j]);
      }
     System.out.println("Prom. Gpo.="+pgpo);
    }
   //Método ordenar los nombres y sus respectivas calificaciones
   static void ordenalos(String[] nom,int [][] a,int [] temp,int
n,int m){
    int i=1, c, x;
    String t;
    c=n;
    while (i>0){
      i=0;   //intercambios realizados
      c=c-1; //comparaciones a realizar
      x=0;   //posición de los elementos
      while (x<c){
       if (nom[x+1].compareTo(nom[x])<=0){
         //Intercambia nombres
         t=nom[x];
          nom[x]=nom[x+1];
          nom[x+1]=t;
          //guarda las calificaciones temporalmente
          for (int d=0; d<=m; d++)
             temp[d]=a[x][d];

           //pasa las calificaciones una posición arriba
          for (int d=0; d<=m; d++)
             a[x][d]=a[x+1][d];

           for (int d=0; d<=m; d++) //copia las calificaciones de
temporal
            a[x+1][d]=temp[d];

          i++;  //incrementa intercambios
          }
        x++;  //incrementa posición de elemento
      }
     }
    }
   }
```

19.

```
import java.util.Random;
import java.util.Scanner;

public class Desviacion_estandar {
public static void main(String[] args){
    Scanner leer = new Scanner(System.in);
    int n, w, x;
    float promedio;

    System.out.print("n: "); n=leer.nextInt();
    System.out.print("w: "); w=leer.nextInt();
    System.out.print("x: "); x=leer.nextInt();
    int a[]=new int[n];  //Crea arreglo de n lugares
    genera(a,w,x);  //Genera los datos y los guarda
    promedio=media(a); //Encuentra la media aritmética
    imprime(a,promedio); // Imprime la información
    }


  //Método para generar los datos del arreglo e imprimirlos
    static void genera(int [] a, int w, int x){
    Random aleatorio=new Random();
    System.out.println("Datos generados entre "+w+" y "+x+":");
    for (int i=0; i<a.length; i++){
       a[i]=aleatorio.nextInt(x-w+1)+w;
       System.out.print(a[i]+" ");
       }
    System.out.println();
    }
     //Método para calcular la media de los datos
    static float media(int [] a){
    int suma=0;
    for (int i=0; i<a.length; i++)
      suma+=a[i];
    return (float)suma/a.length;
    }

  //Método para imprimir la tabla
    static void imprime(int [] a,float promedio){
      int suma=0;
      double sdif=0;
      //Iprime la tabla
      System.out.println("Obs\tDatos\t\t(xi-xmed)2");
      for (int i=0; i<a.length; i++){
         double dif=Math.pow((a[i]-promedio), 2);
```

```
System.out.printf("%3d\t%5d\t\t%.4f\n",(i+1),a[i],dif);
        suma+=a[i];
        sdif+=dif;
     }
    System.out.printf("Suma=   %5d\t\t%.4f\n",suma,sdif);
    System.out.println();
    System.out.printf("Xmedia= %.4f\n",promedio);
    double ds=Math.pow((sdif/(a.length-1)),1/2.0);
    System.out.printf("Desv. Estándar= %.4f\n",ds);
   }
 }
```

21.

```
import java.util.Random;

public class RengPar_RengNon {
static int arr[][]; // matriz donde se almacenarán los números

//Metodo para generar un número impar o par, según el parámetro (c)

static int generar_num(char c){
Random aleat=new Random();  // se crea el objeto para generar alea-
torios
int n=0;

switch(c){
    case 'i': // Si el parámetro es 'i' se genera un impar
             do{
              n=aleat.nextInt(50)+50; //Genera un numero de 50 a 99
                }while(n%2==0);
              break;

     case 'p': // Si el parámetro es 'p' se genera un par
             do{
              n=aleat.nextInt(50)+50; // Genera un num de 50 a 99
                }while(n%2!=0);
              break;
         }
   return n;  // Se devuelve el número generado
   }

// Método para llenar la matriz con números
    static void meter_nums_en_matriz(int x){
    int columnas=x,r,c;
   arr=new int[x][];  // Se definen solo los renglones del arreglo
   System.out.println("El arreglo tiene "+arr.length+" renglones");
```

```
    //en cada iteración columnas se incrementa en 2
    for(r=0; r<arr.length; r++,columnas+=2)
      arr[r]=new int [columnas]; //se definen las columnas de cada
renglón
    for(r=0; r<arr.length; r++){
        for(c=0; c<arr[r].length; c++)
            if(r%2==0)  //Si es renglón par
             arr[r][c]=generar_num('p');   //se mete aleatorio par
            else     // Si es renglón impar
            arr[r][c]=generar_num('i');   //se mete aleatorio impar
    }
    imprimir();  // Se hace el llamado al método imprimir
    }
    //método que imprime la matriz renglón por renglón
    static void imprimir(){
    int r,c;
    System.out.println("En la matriz quedaron los valores ");
    for(r=0; r<arr.length; r++){
            for(c=0; c<arr[r].length; c++)
              System.out.print(arr[r][c]+"  ");
            System.out.println();
        }
    }
//Programa principal
public static void main(String[] args) {
 //Comienza la ejecución en main, llamando al método meter_nums_
en_matriz
 meter_nums_en_matriz(4);  // Se meten los aleatorios en la matriz
de 4 renglones
}
}
```

23.

```
public class Transnum {
public static void main(String[] args) {
  char A[][]={{'1','5','3','8','6','2'},{'4','0','1'}};
  String s="";
  int r, c, numero;
  for(r=0; r<A.length; r++){
    s="";
    for(c=0; c<A[r].length; c++)
     s+=A[r][c];  // en la variable s se concatenan los caracteres
```

```
    System.out.println("La cadena es  "+s);
    numero=Integer.parseInt(s);  //Se convierte la cadena a entero
    System.out.println("Convertido a número y multiplicado por 2
es  "+numero*2);
    }
   }
 }
```

Respuestas a capítulo 6

1. Responda las siguientes preguntas.

   a) Simula 67, Smalltalk y C++.
   b) Modelación comprensible, reutilización de código, flexibilidad y escalabilidad.
   c) Herencia
   d) Abstracción
   e) Polimorfismo
   f) Encapsulamiento
   g) Ocultamiento de información
   h) Modularidad

3.

   a) Objeto *sonaja*
   marca: Fisher Price
   tamaño: 10 cms
   edad recomendada: 3 a 10 meses

   b) Objeto *bermudas*
   talla: 30
   color: azul
   material: mezclilla

   c) Objeto *memoria usb*
   Capacidad: 4 GB
   espacio disponible: 3 GB
   forma: hombre araña

   d) Objeto *revista*
   nombre: selecciones
   periodicidad: mensual
   número de hojas: 100

   e) Objeto *fritura*
   sabor: salado
   fecha de caducidad: 25/dic/2014
   nombre: chetos

5.

```
import java.util.Scanner;

public class RADIO {
    static Scanner leer=new Scanner(System.in);
private  //atributos privados
        int vol, estac, encen;
//********************** método constructor
    RADIO(){  //constructor
    encen=0;  //al crear el objeto está apagado encen=0
    vol=3;  // Se inicia con un volumen de 3
    estac=1;  // se inicia en la estación 1
```

```
    System.out.println("Radio Apagada");
}
// ********************   método funcionar DE ACCESO PÚBLICO
public void funcionar(){
    int menu=0;
    do{
    System.out.println(" MENU \n1.-Encender2.-Apagar3.-
                        Sintonizar\n4.-Subir Volumen5.-Bajar
                        Volumen6.-Abandonar");
    System.out.println("Que accion desea ejecutar?");
    menu=leer.nextInt();
    switch(menu){   //Se ejecutan los métodos según la elección
      case 1: encender();break;
      case 2: apagar();break;
      case 3: sintonizar();break;
      case 4: subir_vol();break;
      case 5: bajar_vol();break;
      case 6: System.out.println("ADIOS"); }
    }while(menu!=6) ;
}
// ********************** método bajar_vol  de acceso privado
private void bajar_vol(){
    if (encen==0){    //si no está encendida
        System.out.println("IMPOSIBLE, Radio Apagada");
        return;}
    int opc=1;
    while(vol>0 && opc==1) {  //mientras sea posible bajar volumen
       pintar_lineas();
       System.out.print("\nbajar mas?  1=SI   2=NO");
       opc=leer.nextInt();
       if (opc==1)
          vol--; }
    if(vol==0)
        pintar_lineas();
}
// ********************** metodo subir_vol  de acceso privado
private void subir_vol(){
    if (encen==0){
        System.out.println("IMPOSIBLE, Radio Apagada");
        return;}
    int opc=1;
    while(vol<10 && opc==1) { //mientras sea posible subir volumen
       pintar_lineas();
       System.out.print("\nsubir mas?  1=SI   2=NO");
      opc=leer.nextInt();
       if (opc==1)
          vol++; }
    if(vol==10)
        pintar_lineas();
```

```
}
// ********************** método encender  de acceso privado
private void encender(){
    if (encen==1)
       System.out.println("Ya está encendida");
    else {
        System.out.println("Se acaba de encender");
        encen=1;}
}
// ********************** metodo apagar  de acceso privado
private void apagar(){
    if (encen==0)
       System.out.println("Ya Apagada");
    else {
        System.out.println("Se acaba de apagar");
         encen=0;}
}

// ********************** metodo sintonizar  de acceso privado
private void sintonizar(){
    if (encen==0){
       System.out.println("IMPOSIBLE, Radio Apagada");
       return;}
    int opc=1,est;
    while(opc==1) {  //mientras se desee cambiar de estación
       System.out.print("\nQué estación?  1  a la 10 ");
       est=leer.nextInt();
       if (est>=1 && est<=10)
          estac=est;
       pintar_lineas();
       System.out.print("  Otra estación ?  1=SI   2=NO");
       opc=leer.nextInt();}
}

// ********************** método pintar líneas  de acceso privado
private void pintar_lineas(){
    int i=0;
    if (vol==10)
      System.out.print("Volumen máximo ");
    else if (vol==0)
      System.out.print("Sin volumen ");
    else
      System.out.print("Volumen ");
     while(i<vol) {    //se imprimen líneas para simular nivel de
volumen
      System.out.print ("|");
      i++;
      }
   System.out.print(" Estación "+ estac);
```

```
}

}

//////////////////// CLASE PRINCIPAL DONDE ES CREADO EL OBJETO
public class RAD_PRIN {
   public static void main(String[] args) {
    RADIO Sony=new RADIO();  // se crea el objeto Sony
    // se envía el mensaje funcionar al objeto Sony
     Sony.funcionar(); //método funcionar es el único accesible
desde main
  }
}
```

7.

```
package ahorro_alc;

class AHORRO{
    private int cantidad;
    AHORRO(int inicial){
        if (inicial>=0) // Se inicializa atributo con al parámetro
            cantidad=inicial;
        else  // Si el parámetro fue negativo
            cantidad=150;
        }
 ////////////  METODO ahorrar
   public void ahorrar(int cant){
       if (cant <=0)  // Si el parámetro es negativo o cero
           return;   // salimos del metodo
       else
           cantidad+=cant;  //se sumacant al atributo
    }
 ////////////  METODO retirar
   public int retirar(int cant){
   int x=0;
// Si lo que se intenta retirar es mayor que lo que se tiene en ahorro
  if (cant>cantidad){
      x=cantidad;  // x es lo máximo se puede retirar
      cantidad=0;  // ahorro queda en cero
      return x;}  // Se devuelve  lo que se retiró
   else if (cant>0){
     cantidad -=cant;  //Se resta el retiro a la cantidad
      return cant;}     //Se devuelve  lo que se retiró
   else
```

```
        return 0;  // Se devuelve cero, no se pudo hacer retiro
    }
//////////// METODO mostrar_ahorro
 public void mostrar_ahorro(){
  System.out.println("Al momento tienes un ahorro de $"+cantidad+"
pesos");
    }
 }
//////////// CLASE PRINCIPAL
public class Main {
   public static void main(String[] args) {
   // Se crea el objeto con 100 de ahorro
   AHORRO alcancia=new AHORRO(100);
   alcancia.mostrar_ahorro();  //Se muestra el ahorro actual
  // Se retiran 50 pesos
   System.out.println("Has retirado " + alcancia.retirar(50)+ "
pesos");
  alcancia.mostrar_ahorro();  //Se muestra el ahorro actual
  alcancia.ahorrar(235);  // Se ahorran 235 pesos
  alcancia.mostrar_ahorro();  //Se muestra el ahorro actual
  //Se intentan retirar 300 pesos
   System.out.println("Has retirado " + alcancia.retirar(300)+ "
pesos");
  alcancia.mostrar_ahorro();  //Se muestra el ahorro actual
    }
}
```

9.

```
package libreta;
import java.util.Scanner;

//Atributos de la clase, conocidos por todos los métodos
public class LIBRETA {
private int num_hojas,hojas_limpias,hojas_utilizadas;
private char A[]=new char [150];

Scanner leer=new Scanner  (System.in); //objeto para leer datos

//método constructor
LIBRETA (int hojas){
int i;  // i solo se conoce en este método
  //Se acepta un máximo de 150 hojas
  if(hojas>=10 && hojas<=150)
     num_hojas=hojas_limpias=hojas;
  else
```

```
     num_hojas=hojas_limpias=50;  //50 hojas por default
  hojas_utilizadas=0;
  for(i=0; i<150; i++)
      A[i]='0';  //A todas las hojas se les pone la marca de lim-
pias
 }

void arrancar_hojas(){
int h;  // h solo se conoce en este método
do{
   System.out.print("Que hoja quieres arrancar? 1- "+ num_hojas+"
");
   h=leer.nextInt();
   //h dentro del rango de hojas existentes
   if(h>0 && h<=num_hojas){
         //utilizada sin arrancar
         if(A[h-1]!='A' && A[h-1]=='x'){
            A[h-1]='A';            //se marca como arrancada
           hojas_utilizadas--;}  // se decrementan las utilizadas
        else if(A[h-1]!='A' && A[h-1]=='0'){  //limpia sin arrancar
            A[h-1]='A';
            hojas_limpias--;}   //se decrementan las limpias
         else
            System.out.println("La hoja "+h+" ya fue arrancada");
      }
    else
      System.out.println("No existe tal hoja esta libreta solo era
de "+num_hojas+" hojas");
      }while(h<=num_hojas && h>0);
  }

void utilizar_hojas(){
int x;
  do{
     System.out.print("Cual hoja utilizaras?");
     x=leer.nextInt();
     //x dentro del rango
     if(x>0 && x<=num_hojas){
        //hoja arrancada
       if(A[x-1]=='A')
           System.out.println("Ya está arrancada");
        else if(A[x-1]=='x')  //hoja utilizada
          System.out.println("Ya está utilizada\n");
        else {
           A[x-1]='x';  //Se marca como utilizada
```

```
            hojas_utilizadas++;
            hojas_limpias--;}
      }
      else
          System.out.println("No existe tal hoja esta libreta solo
era de "+num_hojas+" hojas");
     }while(x<=num_hojas && x>0);
}

void mostrar_caract(){
int i,a=0;  //variables locales al método
for(i=0;i<num_hojas;i++)
  if( A[i]=='A')  //Las que tienen la marca de arrancada se cuentan
       a++;
System.out.println( "Libreta de:"+num_hojas+" hojas, "+hojas_lim-
pias+
" están limpias"+","+ hojas_utilizadas+" están utilizadas y "+a+
" están arrancadas");
}

void agregar_hojas(int e){
  int x;
   if(e<0)   //e es el número de hojas a agregar
         return;  //se sale del método
      else
       //agregar excedería el límite de 150
       if(num_hojas+e>150) {
         x=150-num_hojas;  //x tiene las máximas que se pueden agregar
         System.out.print("\nSe pudieron agregar "+x+" hojas");
         num_hojas=150;   //se agregan hasta llegar al limite
         hojas_limpias+=x; }
      else {
         num_hojas+=e;  //Se agregan las que se indicaron
         hojas_limpias+=e;}
}
} //LAVE FINAL DE LA CLASE

package libreta;
import java.util.Scanner;
public class Prin_Libreta {
public static void main(String[] args) {
 Scanner leer=new Scanner(System.in);

 LIBRETA scribe=new LIBRETA(10);  //Se crea una libreta de 10 hojas
 int x;
```

```
 System.out.println("ANTES de enviar mensajes al objeto se escri-
be, su estado es: ");
    scribe.mostrar_caract();
    scribe.arrancar_hojas();
    scribe.utilizar_hojas();
    System.out.print("Cuantas deseas agregar?");
    x=leer.nextInt();
    scribe.agregar_hojas(x);
System.out.println("DESPUES de enviar mensajes al objeto se escri-
be, su estado es: ");
 scribe.mostrar_caract();
 }
}
```

11.

```
package fracciones;
import java.util.Scanner;

class Fracciones{
// atributos para numeradores y denominadores
   private int num1,num2,den1,den2,num,comun_denom;
   Scanner leer=new Scanner (System.in);

// Método para ingresar los valores de las fracciones
public void ingresar_fracciones(){
System.out.print("Valor del primer numerador: ");
  num1=leer.nextInt();
  do {
      System.out.print("Valor del primer denominador: ");
      den1=leer.nextInt();
     }while(den1<=0);       //denominador no puede ser negativo

 System.out.print("Valor del segundo numerador: ");
  num2=leer.nextInt();
   do {
       System.out.print("Valor del segundo denominador: ");
       den2=leer.nextInt();
       }while(den2<=0);     //denominador no puede ser negativo

         System.out.println("Las   fracciones   ingresadas   son:
"+num1+"/"+den1+ " y "+num2+"/"+den2);
 }
```

```
//Método sumar
public void sumar(){
 comun_denom=den1*den2;
 num=num1*den2 + num2*den1;
  System.out.print(num1+"/"+den1+" + "+num2+"/"+den2+" = "+num +
"/"+comun_denom);
 simplificar();
 }

//Método restar
public void restar(){
  comun_denom=den1*den2;
  num=(num1*den2 - num2*den1);
  System.out.print(num1+"/"+den1+" - "+num2+"/"+den2+" = "+num
                          +"/"+comun_denom);
  simplificar();
 }

//Método multiplicar
public void multiplicar(){
 comun_denom=den1*den2;
 num=num1*num2;
 System.out.print(num1+"/"+den1+" * "+num2+"/"+den2+" = "+
                          num+"/"+comun_denom);
 simplificar();
    }

//Método dividir
  public void dividir(){
   if (num2==0){
     System.out.println("ERROR DIVISION POR CERO no es posible
                    dividir "+num1+"/"+den1+ " / "+num2+"/"+den2);
     return;
     }

     num=num1*den2;
     comun_denom= den1*num2;
     if((num1<0 && num2<0)|| num2<0){
       num*=-1;comun_denom*=-1;
     }
     System.out.print(num1+"/"+den1+" / "+num2+"/"+den2+" = "+
                          num+"/"+comun_denom);
     simplificar();
    }
//Método simplificar protegido, no accesible fuera de la clase
 private void simplificar() {
```

```
boolean signo=false;
System.out.print("  que simplificado es ");
if(num==0){
    System.out.println("0");
    return;
    }
if (num<0) {
   signo=true;  //se enciende bandera del signo
   num*=-1;     // num se hace positivo para simplificarlo
   }
int entero=num/comun_denom;           // se saca parte entera
int numeradornuevo=num%comun_denom;  //se saca parte fraccionaria

 if(entero>=1) {
    if(signo==true) //si era negativo se le devuelve su signo
       entero*=-1;
       System.out.print(entero+" ");
       signo=false;   // se apaga bandera de signo
       }
for(int x=numeradornuevo;x>=2;x--){
  // se reduce parte fraccionaria
  if(numeradornuevo%x==0 && comun_denom%x==0){
     numeradornuevo/=x;
     comun_denom/=x;
     }
    }
// si resulta parte fraccionaria se checa la bandera de signo
if(numeradornuevo>=1) {
   if(signo==true)
     numeradornuevo*=-1;  // se devuelve su signo
     System.out.print(" "+numeradornuevo+"/"+comun_denom); }
     System.out.println();
     }
   }
```

13.

```
//Definición de la clase base
class Producto{

// Atributos de la clase
protected String descripcion,unidad_vta;
protected int existencia;
protected float precio_u,ingresos;
```

```
//Único método de la clase
protected void mostrar_existencias(){
System.out.println("Hay "+existencia+ " "+unidad_vta +" "+descrip-
cion+ " en existencia ");
}

//Método fijar precio
protected void fijar_precio(float p){
precio_u=p;
}

//Método vender
protected void vender(float cantidad){
  if (cantidad<=existencia){
    existencia-=cantidad;
    ingresos+=cantidad*precio_u;
    }
 }

//Método mostrar_ingresos
protected void mostrar_ingresos(){
System.out.println("Hay "+existencia+" "+unidad_vta+" "+descrip-
cion +" en inventario y un total de $"+ingresos+" en ingresos ");
}
}  //Fin de la clase base


//Subclase Resistol: recibe como herencia los elementos de la cla-
se //Producto
class Resistol extends Producto{

//Método Constructor de la clase
Resistol(float pu, String descrip, String uv, int exist){
precio_u=pu;
descripcion=descrip;
unidad_vta=uv;
ingresos=0;
existencia=exist;
}

//Método surtir
boolean surtir(int cajas, float precioxcaja){
    if (ingresos>=cajas*precioxcaja){
        ingresos-=cajas*precioxcaja;
       existencia+=cajas*10; //cada caja trae 10 unidades de re-
sistol
```

```
        return true;
    }
    else {
       System.out.println("No hay suficientes ingresos para surtir
"+ cajas+ " "+unidad_vta +" "+descripcion);
    return false;
    }
 }
} //Fin de la subclase Resistol

//Subclase Clip: recibe como herencia los elementos de la clase
Producto
class Clip extends Producto{

//Metodo Constructor de la clase
Clip(float pu, String descrip, String uv, int exist){
    precio_u=pu;
    descripcion=descrip;
    unidad_vta=uv;
    ingresos=0;
    existencia=exist;
    }

//Método surtir
boolean surtir(int cajas, float precioxcaja){
    if (ingresos>=cajas*precioxcaja){
        ingresos-=cajas*precioxcaja;
        existencia+=cajas;
        return true;
    }
    else {
       System.out.println("No hay suficientes ingresos para surtir
"+ cajas+" "+unidad_vta +" de "+descripcion);
        return false;
    }
}

}//Fin de la subclase Clip
```

15.

```
import java.util.Scanner;

class Sobrecar {
```

```
//Primera versión del método sumar
public int sumar(int a, int b, int c){
  System.out.print("Se sumaron los valores "+a+", "+b+", "+c);
  return a+b+c;
  }

//Segunda versión del método sumar
public int sumar(){
  Scanner leer=new Scanner(System.in);
  int i=1, suma=0;
  while(i!=0){
     System.out.print("Dame un valor entero o cero para terminar:
");
    i=leer.nextInt();
    suma+=i;
    }
     return suma;
   }

//Tercera versión del método sumar
public String sumar(String a,String b, String c, String d){
 System.out.print("Se sumaron los valores "+a+", "+b+", "+c+",
"+d);
 String suma=a.concat(b).concat(c).concat(d);
 return suma;
 }
}

public class Sobrecarga{
public static void main(String[] args){
//Creacion del objeto
Sobrecar objeto=new Sobrecar();

System.out.print("Versión para varios valores enteros\n");
System.out.println("La suma fue "+objeto.sumar());
//llamada a sumar
System.out.print("Versión para 3 valores enteros, ");
System.out.println(" la suma fue "+objeto.sumar(23,45,56));
//llamada a sumar
System.out.print("Versión para concatenar, ");

// llamada a sumar
System.out.println(" la suma fue "+objeto.sumar("hola ","soy ","tu
","amigo"));
}
}
```

17.

```
//Clase que contiene las distintas operaciones que se realizan de
// acuerdo al número de parámetros.
package sobrecarga_operaciones;
public class operaciones {
//Método sin parámetros
public void operacion(){
System.out.println("Hola");
}

//Método con un parametro, solamente lo imprime.
public void operacion(int a){
System.out.println(a);
}

 //Con dos parámetros, los imprime e imprime la suma.
 public void operacion(int a, int b){
 System.out.printf("%d  %d  Suma=%d\n",a,b,(a+b));
 }

//Con tres parámetros. Los imprime, suma los dos primeros y le
//resta el tercero.
 public void operacion(int a, int b, int c){
System.out.printf("%d %d %d Suma=%d Resta=%d\n",a,b,c,(a+b),(a+b-
c));
 }

 //Con cuatro parámetros. Los imprime, suma los dos primeros, le
 //resta el tercero y al resultado lo multiplica por el cuarto.
 public void operacion(int a, int b, int c, int d){
 System.out.printf("%d %d %d %d Suma=%d Resta=%d multiplica=%d\
n",a,b,c,d, (a+b),(a+b-c),(a+b-c)*d);
 }

//Con cuatro parámetros. Los imprime, suma los dos primeros, le
//resta el tercero, al resultado lo multiplica por el cuarto y
//divide entre el quinto.
public void operacion(int a, int b, int c, int d, int e){
System.out.printf("%d %d %d %d %d Suma=%d Resta=%d multiplica=%d"
     +"  Divide=%.2f\n",a,b,c,d,e,(a+b),(a+b-c),(a+b-c)*d,(float)
((a+b-c)*d)/e);
}
}
```

```
//Clase que contiene el programa principal
package sobrecarga_operaciones;
public class Sobrecarga_operaciones {
public static void main(String[] args) {

   //Se crea el objeto opera de la clase operaciones
   operaciones opera=new operaciones();

   opera.operacion(30,56);
   opera.operacion(7,-4,8,10,8);
   opera.operacion();
   opera.operacion(14,3,50,4);
   opera.operacion(-5,13,16);
   opera.operacion(17);
   opera.operacion(23,2,7,16,18);
    }
}
```

19.

```
package sobre_carga;

//Clase que contiene los métodos sobrecargados.
public class diferente {

 //Método que recibe un entero y lo eleva al cubo
   static void encuentra(int x){
   System.out.println(x+" Al cubo=  "+(int)Math.pow(x,3));
   }

   //Método que recibe un número el radio del círculo y calcula
   //el área
   static void encuentra(Double x){
   System.out.println(x+"  área = "+3.1416*Math.pow(x,2));
   }

   //Método que recibe una cadena y encuentra el número de vocales
   //de la misma
  static void encuentra(String x){
    int voc=0;
    char letra;
    for (int i=0; i<x.length(); i++){
     letra=x.charAt(i);
     switch (letra){
         case 'a':
         case 'á':
         case 'e':
```

```
        case 'é':
        case 'i':
        case 'í':
        case 'o':
        case 'ó':
        case 'u':
        case 'ú':
          voc++;
        break;
    }
  }
     System.out.println(x+"  TIENE "+voc+" vocales");
  }

  //Método que recibe una cadena y un número entero positivo
  //si el número es menor que la longitud de la cadena, extrae
  //esa cantidad de
  // caracteres de la cadena, en caso contrario invierte la cadena
  static void encuentra(String x, int n){
   int longitud;
   String cad2;
   longitud=x.length();
   if (n<longitud){
    cad2=x.substring(0, n);
    System.out.println(x+"  SUBCADENA= "+cad2);
    }
   else
      {
      StringBuilder invierte=new StringBuilder(x);
      cad2=invierte.reverse().toString();
      System.out.println(x+"  INVERTIDA= "+cad2);
      }
  }
}
//Clase que contiene el programa principal desde donde se llaman
//los métodos
package sobre_carga;
public class Sobre_carga {
public static void main(String[] args) {
  diferente opera=new diferente();
  opera.encuentra("imposible de creerlo",12);
  opera.encuentra(4.56);
  opera.encuentra("no se olvida si se practíca");
  opera.encuentra(2);
  opera.encuentra("ojalá fuera más sencillo",57);
    }
}
```

Respuestas a capítulo 7

1.

```
/*Se importa la clase ActionEvent y ActionListener
 *de awt para poder manejar los eventos  */
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
import org.netbeans.lib.awtextra.AbsoluteConstraints;
import org.netbeans.lib.awtextra.AbsoluteLayout;

public class Capitales_grafico {
public Capitales_grafico (){
// Se crea el contenedor de los elementos gráficos
JFrame ventana = new JFrame ("LISTA DE PAÍSES");

/*Se declaran e inicializan cuadros de texto con sus leyendas en-
tre paréntesis */
JTextField cod = new JTextField ("Ingresa Código: ");
JTextField pais = new JTextField ("Códigos de Países");
JTextField mex = new JTextField ("1 México");
JTextField ita = new JTextField ("2 Italia");
JTextField suri = new JTextField ("3 Surinám");
JTextField sue = new JTextField ("4 Suecia");

// Se especifican los cuadros de texto no serán editables
cod.setEditable(false);
pais.setEditable(false);
mex.setEditable(false);
ita.setEditable(false);
suri.setEditable(false);
sue.setEditable(false);

//Se crea un cuadro de texto con un valor predefinido de cero_uno
final JTextField campoTextoPredefinido = new JTextField ("01");

// Se crea un botón con la leyenda Mostrar info
JButton boton = new JButton("Mostrar Info");

// Se define el contenedor de tipo Absoluto
ventana.setLayout(new AbsoluteLayout());
ventana.setSize(300,200);    //se especifica el tamaño del panel
ventana.setVisible(true);  //se indica que será visible el panel

//se determina que el panel no podrá ser redimensionado
ventana.setResizable (false);
```

```
//Indica terminar la ejecución del programa al cerrar ventana(panel)
ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

//se agregan al panel los elementos, con sus respectivas coorde-
nadas
ventana.add(pais, new AbsoluteConstraints(20,10));
ventana.add(mex, new AbsoluteConstraints(20,35));
ventana.add(ita, new AbsoluteConstraints(20,60));
ventana.add(suri, new AbsoluteConstraints(20,85));
ventana.add(sue, new AbsoluteConstraints(20,110));
ventana.add(cod, new AbsoluteConstraints(20,135));
ventana.add(campoTextoPredefinido,      new      AbsoluteConstra-
ints(120,135));
ventana.add(boton,new AbsoluteConstraints(170,130));

// Evento de presionar el botón
boton.addActionListener(
         new ActionListener(){
            public void actionPerformed(ActionEvent ae) {

 // Se obtiene el texto contenido en el JTextField
  int valorcod=Integer.parseInt(campoTextoPredefinido.getText());
            String msg;
            switch (valorcod)
               {
               case 1:
               msg="País: México \nCapital: Distrito Federal \n" +
                    "Continente: Americano \nIdioma: Español \n" +
                          "Moneda: Peso\n";
                      break;
               case 2:
                  msg="País: Italia \nCapital: Roma \n" +
                     "Continente: Europeo \nIdioma: Italiano  \n"+
                      "Moneda: Euro\n";
                      break;
             case 3:
                 msg="País: Surinam \nCapital: Paramaribo \n"+
                 "Continente: Africano \nIdioma: Holandés  \n"+
                          "Moneda: Guinea\n";
                      break;
              case 4:
                msg="País: Suecia \nCapital: Estocolmo \n"+
               "Continente: Europeo \nIdioma: Sueco  \nMoneda:"+
                          " Corona Sueca\n";
                      break;
              default:
```

3.

```
import java.util.Random;
import javax.swing.*;

public class Aguila_Sol_Scroll {
Aguila_Sol_Scroll(){

//Se crea la ventana donde aparecerán las salidas
JFrame ventana= new JFrame();

// Se crea el panel  que permitirá hacer scroll en la ventana
JScrollPane scroll = new JScrollPane();
ventana.add(scroll);     // Se agrega el scroll a la ventana
ventana.setTitle("EJECUCIÓN");  // Se da título a la ventana

//se indica terminar ejecución al cerrar ventana
ventana.setDefaultCloseOperation(ventana.EXIT_ON_CLOSE);
ventana.setSize(250,200);     // Se define tamaño de la ventana
// Se indican coordenadas x,y donde aparecerá
ventana.setLocation(400, 150);
String result=lanzar();  // Se ejecuta el método lanzar

// Se crea un control de área de texto para poner en el la salida
// resultante
JTextArea texto = new JTextArea(result);

//se indica hacer la ventana visible
ventana.setVisible(true);

//se determina que la ventana no podrá ser redimensionada
ventana.setResizable (false);

// Se activa el scroll de acuerdo al control texto
scroll.setViewportView(texto);
}

//////////////////////  método lanzar  ///////////////
public String lanzar(){
    String lanzamientos = JOptionPane.showInputDialog(null,
           "Cuántos lanzamientos : ", "Ingresar lanzamientos", 1);
   //En variable salida se concatenarán resultados
    String salida=" Resultados:\n\n";
    Random aleatorio=new Random();
    int n, soles, x, aguilas, c;
    n=Integer.parseInt(lanzamientos);
```

```
    soles=0; aguilas=0; c=0;
    do {
       //genera número entero aleatorio menor a 2
       x=aleatorio.nextInt(2);
       if (x==1){
            salida+="Sol\n";
            soles+=1;
            }
       else {
          salida+="Águila\n";
          aguilas+=1;
            }
       c++;
    } while (c<n);

    salida+="\n\n"+soles+" Soles y "+aguilas+" Águilas";
    return salida;
}

// Principal de programa
public static void main(String[] args){
// Creación del objeto
Aguila_Sol_Scroll b=new Aguila_Sol_Scroll();
 }; // termina main
} // termina clase
```

5.

```
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

public class Tablas_GUI {
    //Se crea la ventana
    JFrame ventana= new JFrame();

   // Se crea el panel  que permitirá hacer scroll en la ventana
    JScrollPane scroll = new JScrollPane();

  //  constructor
   public Tablas_GUI(){
   ventana.add(scroll); // Se agrega el scroll a la ventana
    ventana.setTitle("EJECUCIÓN");  // Se da título a la ventana

   //se indica terminar ejecución al cerrar ventana
```

```
ventana.setDefaultCloseOperation(ventana.EXIT_ON_CLOSE);
    ventana.setSize(200,500);    // Se define tamaño de la ventana
    ventana.setLocation(400, 150); // Se indican coordenadas

    //se indica hacer la ventana visible
    ventana.setVisible(true);

    // Se crea un elemento de área de texto
    JTextArea texto = new JTextArea("TABLAS DE MULTIPLICAR \n\n");
    int a, b, c;
    for (b=1; b<=10; b++){
       for (a=1; a<=10; a++){
          c=a*b;
          // Se agrega texto al elemento
                  texto.append(b+"X"+a+"="+c+"\n");
         }
        texto.append("\n");  // Se agrega salto de línea al elemento
      }
  texto.append("FIN \n");  // Se agrega texto al elemento

  // Se activa el scroll de acuerdo al control texto
  scroll.setViewportView(texto);
  }

   //Programa principal
   public static void main(String[] args) {
   Tablas_GUI ob=new Tablas_GUI();

    }
}
```

7.

```
import javax.swing.JOptionPane;
public class Pilas_Colas_GUI {
  public static void main(String[] args) {
   PILA op=new PILA (5);
   COLA oc=new COLA (8);
   int opc=1;
   while(opc!=3){
opc=Integer.parseInt(JOptionPane.showInputDialog(null,"ELIGE "
    + "UNA OPCION: \n1.-PILA\n2.-COLA\n3.-SALIR DE EJECUCION"));
     switch(opc){
        case 1: op.comenzar();
                break;
```

```
        case 2:oc.comenzar();
               break;
         case 3: JOptionPane.showMessageDialog(null, "FIN DE EJE-
CUCION... "
                + "Bye Bye","PRINCIPAL",1,null);break;
         default: JOptionPane.showMessageDialog(null, "Esa opción
no "
                + "existe ","PRINCIPAL",1,null);
   } // del switch
   } // del while

 }//del metodo main

}
/******************** CLASE PILA ************************/
 class PILA{
   int [] arr;  //  Atributos de la clase pila para alumnos enteros
   int indice;
   public PILA(int dim)  // Método  constructor
      {
      if(dim>0)
          {JOptionPane.showMessageDialog(null, "Hay una pila para
"+dim+
                 " valores","TRABAJANDO CON LA PILA",1,null);
          arr=new int [dim];
          indice=0;    //pila vacía
         }
       }
   public boolean verificar_vacia()
      {
      if (indice==0)  // la pila esta vacía
         return true;
      else
         return false;
      }
   public boolean verificar_llena()

      {
      if (indice==arr.length)
         return true;
      else
         return false;
      }
   public void apilar(int x)
   {
   if (verificar_llena())
     JOptionPane.showMessageDialog(null, "No se puede apilar el "+x+
```

```
                ", PILA LLENA","TRABAJANDO CON LA PILA",1,null);
   else
      {arr[indice]=x;
       indice++;
      }
   }
   public int desapilar()
   {
   if (verificar_vacia())
   {JOptionPane.showMessageDialog(null, "IMPOSIBLE DESAPILAR, PILA
VACIA",
            "TRABAJANDO CON LA PILA",1,null);
    return -1; //Restringe a que no se puedan almacenar el -1 en
el arreglo
   }
   else
      {indice--;
       return arr[indice];
      }
   }

   public void comenzar()  //
   {
   int opc=1,elem;
   while(opc!=4)
{opc=Integer.parseInt(JOptionPane.showInputDialog("ELIGE UNA
OPCION:"
            + " \n1.-Apilar nuevo valor\n2.-Desapilar valor\n"
            + "3.-Imprimir valores existentes\n4.-Dejar la Pila"));
      switch(opc){
        case 1: if (!verificar_llena())
        {elem= Integer.parseInt(JOptionPane.showInputDialog("Qué "
                                        + "valor quieres apilar?"));
                 apilar(elem); // Recordar no dar -1
                 }
              else
                  JOptionPane.showMessageDialog(null, "IMPOSIBLE
APILAR, "
                + "PILA LLENA","TRABAJANDO CON LA PILA",1,null);
               break;
        case 2: elem=desapilar();
                 if (elem!=-1)
                    JOptionPane.showMessageDialog(null, "Se desa-
pilo el "
                              + "valor "+elem,"TRABAJANDO CON LA
PILA",1,null);
```

```
                break;
        case 3: impila();break;
        case 4: JOptionPane.showMessageDialog(null, "... DEJAS LA "
                + "PILA...","TRABAJANDO CON LA PILA",1,null);break;
          default: JOptionPane.showMessageDialog(null, "Esa opción
no "
                 + "existe ","TRABAJANDO CON LA PILA",1,null);
   } // del switch
   } // del while
}// fin del método comenzar

   public void impila()
   {
   String cad="";
   if (!verificar_vacia())
        {for(int i=0;i<indice;i++)
           cad=cad.concat(String.valueOf(arr[i])+", ");
         JOptionPane.showMessageDialog(null, "Los valores en la "
              + "pila son: "+cad,"TRABAJANDO CON LA PILA",1,null);
        }
   else
     JOptionPane.showMessageDialog(null, "PILA VACIA","TRABAJANDO "
              + "CON LA PILA",1,null);

   }
}

 /******************* CLASE COLA *******************/
 class COLA{
    String [] arr;  //  Atributos de la clase COLA, para alumnos
String
   int indice;
   public COLA(int dim)  // **************  método  constructor
      {
      if(dim>0)
          {JOptionPane.showMessageDialog(null, "Hay una COLA para
"+dim+
                " alumnos","TRABAJANDO CON LA COLA",1,null);
          arr=new String [dim];
          indice=0;//cola  vacía
         }
       }

  public boolean verificar_vacia()
     {
```

```
      if (indice==0)  // la cola está vacía
         return true;
      else
         return false;
      }
   public boolean verificar_llena()
      {
      if (indice==arr.length)
         return true;
      else
         return false;
      }
   public void formar(String alumno)
   {if (verificar_llena())
       JOptionPane.showMessageDialog(null,"No se puede formar el
alumno "+
          alumno+" ,COLA LLENA" ,"TRABAJANDO CON LA COLA",1,null);
   else
      {arr[indice]=alumno;
       indice++;
      }
   }
   public String despachar()
   {String despachado;
   if (verificar_vacia())
   {JOptionPane.showMessageDialog(null, "NO HAY NADIE FORMADO, "
          + "COLA VACIA","TRABAJANDO CON LA COLA",1,null);
    return null;
   }
   else
      {despachado=arr[0]; // alumno que será despachado
       recorrer(); //se reacomodan los alumnos desde la posición 0
       indice--;
       return despachado;
      }
   }

   public void recorrer()
   {
    for(int i=0;i<indice-1;i++)
        arr[i]=arr[i+1];
   }
   public void comenzar()
   {
   int opc=1;
   String alumno;
```

```
   while(opc!=4)
     {opc=Integer.parseInt(JOptionPane.showInputDialog("ELIGE UNA
OPCION:"
             + " \n1.-Formar Alumno\n2.-Despachar Alumno\n"
             + "3.-Ver a Todos los formados\n4.-Dejar la Cola"));
      switch(opc){
        case 1: if (!verificar_llena())
                  {alumno= JOptionPane.showInputDialog("A quien "
                          + "quieres formar?");
                  formar(alumno);
                  }
                else
                   JOptionPane.showMessageDialog(null, "IMPOSIBLE
FORMAR, "
                                  + "COLA LLENA","TRABAJANDO CON LA
COLA",1,null);
                break;
        case 2: alumno=despachar();
                 if (alumno!=null)
                     JOptionPane.showMessageDialog(null, "Se des-
pacho a "+
                          alumno,"TRABAJANDO CON LA COLA",1,null);
                break;
        case 3: impcola();break;
        case 4: JOptionPane.showMessageDialog(null, "... DEJAS LA
COLA...",
                "TRABAJANDO CON LA COLA",1,null);break;
         default: JOptionPane.showMessageDialog(null, "Esa opción
no "
                + "existe ","TRABAJANDO CON LA COLA",1,null);
   } // del switch
   } // del while
}// fin del método comenzar

   public void impcola()   // ****************************

  {
   String cad="";
   if (!verificar_vacia())
        {for(int i=0;i<indice;i++)
            //Se concatenan en cad los alumnos del arreglo
           cad=cad.concat((arr[i])+"\n");
            JOptionPane.showMessageDialog(null, "Los alumnos for-
mados son: "
                  + "\n"+cad,"TRABAJANDO CON LA COLA",1,null);
        }
```

```
    else
       JOptionPane.showMessageDialog(null, "COLA VACIA","TRABAJANDO
"
                 + "CON LA COLA",1,null);

    }
}
```

APÉNDICES

Apéndice A. Instalación de NetBeans IDE

1) Antes de instalar el NetBeans IDE, se debe tener instalado el JDK (*Java Development Kit*, Equipo de Desarrollo de Java). El JDK es un software que incluye herramientas útiles para desarrollar y probar programas escritos en el lenguaje de programación Java.

   El JDK incluye el compilador de Java (javac.exe) y el intérprete de Java (java.exe), entre otros componentes; los cuales permiten crear aplicaciones en Java. De esta manera, un programador puede escribir en un blog de notas su programa para compilarlo y ejecutarlo con los elementos del JDK; sin embargo, esto presenta más complejidad porque al no tener una interfaz gráfica (como NetBeans IDE), el programador tiene que hacer todo desde línea de comandos. Para descargar el JDK se recomienda ir a la página:

   http://www.oracle.com/technetwork/java/javase/downloads/index.html

Allí aparecerá un botón como el que se muestra en la **Figura A.1** al cuál se le dará clic. Es probable que aparezca una leyenda debajo como "*Java Platform (JDK) 7u7*" o "*Java Platform (JDK) 7u9*", esto quiere decir que la versión que se descargará es "la 7 con la actualización 7" para el primer caso o "la versión 7 con la actualización 9" en el segundo caso; el JDK sufre actualizaciones constantemente y se recomienda bajar la versión más reciente.

Figura A.1. Botón de descarga del JDK.

Después de hacer clic en el botón, aparecerá una página con un recuadro como el de la **Figura A.2**, donde se deberá aceptar el acuerdo de licencia, por defecto estará seleccionado "*Decline LicenseAgreement*" por lo que será necesario seleccionar "*AcceptLicenseAgreement*".

Figura A.2. Aceptación de la licencia del JDK.

Al aceptar los términos de la licencia, este recuadro cambiará y se mostrará en su lugar uno como el que aparece en la **Figura A.3**.

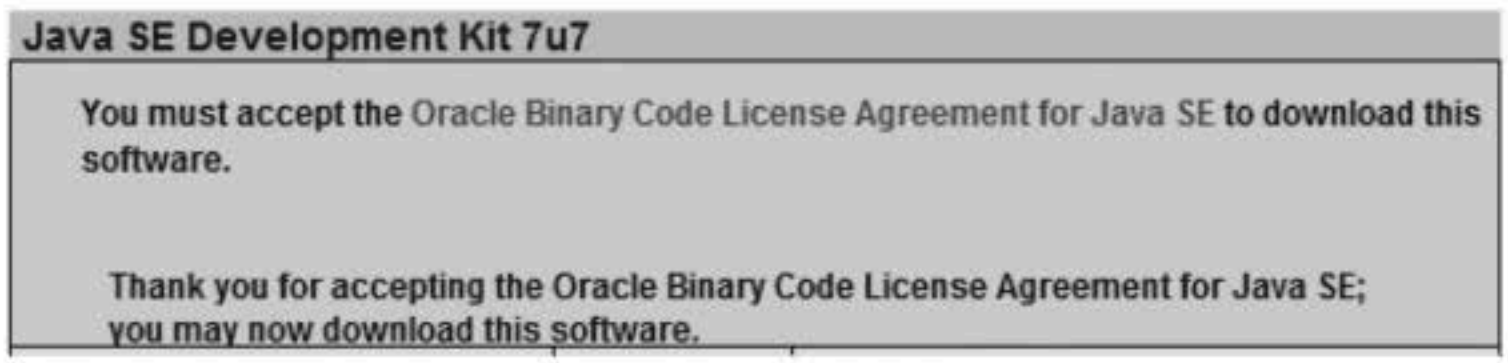


Figura A.3. Aceptación de la licencia del JDK (segundo paso).

En la misma página aparecerá una lista como la que se muestra en la **Figura A.4**; por lo que, el siguiente paso consiste en seleccionar la versión del JDK dependiendo del sistema operativo en el que se trabajará, para lo cual se le dará clic en la liga en azul de la columna "*Download*" del JDK que se deseé utilizar y se iniciará la descarga del archivo.

Product / File Description	File Size	Download
Linux x86	120.63 MB	jdk-7u9-linux-i586.rpm
Linux x86	92.85 MB	jdk-7u9-linux-i586.tar.gz
Linux x64	118.82 MB	jdk-7u9-linux-x64.rpm
Linux x64	91.59 MB	jdk-7u9-linux-x64.tar.gz
Mac OS X	143.47 MB	jdk-7u9-macosx-x64.dmg
Solaris x86	135.14 MB	jdk-7u9-solaris-i586.tar.Z
Solaris x86	91.51 MB	jdk-7u9-solaris-i586.tar.gz
Solaris SPARC	135.7 MB	jdk-7u9-solaris-sparc.tar.Z
Solaris SPARC	95.15 MB	jdk-7u9-solaris-sparc.tar.gz
Solaris SPARC 64-bit	22.8 MB	jdk-7u9-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	17.51 MB	jdk-7u9-solaris-sparcv9.tar.gz
Solaris x64	22.48 MB	jdk-7u9-solaris-x64.tar.Z
Solaris x64	14.94 MB	jdk-7u9-solaris-x64.tar.gz
Windows x86	88.35 MB	jdk-7u9-windows-i586.exe
Windows x64	90.03 MB	jdk-7u9-windows-x64.exe

Figura A.4. Lista de versiones del JDK.

En seguida, será necesario abrir la carpeta en donde se guardó el archivo descargado (véase **Figura A.5**) y darle doble clic.

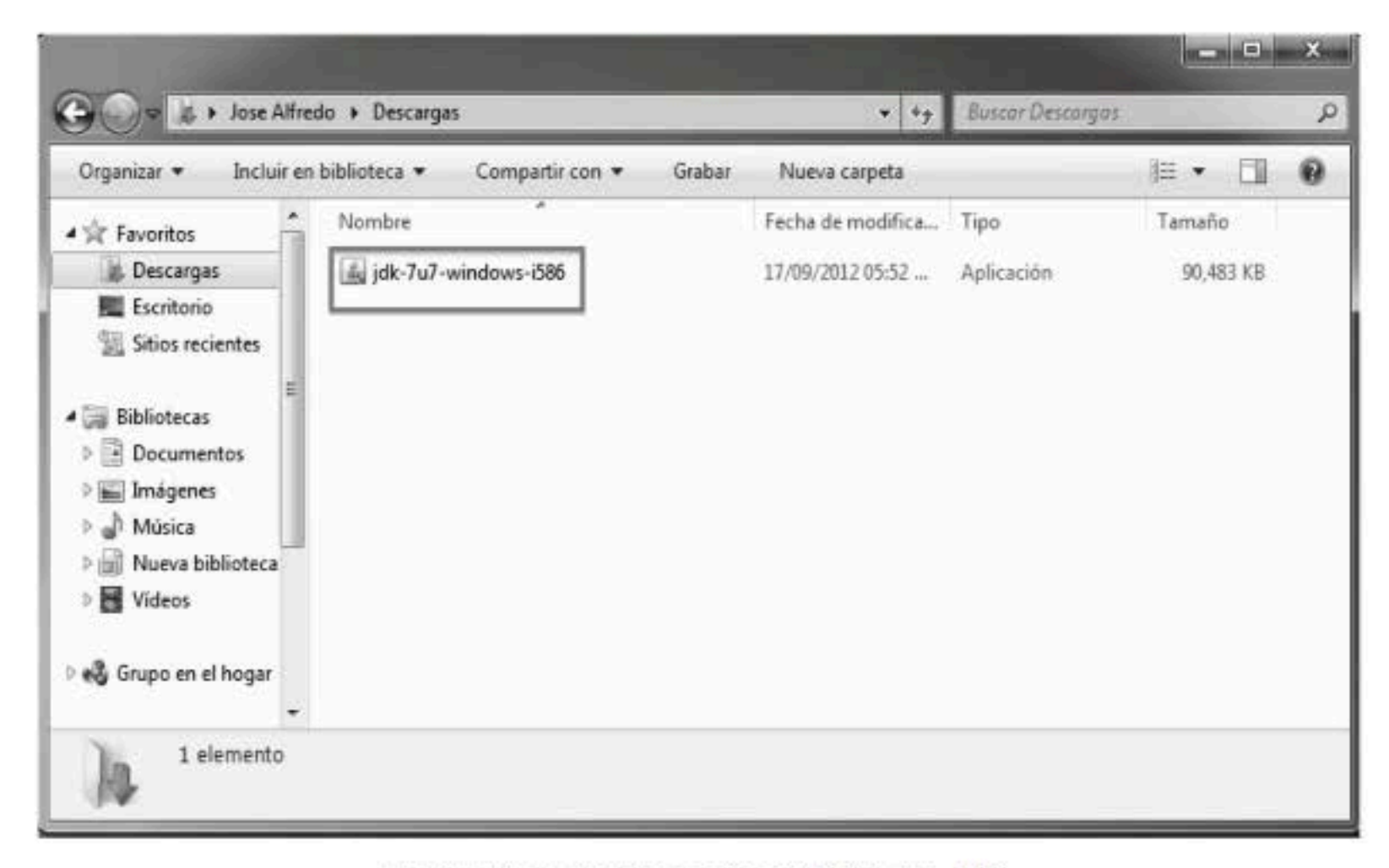


Figura A.5. Archivo de instalación del JDK.

Después de dar doble clic, aparecerá una ventana como la que se muestra en la **Figura A.6**.

Figura A.6. Primera pantalla de instalación del JDK.

Será necesario leer las instrucciones de las ventanas que aparecerán para instalar el JDK, la última de las ventanas será la que se muestra en la **Figura A.7**; al oprimir el botón "close" se cerrará la ventana y quedará instalado exitosamente el JDK.

Figura A.7. Última pantalla de instalación del JDK.

2) Una vez instalado el JDK, se instalaNetBeans IDE. Para descargarlo se recomienda ir al sitio web:

http://netbeans.org/downloads/

Aparecerá una página Web como la que muestra en la **Figura A.8**, en la cual será necesario elegir el idioma del NetBeans IDE y el sistema operativo, éstas opciones están localizadas en la parte superior derecha, tal como se indica con rectángulos rojos.

Existen diferentes versiones del NetBeans IDE:

- Java SE (*Java Standard Edition*, Edición Estándar de Java), la cual tiene los elementos básicos para desarrollar aplicaciones en Java.
- Java EE (*Java Enterprise Edition*, Edición Empresarial de Java), la cual tiene más elementos que la versión Java SE y permite crear aplicaciones más complejas.
- C/C++, permite programar en lenguajes C y C++.
- PHP, permite programar en el lenguaje PHP.
- All, proporciona todos elementos de las versiones Java SE, Java EE, C/C++ y PHP.

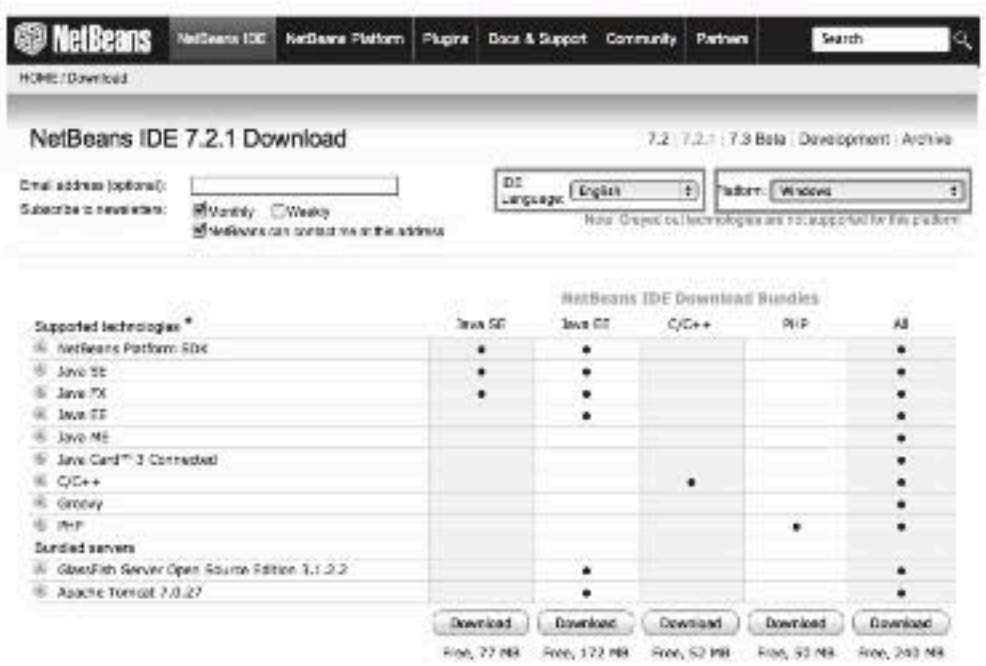


Figura A.8. Descarga de NetBeans IDE.

Para fines didácticos se recomienda descargar la versión Java SE, para lo cual se da clic en el botón "*Download*" de la columna de esta versión. Se abrirá una nueva página como la que se muestra en la **Figura A.9** y se iniciará la descarga del instalador del NetbeansIDE.

Figura A.9. Descarga de NetBeansIDE (segunda página web).

Después; es necesario dar doble clic sobre el archivo que se descargó, tal como se puede observar en la **Figura A.10**.

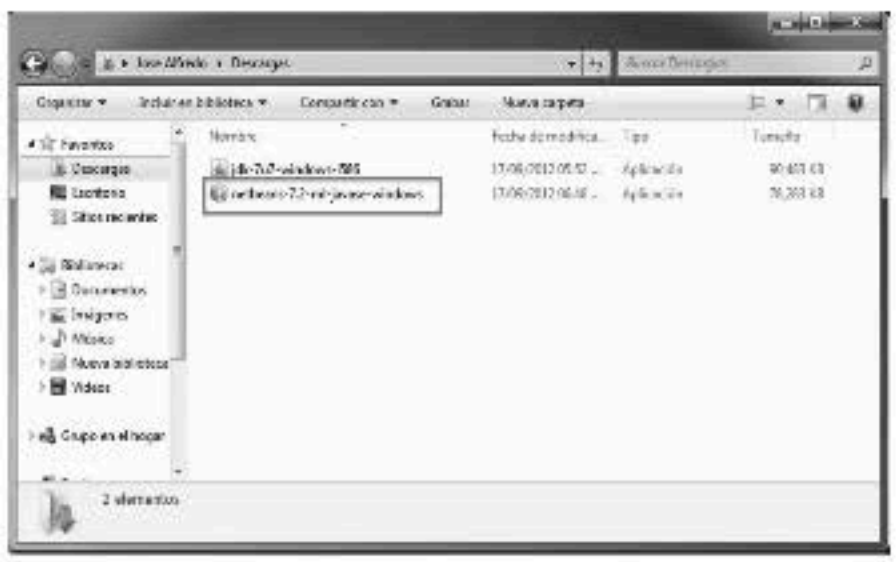


Figura A.10. Archivo de instalación de NetBeans IDE.

Al dar doble clic se abrirá una serie de ventanas para realizar la instalación, la primera de ellas se muestra en la **Figura A.11**.

Figura A.11. Primera pantalla de instalación de NetBeans IDE.

La última ventana del instalador es la que se muestra en la **Figura A.12**, sólo será necesario dar clic en el botón "*Finish*", se cerrará la ventana y quedará instalado Netbeans IDE.

Figura A.12. Última pantalla de instalación de NetBeans IDE.

Finalmente, en el escritorio aparecerá el ícono de la aplicación, el cual corresponde al de la **Figura A.13**. Cada vez que se deseé utilizar NetBeans sólo se dará clic sobre ese ícono.

Figura A.13. Ícono de NetBeans IDE.

Apéndice B. Sintaxis para importar una clase de otro paquete

Para importar una clase que pertenece a un paquete diferente se requiere hacer uso de la siguiente sintaxis en Java:

```
package paquete_que_importara.paquete_a_importar;
import paquete_a_importar.clase_a_importar;
```

Por ejemplo, si se tienen en un mismo proyecto en NetBeans IDE, dos paquetes llamados `artículosDisponibles y tienda` que a su vez tienen una clase cada uno, llamadas Zapatos y `Venta_articulos` respectivamente, y se desea que la clase Venta_articulos pueda utilizar los atributos y métodos públicos (**`public`**) de la clase Zapatos sería necesario codificar lo siguiente:

Clase Venta_articulos

```
//Líneas para importar la clase Zapatos que pertence a un paquete
//diferente llamado artículosDisponibles
package tienda.artículosDisponibles;
import artículosDisponibles.Zapatos;

public class Venta_articulos {
    public static void main (String args[]){

        /*se puede crear un objeto de la clase Zapatos
          porque al importarla ya es visible
          con el objeto "venta" se pueden acceder a los
          atributos y/o métodos public de la clase Zapatos*/
        Zapatos venta = new Zapatos ();

  //se imprime el valor del atributo "precio"
        System.out.println(venta.precio);

        //se llama al método "imprimirTalla"
        venta.imprimirTalla();

        //se modifica el valor del atributo "precio"
        venta.precio=676.9;

        //se imprime el valor del atributo "precio"
        System.out.println(venta.precio);
    }
}
```

Clase Zapatos

```
package artículosDisponibles;
public class Zapatos {

   //atributos de la clase Zapatos
    int modelo = 7656;
    public double precio = 349.50;
    protected String tipo = "tenis";
    protected int color = 2;
    private float talla = 24.5f;

    //métodos de la clase Zapatos
    public void imprimirTalla(){
        System.out.println("La talla es="+talla);
    }
}
```

```
run:
349.5
La talla es=24.5
676.9
BUILD SUCCESSFUL (total time: 1 second)
```

Apéndice C. Instrucciones para agregar la biblioteca Absolute Layout

Para poder utilizar el administrador de elementos gráficos Absolute Layout, es necesario agregar la biblioteca de Java a la que pertenece.

El primer paso consiste en ubicar la carpeta "*Libraries*" (Bibliotecas) del proyecto de NetBeans IDE en el cuál se está trabajando, véase **Figura C.1**.

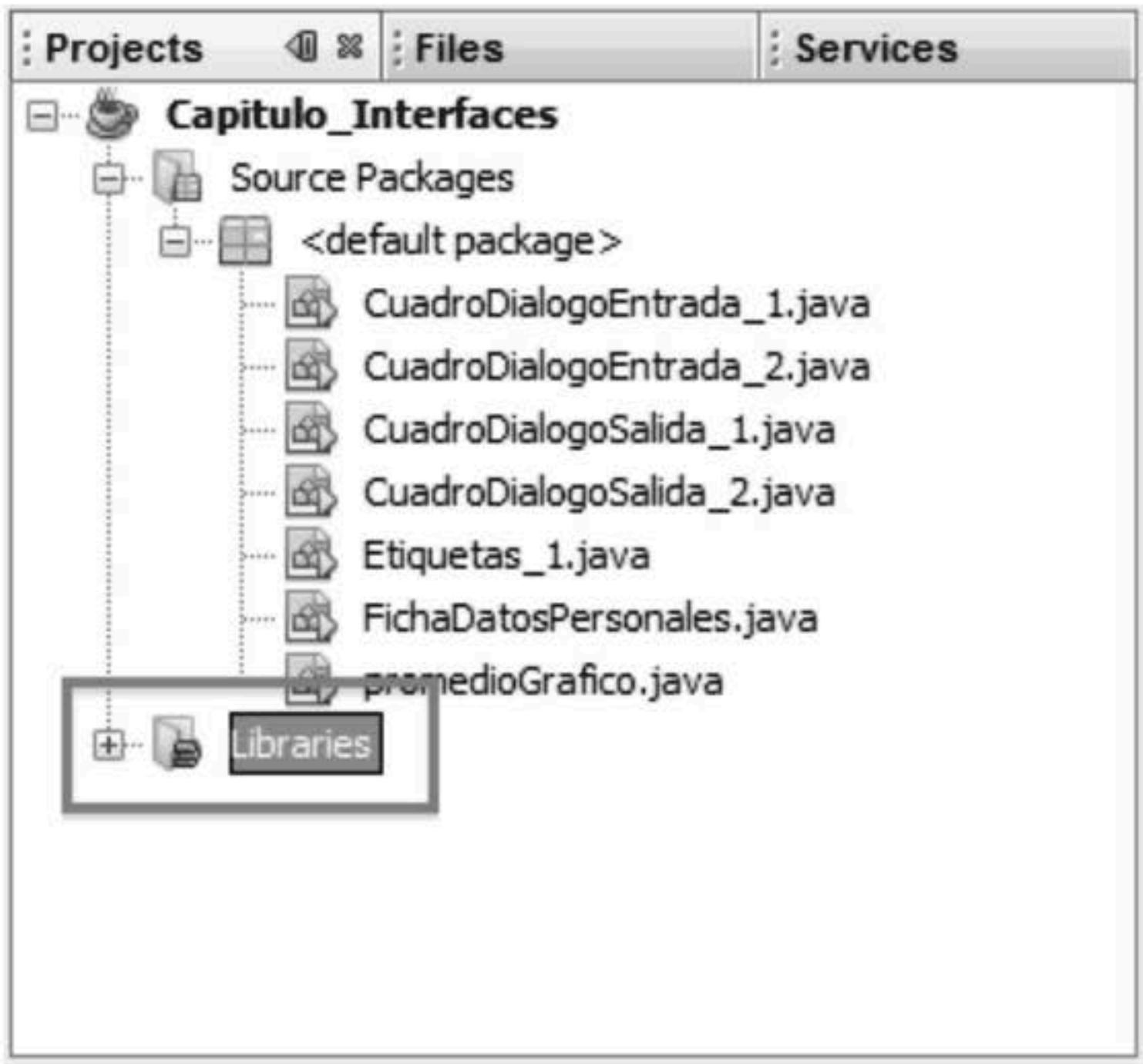


Figura C.1. Selección de la carpeta "Libraries".

El segundo paso consiste en dar clic sobre la carpeta con el botón derecho del mouse para que aparezca el menú desplegable de la **Figura C.2**.

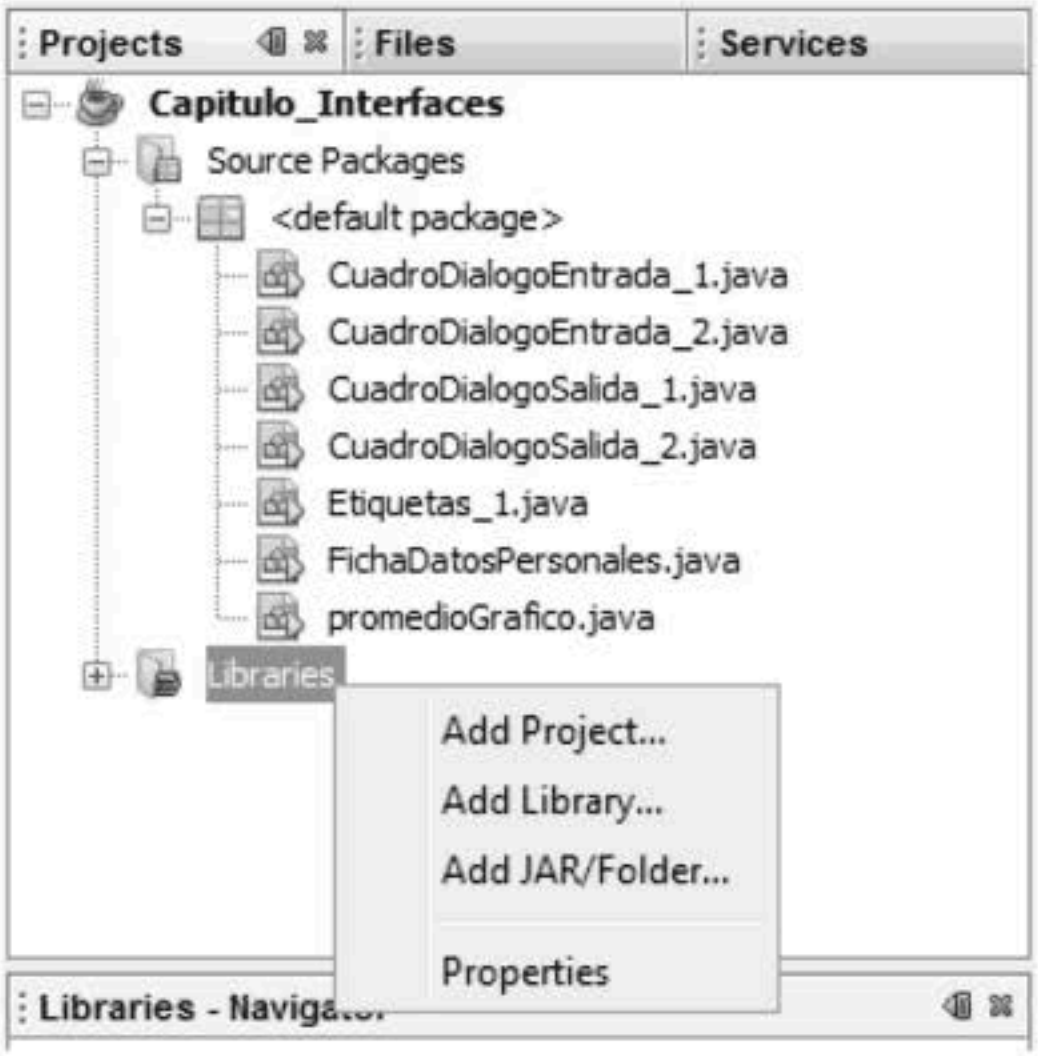


Figura C.2. Menú desplegable de la carpeta "Libraries".

El tercer paso es seleccionar "*Add Library*" (Agregar Biblioteca). Al dar clic en esa opción se abrirá una ventana como la que se muestra en la **Figura C.3**, de la cual se deberá seleccionar la biblioteca "*Absolute Layout*" y dar clic en el botón "*Add Library*" (Agregar Biblioteca).

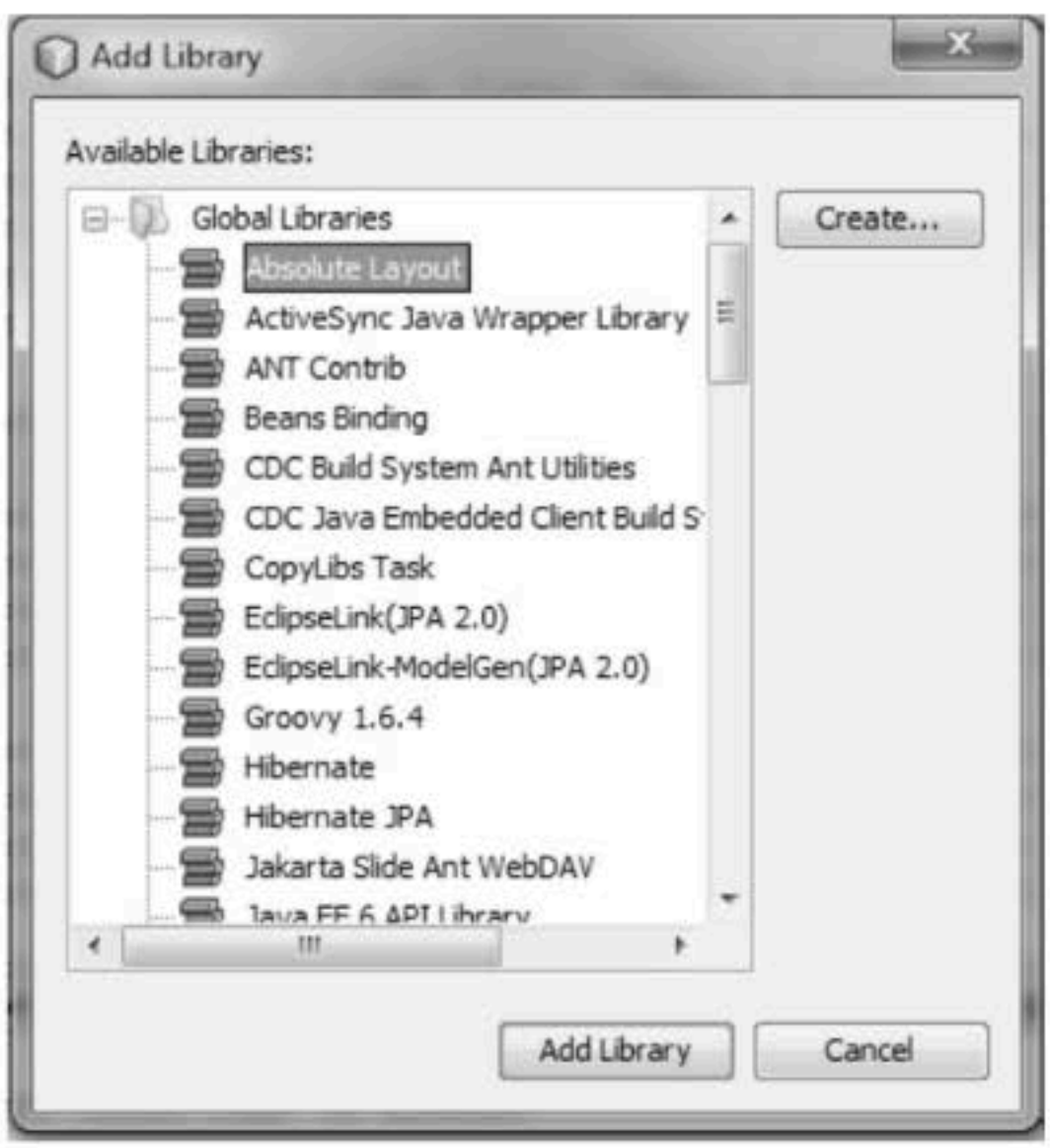
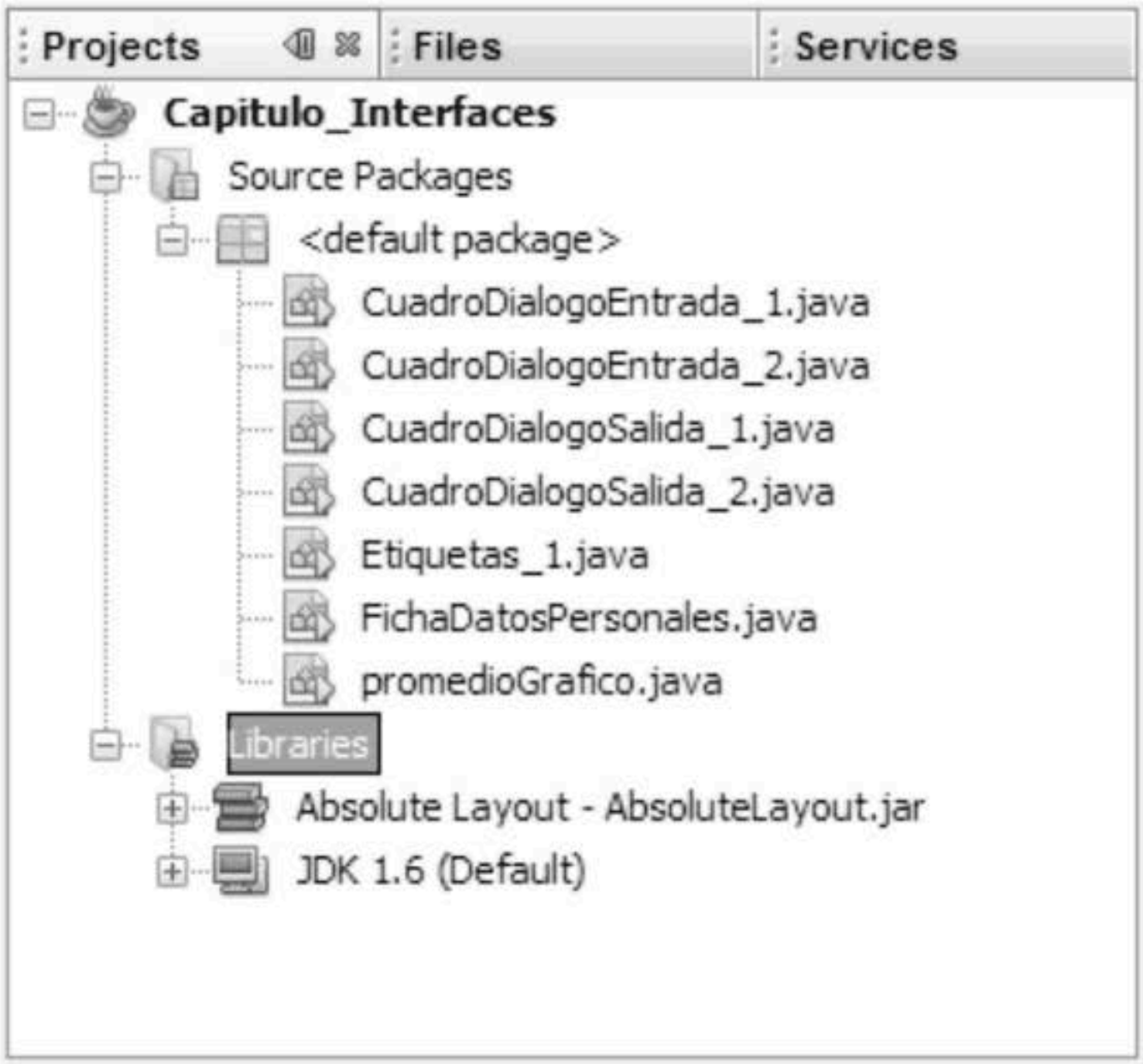


Figura C.3. Selección de la biblioteca Absolute Layout.

Como resultado, aparecerá en la carpeta de "*Libraries*" del proyecto de NetBeans IDE, una carpeta con el nombre de la biblioteca "*Absolute Layout – AbsoluteLayout.jar*", véase **Figura C.4**.