CENGAGE Learning

PROGRAMACIÓN JAVA

DEL ANÁLISIS DE PROBLEMAS AL DISEÑO DE PROGRAMAS

5a. Ed.

D.S. Malik

PROGRAMACIÓN JAVA

DEL ANÁLISIS DE PROBLEMAS AL DISEÑO DE PROGRAMAS

5a. Ed.

D.S. Malik

Traducción

Ing. Javier León Cárdenas

Profesor del Departamento de Ciencias Básicas
Escuela Superior de Ingeniería Química e Industrias Extractivas
Instituto Politécnico Nacional

Revisión técnica

M. en C. Jorge Cortés Galicia

Profesor del Departamento de Ingeniería y Sistemas Computacionales
Escuela Superior de Cómputo
Instituto Politécnico Nacional

CENGAGE Learning®

Australia • Brasil • Japón • Corea • México • Singapur • España • Reino Unido • Estados Unidos

Programación Java
del Análisis de Problemas al Diseño de Programas
5a. Ed.
D. S. Malik

Presidente de Cengage Learning Latinoamérica
Fernando Valenzuela Migoya

Director editorial, de producción y de plataformas digitales para Latinoamérica
Ricardo H. Rodríguez

Gerente de procesos para Latinoamérica
Claudia Islas Licona

Gerente de manufactura para Latinoamérica
Raúl D. Zendejas Espejel

Gerente editorial de contenidos en español
Pilar Hernández Santamarina

Gerente de proyectos especiales
Luciana Rabuffetti

Coordinador de manufactura
Rafael Pérez González

Editores
Sergio R. Cervantes González
Gloria Luz Olguín Sarmiento

Diseño de portada
Gerardo Larios

Imagen de portada
© Shutterstock/Semisatch

Composición tipográfica
Black Blue impresión y diseño S.A. de C.V.
Baktun 13 Comunicación, Gerardo Larios

Traducido del libro: Java Programming, From Problem Analysis to Program Design. Fifth Edition
D. S. Malik
Publicado en inglés por Course Technology, una compañía de Cengage Learning © 2012
ISBN 13: 978-1-111-53053-2

Datos para catalogación bibliográfica:
Malik, D. S.
Programación Java
del Análisis de Problemas al Diseño de Programas
5a. Ed.
ISBN 13: 978-607-481-926-7

Visite nuestro sitio en:
http://latinoamerica.cengage.com

Impreso en México
1 2 3 4 5 6 7 15 14 13 12

Contenido abreviado

Tabla de contenido

CAPÍTULO 1

REVISIÓN GENERAL DE COMPUTADORAS Y LENGUAJES DE PROGRAMACIÓN

EN ESTE CAPÍTULO:

- Aprenderá acerca de los diferentes tipos de computadoras
- Explorará los componentes de hardware y software de un sistema de cómputo
- Aprenderá acerca del lenguaje de una computadora
- Aprenderá acerca de la evolución de los lenguajes de programación
- Examinará lenguajes de programación de alto nivel
- Descubrirá qué es un compilador y qué hace
- Examinará cómo se procesa un programa en Java
- Aprenderá acerca de la Internet y de la World Wide Web
- Aprenderá qué es un algoritmo y explorará técnicas de solución de problemas
- Se familiarizará con las metodologías de diseño de la programación estructurada y de la orientada a objetos

Introducción

Los términos como la "Internet", que no eran familiares hace apenas algunos años, ahora son comunes. Los estudiantes de primaria con regularidad "navegan" en la Internet y utilizan computadoras para diseñar sus proyectos de clase. Mucha gente utiliza la Internet para consultar información y comunicarse con otros. Estas actividades en la Internet son posibles por la disponibilidad de diferente tipo de software, también conocido como programas de computadora. El software se desarrolla utilizando lenguajes de programación. El lenguaje de programación Java está específicamente bien adecuado para desarrollar software que realice tareas específicas. Nuestro objetivo principal es enseñarle cómo escribir programas en el lenguaje de programación Java. Antes de que comience a programar, es útil si comprende algo de la terminología básica y algunos de los diferentes componentes de una computadora. Comenzamos con una revisión general de la historia de las computadoras.

Revisión general de la historia de las computadoras

El primer dispositivo conocido para efectuar cálculos fue el ábaco. El ábaco se inventó en Asia pero se utilizó en la antigua Babilonia, China y a través de Europa hasta finales de la Edad Media. En el ábaco se utiliza un sistema de cuentas deslizantes sobre un marco para realizar sumas y restas. En 1642, el filósofo y matemático francés Blaise Pascal inventó un dispositivo de cálculo denominado Pascalina. Tenía ocho discos móviles sobre ruedas que podían calcular sumas con una longitud de hasta ocho cifras. Tanto el ábaco como la Pascalina podían efectuar sólo operaciones de suma y resta. Más tarde en el siglo XVII, Gottfried von Leibniz inventó un dispositivo que podía sumar, restar, multiplicar y dividir. En 1819, Joseph Jacquard, un tejedor francés, descubrió que las instrucciones de tejido para sus telares se podían almacenar en tarjetas con agujeros perforados en ellas. Al tiempo que las tarjetas se movían en secuencia a través del telar, las agujas se pasaban a través de los agujeros y recogían hilos del color y textura correctos. Un tejedor podía reacomodar las tarjetas y cambiar el patrón que se estaba tejiendo. En esencia, las tarjetas programaban un telar para producir patrones en una tela. La industria de hilados y tejidos parece tener poco en común con la industria de las computadoras. Sin embargo, la idea de almacenar información perforando agujeros en una tarjeta resultó ser de gran importancia en el desarrollo posterior de las computadoras.

A inicios y mediados de 1800, Charles Babbage, un matemático y físico inglés, diseñó dos máquinas de cálculo: la diferencial y la analítica. La máquina diferencial podía efectuar de manera automática operaciones complejas, como elevar un número al cuadrado. Babbage construyó un prototipo de la máquina diferencial pero no el dispositivo real. La primera máquina diferencial completa se finalizó en Londres en 2002, 153 años después de que se diseñó. Consiste de 8 000 partes, pesa cinco toneladas y mide 11 pies de longitud. Una réplica de la máquina diferencial se terminó en 2008 y está en exhibición en el Computer History Museum en Mountain View, California (*http://www.computerhistory.org/babbage/*). La mayoría del trabajo de Babbage se conoce a través de los escritos de su colega Ada Augusta, Condesa de Lovelace. Augusta es considerada la primera programadora de computadoras.

Al final del siglo XIX, los oficiales del Censo de los Estados Unidos de América necesitaban ayuda para tabular con precisión los datos del censo. Herman Hollerith inventó una máquina de cálculo que operaba con electricidad y utilizaba tarjetas perforadas para almacenar datos. Su máquina fue un gran éxito. Hollerith fundó la Tabulating Machine Company, que más tarde se convirtió en la corporación de computadoras y tecnología conocida como IBM.

La primera máquina similar a una computadora fue la Mark I, que se construyó en 1944 de manera conjunta por la IBM y la Universidad de Harvard bajo la dirección de Howard Aiken. Se utilizaron tarjetas perforadas para suministrar datos a la máquina. La Mark I medía 52 pies, pesaba 50 toneladas y tenía 750 000 partes. En 1946, la ENIAC (Electronic Numerical Integrator and Calculator) se construyó en la Universidad de Pensilvania. Contenía 18 000 tubos de vacío y pesaba unas 30 toneladas.

Las computadoras que conocemos en la actualidad utilizan reglas de diseño propuestas por John von Neumann a finales de la década de 1940. Su diseño incluía componentes como una unidad aritmética lógica, una unidad de control, memoria y dispositivos de entrada/salida. Estos componentes se describen en la siguiente sección. El diseño de la computadora de Von Neumann hace posible almacenar las instrucciones de diseño y los datos en el mismo espacio de la memoria. En 1951, la UNIVAC (Universal Automatic Computer) se construyó y se vendió a la Oficina del Censo de los Estados Unidos de América.

En 1956, la invención de los transistores resultó en computadoras de menor tamaño, más rápidas, más confiables y con mayor rendimiento de energía. Esta época también vio el surgimiento de la industria del desarrollo del software con la introducción de FORTRAN y COBOL, dos de los primeros lenguajes de programación. En el siguiente avance tecnológico importante, los transistores se reemplazaron por circuitos integrados diminutos o "chips." Los chips son más pequeños y más económicos que los transistores y pueden contener cientos de circuitos en un solo chip. Estos circuitos proporcionan a las computadoras una velocidad de procesamiento asombrosa.

En 1970 se inventó el microprocesador, una unidad central de procesamiento (CPU) en un solo chip. En 1977, Stephen Wozniak y Steven Jobs diseñaron y construyeron la primera computadora Apple en su cochera. En 1981 la IBM introdujo su computadora personal (PC). En la década de 1980 los clones de la PC IBM hicieron aún más asequible la computadora personal. A mediados de la década de 1990 la gente de diversos estratos sociales podía comprarlas. Las computadoras continúan haciéndose más rápidas y menos costosas conforme avanza la tecnología.

Las computadoras actuales son muy poderosas, confiables y muy fáciles de usar. Pueden aceptar instrucciones habladas e imitar el razonamiento humano mediante la inteligencia artificial. Las aplicaciones de cómputo móviles crecen de manera significativa. Utilizando dispositivos portátiles, los conductores de entregas pueden acceder a satélites de posicionamiento global (GPS) a fin de verificar las ubicaciones de los clientes para recolecciones y entregas. Los teléfonos celulares pueden verificar su correo electrónico, hacer reservaciones en aerolíneas, ver cómo se comporta la bolsa de valores y acceder a sus cuentas bancarias.

Si bien existen varias categorías de computadoras, como unidades centrales, de tamaño medio y micro, todas comparten algunos elementos básicos.

Elementos de un sistema de cómputo

Una computadora es un dispositivo electrónico capaz de ejecutar comandos. Los comandos básicos que ejecuta una computadora son de entrada (obtener datos), de salida (presentar resultados), de almacenamiento y de realización de operaciones aritméticas y lógicas. Hay dos componentes principales de un sistema de cómputo: hardware y software. En las siguientes secciones se presenta un panorama resumido de estos componentes. Primero se analiza el hardware.

Hardware

Los principales componentes de hardware incluyen la unidad central de procesamiento (CPU); la memoria principal (MP), también denominada memoria de acceso aleatorio (RAM); los dispositivos de entrada/salida y el almacenamiento secundario. Algunos ejemplos de los dispositivos de entrada son teclado, ratón y almacenamiento secundario. Ejemplos de los dispositivos de salida son monitor o pantalla, impresora y almacenamiento secundario.

UNIDAD CENTRAL DE PROCESAMIENTO Y MEMORIA PRINCIPAL

La **unidad central de procesamiento (CPU)** es el "cerebro" de la computadora y la pieza individual más costosa del hardware en una computadora. Cuanto más poderosa sea la CPU, más rápida es la computadora. Las operaciones aritméticas y lógicas se efectúan dentro de la CPU. En la figura 1-1*a*) se muestran algunos componentes de hardware.

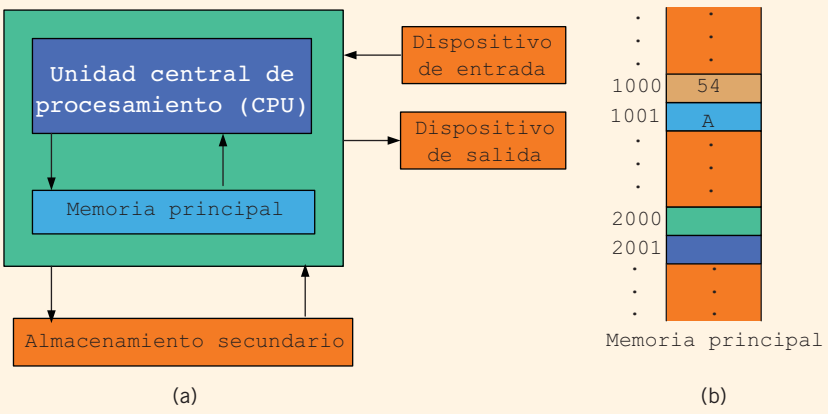

FIGURA 1-1 Componentes de hardware de una computadora y memoria principal

La **memoria principal** o memoria de acceso aleatorio (RAM) está conectada directamente a la CPU. Todos los programas se deben cargar en la memoria principal antes de que se puedan ejecutar. De manera similar, todos los datos se deben llevar a la memoria principal antes de que

un programa los pueda manipular. Cuando la computadora se apaga, todo lo contenido en la memoria principal se pierde.

La memoria principal es una secuencia ordenada de celdas, denominadas **celdas de memoria**. Cada una tiene ubicación única en la memoria principal, llamada **dirección** de la celda. Estas direcciones ayudan a acceder a la información almacenada en la celda. En la figura 1-1*b*) se muestra la memoria principal con algunos datos.

Las computadoras actuales disponen de una memoria principal compuesta de millones a miles de millones de celdas. Aunque en la figura 1-1*b*) se muestran datos almacenados en las celdas, el contenido de cada una puede ser una instrucción de programación o bien datos. Además, en esta figura se muestran los datos como números y letras. Sin embargo, como se explica más adelante en este capítulo, la memoria principal almacena todo el contenido como secuencias de ceros y unos. Las direcciones de memoria también se expresan como secuencias de unos y ceros.

ALMACENAMIENTO SECUNDARIO

Dado que los programas y datos se deben almacenar en la memoria principal antes de su procesamiento y debido a que todo el contenido en dicha memoria se pierde cuando se apaga la computadora, la información almacenada en la memoria principal se debe transferir a otro dispositivo para su almacenamiento a largo plazo. Un dispositivo que almacena información a largo plazo (a menos que el dispositivo se vuelva inutilizable o que se cambie la información rescribiéndola con otra) se denomina **almacenamiento secundario.** Para transferir información de la memoria principal al almacenamiento secundario, estos componentes se deben conectar directamente entre sí. Ejemplos de almacenamiento secundario son los discos duros, discos flexibles (actualmente de uso poco común), memoria flash (USB), discos ZIP, CD-ROM y las cintas.

DISPOSITIVOS DE ENTRADA/SALIDA

Para que una computadora realice una tarea útil, debe poder tomar datos y programas presentando los resultados de la manipulación de los datos. Los dispositivos que suministran datos y programas a las computadoras se denominan **dispositivos de entrada.** El teclado, ratón y almacenamiento secundario son ejemplos de dispositivos de entrada. Los artefactos que la computadora utiliza para presentar y almacenar resultados se denominan **dispositivos**

FIGURA 1-2 Algunos dispositivos de entrada y de salida

de salida. Un monitor, impresora y almacenamiento secundario son ejemplos de dispositivos de salida. En la figura 1-2 se muestran algunos dispositivos de entrada y salida.

Software

El software consiste de programas escritos para realizar tareas específicas. Por ejemplo, los programas de procesamiento de textos se utilizan para escribir cartas, artículos y libros. Los dos tipos de programas son de sistema y de aplicación.

Los **programas de sistema** controlan la computadora. El programa de sistema que se carga primero cuando se enciende la PC se denomina **sistema operativo**. Sin un sistema operativo, la computadora es inútil. El sistema operativo monitorea la actividad global de la computadora y proporciona servicios como administración de la memoria, de actividades de entrada/salida y administración del almacenamiento. El sistema operativo tiene un programa especial que organiza el almacenamiento secundario de manera que se pueda acceder a la información en forma conveniente. El sistema operativo es el programa que ejecuta los programas de aplicación. Los **programas de aplicación** realizan tareas específicas. Los procesadores de texto, las hojas de cálculo y los juegos son ejemplos de programas de aplicación. Los sistemas operativos y los programas de aplicación están escritos en lenguajes de programación.

Lenguaje de una computadora

Cuando se presiona la tecla A en el teclado, la computadora presenta A en la pantalla, pero ¿qué está almacenado en realidad dentro de la memoria principal de la computadora?, ¿cuál es el lenguaje de la computadora?, ¿cómo almacena lo que se escribe en el teclado?

Recuerde que una computadora es un dispositivo electrónico. Las señales eléctricas se mueven a lo largo de canales dentro de la computadora. Hay dos tipos de señales eléctricas: analógicas y digitales. Las **señales analógicas** son formas de onda continua utilizadas para representar cosas, como sonidos. Las cintas de audio, por ejemplo, almacenan datos en señales analógicas. Las **señales digitales** representan información con una secuencia de ceros y unos. Un 0 representa un voltaje bajo y un 1 representa un voltaje alto. Las señales digitales son portadoras de información más confiable que las señales analógicas y se pueden copiar de un dispositivo a otro con precisión exacta. Quizás haya observado que cuando se hace una copia de una cinta de audio, la calidad sonora de la copia no es tan buena como la de la original. Las computadoras utilizan señales digitales.

Debido a que las señales digitales se procesan dentro de una computadora, el lenguaje de una computadora, denominado **lenguaje de máquina**, es una secuencia de ceros y unos. El dígito 0 o 1 se denomina **dígito binario** o **bit**. En ocasiones a una secuencia de ceros y unos se le refiere como **código binario** o **número binario.**

Bit: digito binario `0` o `1`.

Una secuencia de ocho bits se denomina **byte.** Además, 2^{10} = 1 024 bytes y se denomina **kilobyte (KB)**. En la tabla 1-1 se resumen los términos empleados para describir los diversos números de bytes.

TABLA 1-1 Unidades binarias

Unidad	Símbolo	Bits/bytes
Byte		8 bits
Kilobyte	KB	2^{10} bytes = 1 024 bytes
Megabyte	MB	1024 KB = 2^{10} KB = 2^{20} bytes = 1 048 576 bytes
Gigabyte	GB	1024 MB = 2^{10} MB = 2^{30} bytes = 1 073 741 824 bytes
Terabyte	TB	1024 GB = 2^{10} GB = 2^{40} bytes = 1 099 511 627 776 bytes
Petabyte	PB	1024 TB = 2^{10} TB = 2^{50} bytes = 1 125 899 906 842 624 bytes
Exabyte	EB	1024 PB = 2^{10} PB = 2^{60} bytes = 1 152 921 504 606 846 976 bytes
Zettabyte	ZB	1024 EB = 2^{10} EB = 2^{70} bytes = 1 180 591 620 717 411 303 424 bytes

Cada letra, número o símbolo especial (como * o {) en el teclado está codificado como una secuencia de bits, cada uno con una representación única. El esquema de codificación de uso más común en computadoras personales es el **American Standard Code for Information Interchange (ASCII)** de siete bits. El conjunto de datos ASCII consiste de 128 caracteres numerados del 0 al 127. (Observe que 2^7 = 128 y 2^8 = 256.) Es decir, en el conjunto de datos ASCII, la posición del primer carácter es 0, la posición del segundo carácter es 1 y así sucesivamente. En este esquema, `A` está codificada como `1000001`. De hecho, `A` es el 66vo. carácter en el código de caracteres ASCII, pero su posición es 65 debido a que la posición del primer carácter es 0. Además, `1000001` es la representación binaria de `65`. El carácter `3` está codificado como `0110011`. Para una lista completa del conjunto de caracteres ASCII, consulte el apéndice C.

NOTA

El sistema de numeración que utilizamos en nuestra vida cotidiana se denomina **sistema decimal** o **sistema base 10.** Debido a que todo lo que esté dentro de una computadora está representado como una secuencia de ceros y unos, es decir, números binarios, el sistema de numeración que una computadora utiliza se denomina binario o **base 2.** En el párrafo anterior se indicó que el número 1000001 es la representación binaria de 65. En el apéndice D se describe cómo convertir un número de base 10 a base 2 y viceversa; también cómo convertir un número entre base 2 y base 16 (hexadecimal) y entre base 2 y base 8 (octal).

Dentro de una computadora cada carácter está representado como una secuencia de ocho bits, es decir, como un byte. Debido a que el código ASCII es de siete dígitos, se debe agregar un `0` a la izquierda de la codificación ASCII de un carácter. De aquí, dentro de una computadora, el carácter `A` se representa como `01000001` y el carácter `3` se representa como `00110011`.

Otros esquemas de codificación incluyen el código Unicode, que es un desarrollo más reciente. **Unicode** consiste de 65 536 caracteres. Para almacenar un carácter de Unicode, se necesitan 2 bytes. En Java se utiliza el conjunto de caracteres Unicode. Por tanto, en Java cada carácter se representa como una secuencia de 16 bits, es decir, 2 bytes. En Unicode el carácter A se representa como `0000000001000001`.

El conjunto de caracteres ASCII es un subconjunto de Unicode; los primeros 128 caracteres de Unicode son los mismos que los caracteres en ASCII. Si se está tratando sólo con el idioma inglés, el conjunto de caracteres ASCII es suficiente para escribir programas en Java. La ventaja del conjunto de caracteres Unicode es que los símbolos de otros idiomas distintos del inglés se pueden manejar con facilidad.

Evolución de los lenguajes de programación

El lenguaje de computadora más básico, el lenguaje de máquina, proporciona instrucciones de un programa en bits. Aunque la mayoría de las computadoras realizan el mismo tipo de operaciones, los diseñadores de diferentes CPU en ocasiones eligen conjuntos distintos de códigos binarios para efectuar estas operaciones. Por tanto, el de una computadora no necesariamente es el mismo que el de otra. La única consistencia entre computadoras es que en cualquiera de ellas, todos los datos se almacenan y manipulan como un código binario.

Las primeras computadoras se programaban en lenguaje de máquina. Para ver cómo se escriben las instrucciones en lenguaje de máquina, suponga que se quiere utilizar la ecuación:

```
wages = rate • hours
```

para calcular el *wages* (salario) semanal. Suponga que las localizaciones de memoria de `rate` (pago por hora), `hours` (horas) y `wages` (salario) son `010001`, `010010` y `01011`, respectivamente. Suponga además que el código binario `100100` representa load (carga), `100110` representa multiplication (multiplicación) y `100010` representa store (almacenar). En lenguaje de máquina se podría necesitar la secuencia de instrucciones siguiente para calcular el salario semanal:

```
100100 010001
100110 010010
100010 010011
```

Para representar la ecuación del salario semanal en lenguaje de máquina, el programador tenía que recordar los códigos del lenguaje de máquina para varias operaciones. Además, para manipular datos, el programador tenía que recordar las localizaciones de los datos en la memoria principal. Tener que recordar códigos específicos hacía difícil la programación y era propensa a errores.

Los lenguajes ensambladores se desarrollaron para facilitar el trabajo del programador. En el **lenguaje ensamblador,** una instrucción es una forma fácil de recordar denominada **nemotécnica.** En la tabla 1-2 se muestran algunos ejemplos de instrucciones en lenguaje ensamblador y su código en lenguaje de máquina correspondiente.

TABLA 1-2 Ejemplos de instrucciones en lenguaje ensamblador y lenguaje de máquina

Lenguaje ensamblador	Lenguaje de máquina
LOAD	100100
STOR	100010
MULT	100110
ADD	100101
SUB	100011

Utilizando instrucciones en lenguaje ensamblador se puede escribir la ecuación para calcular el salario semanal como sigue:

```
LOAD  rate
MULT  hours
STOR  wages
```

Como se puede ver, es mucho más fácil escribir instrucciones en lenguaje ensamblador. Sin embargo, una computadora no puede ejecutar instrucciones en lenguaje ensamblador de manera directa. La instrucción primero se tiene que traducir a lenguaje de máquina. Un programa llamado **ensamblador** traduce las instrucciones en lenguaje ensamblador a lenguaje de máquina.

Ensamblador: programa que traduce un programa escrito en lenguaje ensamblador en un programa equivalente en lenguaje de máquina.

Cambiar de lenguaje de máquina a lenguaje ensamblador facilitó la programación, pero un programador aún estaba obligado a pensar en términos de instrucciones de máquina individuales. El paso siguiente hacia facilitar la programación fue idear **lenguajes de alto nivel** que fueran más similares a los idiomas hablados, como inglés y español. Basic, FORTRAN, COBOL, Pascal, C, C++ y Java son lenguajes de alto nivel. En este libro aprenderá el lenguaje de alto nivel Java.

En Java la ecuación del salario semanal se escribe así:

```
wages = rate * hours;
```

la instrucción escrita en Java se comprende mucho más fácil y se explica a sí misma para un usuario principiante familiarizado con la aritmética básica. Sin embargo, al igual que en el caso del lenguaje ensamblador, una computadora no puede ejecutar de manera directa instrucciones escritas en un lenguaje de alto nivel. Para que corran en una computadora, estas instrucciones en Java primero se necesitan traducir a un lenguaje intermedio denominado **bytecode** y después interpretarlas en un lenguaje de máquina particular. Un programa denominado **compilador** traduce las instrucciones escritas en Java a bytecode.

Compilador: programa que traduce un programa escrito en un lenguaje de alto nivel a un lenguaje de máquina equivalente. (En el caso de Java, este lenguaje de máquina es el bytecode.)

Recuerde que una computadora comprende sólo el lenguaje de máquina. Además, tipos diferentes de CPU utilizan distintos lenguajes de máquina. Para hacer los programas en Java **independientes de la máquina**, es decir, que puedan ejecutarse en tipos diferentes de plataformas de computadoras, los diseñadores de Java introdujeron una computadora hipotética denominada la **Máquina Virtual Java (JVM).** De hecho, el bytecode es el lenguaje de máquina para la JVM.

En lenguajes como C y C++, el compilador traduce directamente el código fuente al lenguaje de máquina de la CPU de una computadora. Para esos lenguajes se necesita un compilador diferente para cada tipo de CPU. Por tanto, los programas en estos lenguajes no se trasladan con facilidad de un tipo de máquina a otro. El código fuente se debe compilar para cada tipo de CPU. Para hacer los programas en Java independientes de la máquina y facilitar su traslación así como para permitir que corran en un navegador Web, los diseñadores de Java introdujeron la Máquina Virtual Java (JVM) y el bytecode como el lenguaje (de máquina) de esta máquina. Es más fácil traducir un bytecode a un tipo particular de CPU. Este concepto se analiza aún más en la siguiente sección, "Procesamiento de un programa en Java".

Procesamiento de un programa en Java

Java tiene dos tipos de programas: aplicaciones y applets. El siguiente es un ejemplo de un programa de aplicación en Java:

```
public class MyFirstJavaProgram
{
    public static void main(String[] args)
    {
       System.out.println("Mi primer programa en Java.");
    }
}
```

En este punto no es necesario preocuparse por los detalles de este programa. Sin embargo, si se ejecuta (corre) este programa, presentará la línea siguiente en la pantalla:

```
Mi primer programa en Java.
```

Recuerde que una computadora sólo puede comprender el lenguaje de máquina. Por tanto, a fin de ejecutar este programa de manera exitosa, el código primero se tiene que traducir a lenguaje de máquina. En esta sección se analizan los pasos requeridos para ejecutar programas escritos en Java.

Para procesar un programa escrito en Java, se efectúan los pasos siguientes, como se ilustra en la figura 1-3.

1. Se utiliza un editor de texto, como Notepad, para crear (es decir, teclear) un programa en Java siguiendo las reglas o sintaxis del lenguaje. Este programa se denomina **programa fuente**. El programa se debe guardar en un archivo de texto nombrado `ClassName.java`, donde `ClassName` es el nombre de la clase en Java contenida en el archivo. Por ejemplo, en el programa en Java ilustrado

antes, el nombre de la clase (`public`) que contiene el programa en Java es `MyFirstJavaProgram`. Por tanto, este programa se debe guardar en el archivo de texto nombrado `MyFirstJavaProgram.java`. De otra forma ocurrirá un error.

Programa fuente: programa escrito en un lenguaje de alto nivel.

2. Se debe verificar que el programa obedezca las reglas del lenguaje de programación, es decir, el programa debe estar sintácticamente correcto y traducir el programa al bytecode equivalente. El compilador verifica si el programa fuente tiene errores de sintaxis y, si no encuentra algún error, traduce el programa en bytecode. El bytecode se guarda en el archivo con la extensión `.class`. Por ejemplo, el bytecode para `MyFirstJavaProgram.java` lo guarda el compilador en el archivo `MyFirstJavaProgram.class`.
3. Para ejecutar un programa de aplicación en Java, el archivo `.class` se debe cargar en la memoria principal. Para ejecutar una applet de Java, se debe utilizar un navegador Web o un visor de applets y un navegador Web simplificado para ejecutar los applets. Los programas que se escriben en Java por lo general se desarrollan utilizando un **entorno de desarrollo integrado (IDE).** El IDE contiene muchos programas que son útiles al crear sus programas. Por ejemplo, el código necesario para presentar los resultados del programa y varias funciones matemáticas para facilitar un poco el trabajo del programador. Dado que cierto código ya está disponible, se puede utilizar en vez de escribir el propio. También se pueden desarrollar bibliotecas propias (denominadas *paquetes* en Java). (Observe que en Java, por lo común, un paquete es un conjunto de clases relacionadas. Por tanto, en general, un programa en Java es una colección de clases. En los capítulos 2 y 8 esto se explica detalladamente. En este punto no es necesario preocuparse por estos detalles). En general, para ejecutar con éxito un programa en Java, el bytecode para las clases utilizadas en el programa debe estar conectado. El programa que hace esto de manera automática en Java se conoce como **cargador.**
4. El paso siguiente es ejecutar el programa en Java. Además de conectar el bytecode de varias clases, el cargador también carga el bytecode del programa en Java en la memoria principal. Conforme las clases se cargan en la memoria principal, el *verificador de bytecode* verifica que el bytecode para las clases sea válido y que no viole las restricciones de seguridad de Java. Por último, un programa denominado **intérprete** traduce cada instrucción en bytecode a lenguaje de máquina de la computadora y luego lo ejecuta.

   Intérprete: programa que lee y traduce cada instrucción en bytecode a lenguaje de máquina de la computadora y luego lo ejecuta.

Observe que el intérprete de Java traduce y ejecuta una instrucción de bytecode a la vez. No traduce primero todo el bytecode a lenguaje de máquina de la computadora. Como se destacó antes, en lenguajes como C++ se necesita un compilador diferente para cada tipo de CPU, en tanto que un compilador de Java traduce un programa fuente en Java a bytecode, el lenguaje de máquina de JVM, el cual es independiente de cualquier tipo particular de CPU.

El intérprete de Java traduce cada instrucción en bytecode a un tipo particular de lenguaje de máquina de una CPU y luego ejecuta la instrucción. Así pues, en el caso del lenguaje Java, se necesita un tipo diferente de intérprete para un tipo particular de CPU. Sin embargo, los intérpretes son programas más simples que los compiladores. Debido a que el intérprete de Java traduce una instrucción en bytecode a la vez, los programas en Java se ejecutan más lentamente.

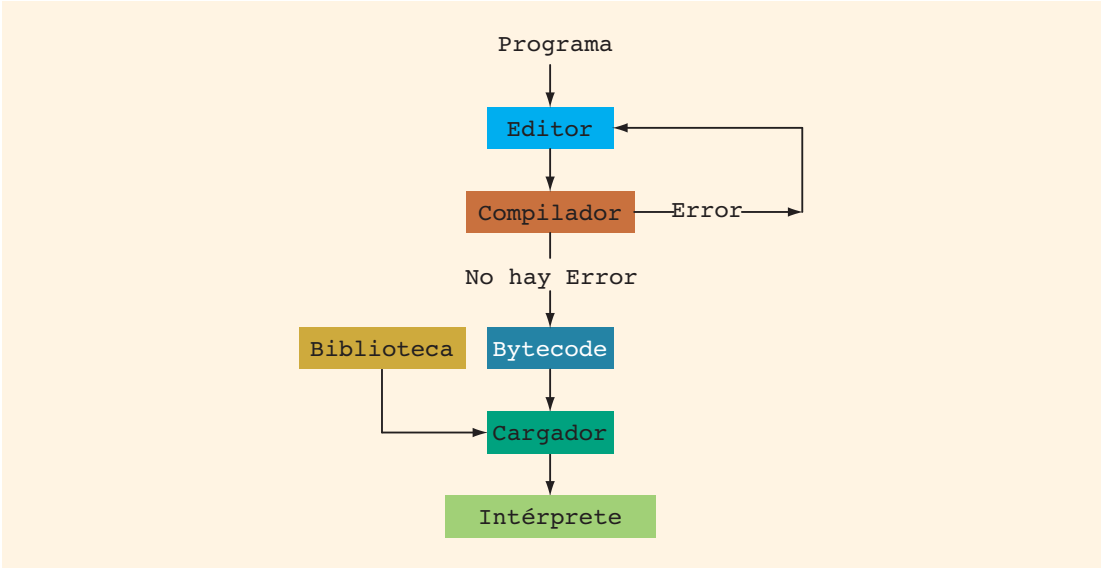

FIGURA 1-3 Procesamiento de un programa en Java

Como programador, una de sus preocupaciones principales se relaciona con el paso 1. Es decir, debe aprender, comprender y dominar las reglas del lenguaje de programación para crear programas fuente. Los programas se desarrollan utilizando un IDE. Los IDE más conocidos empleados para crear programas en Java incluyen JBuilder (de Borland), CodeWarrior (Metrowerks) y jGrasp (Universidad de Auburn). Estos IDE contienen un editor para crear el programa, un compilador para verificar si hay errores de sintaxis, un programa para cargar los códigos de los objetos de los recursos utilizados del IDE y un programa para ejecutar el programa. Estos IDE también son muy amigables con el usuario. Cuando se compila un programa, el compilador no sólo identifica los errores de sintaxis, sino que también sugiere cómo corregirlos.

NOTA

Otro software que se puede utilizar para desarrollar programas en Java incluye Eclipse, TextPad, JCreator, BlueJ y DrJava.

En el capítulo 2, después de la introducción de algunos elementos básicos de Java, se verá cómo se crea un programa en Java.

CAPÍTULO 2

ELEMENTOS BÁSICOS DE JAVA

EN ESTE CAPÍTULO:

- Se familiarizará con los componentes básicos de un programa en Java, incluyendo métodos, símbolos especiales e identificadores
- Explorará los tipos de datos primitivos
- Descubrirá cómo utilizar operadores aritméticos
- Examinará cómo un programa evalúa las expresiones aritméticas
- Explorará cómo se evalúan las expresiones mezcladas
- Aprenderá acerca del casting de tipos
- Se familiarizará con el tipo `String`
- Aprenderá qué es una instrucción de asignación y qué hace
- Descubrirá cómo ingresar datos en la memoria utilizando instrucciones de entrada
- Se familiarizará con el uso de operadores de incremento y decremento
- Examinará formas para dar salida a los resultados utilizando instrucciones de salida
- Aprenderá cómo importar paquetes y por qué estos son necesarios
- Descubrirá cómo crear un programa de aplicación en Java
- Aprenderá cómo comprender y corregir errores de sintaxis
- Explorará cómo estructurar apropiadamente un programa, incluyendo el uso de comentarios para documentar un programa
- Aprenderá cómo evitar errores utilizando un formateo consistente y apropiado, y recorriendo el código
- Aprenderá cómo hacer un recorrido del código

En este capítulo aprenderá los fundamentos de Java. A medida que comience a aprender el lenguaje de programación Java, surgen de manera natural dos interrogantes: primero, ¿qué es un programa de computadora? Segundo, ¿qué es programación? Un **programa de computadora** o un programa es una secuencia de instrucciones cuyo fin es realizar una tarea. **Programación** es un proceso de planeación y creación de un programa. Estas dos definiciones dicen la verdad, pero no toda acerca de la programación. Puede tomar un libro completo para dar una definición satisfactoria al respecto. Una analogía podría ayudar a obtener una mejor comprensión de la naturaleza de la programación, por lo que se utilizará un tema sobre el cual casi todos tienen cierto conocimiento: cocinar. Una receta también es un programa y todos con alguna experiencia en cocinar pueden concordar sobre lo siguiente:

1. Usualmente es más fácil seguir una receta que crear una.
2. Hay recetas buenas y recetas malas.
3. Algunas recetas se siguen con facilidad y algunas no.
4. Algunas recetas producen resultados confiables y algunas no.
5. Se debe tener algún conocimiento de cómo utilizar los utensilios de cocina para seguir una receta hasta el fin.
6. Para crear buenas recetas se debe tener un conocimiento y una comprensión significativos sobre cómo cocinar.

Estos mismos seis puntos también se pueden aplicar a la programación. Llevemos la analogía de cocinar un paso más adelante. Suponga que quiere enseñar a alguien cómo convertirse en un chef. ¿Cómo procedería? ¿Introduciría a la persona a la buena comida, esperando que desarrolle el gusto por ella? ¿Haría que la persona siga receta tras receta esperando que algunas de las técnicas se le transmitan, o primero le enseñaría el uso de los utensilios, la naturaleza de los ingredientes y alimentos, las especias y luego le explicaría cómo aprovechar todos estos elementos?

Al igual que hay muchas formas para enseñar a cocinar, también existen distintas maneras para enseñar cómo programar. Sin embargo, algunos fundamentos se aplican a la programación, al igual que se usan para cocinar u otras actividades, como la música.

Aprender un lenguaje de programación es como ejercitarse para convertirse en un chef o aprender a tocar un instrumento musical. Las tres habilidades requieren de una interacción directa con las herramientas, utensilios e instrumentos. No se puede convertir en un chef sólo leyendo recetas. De manera similar, no se puede aprender a tocar instrumentos musicales leyendo libros acerca de instrumentos musicales. Lo mismo es cierto para la programación. Se debe tener un conocimiento fundamental del lenguaje y probar los programas en la computadora para estar seguro de que cada programa realiza lo que se supone que hace.

Un programa en Java

En este y en el capítulo siguiente aprenderá los elementos y conceptos básicos del lenguaje de programación Java que se utilizan para crear un programa en Java. Además de dar ejemplos para ilustrar los múltiples conceptos, también se incluyen programas en Java para ayudar a clarificar dichos conceptos. En esta sección se incluye un ejemplo de un programa en Java. En

este punto no es necesario preocuparse demasiado por los detalles de este programa. Sólo se necesita comprender el efecto de una instrucción de *salida*, la cual se introduce en el programa.

Considere el siguiente programa (aplicación) en Java:

```
//*********************************************************
// Este es un programa simple en Java. Presenta tres lineas
// de texto, incluyendo la suma de dos numeros.
//*********************************************************

public class ASimpleJavaProgram
{
    public static void main(String[] args)
    {
       System.out.println("Mi primer programa en Java.");
       System.out.println("La suma de 2 y 3 = " + 5);
       System.out.println("7 + 8 = " + (7 + 8));
    }
}
```

Ejecución del ejemplo: (cuando se compila y ejecuta este problema, las tres líneas siguientes se presentan en la pantalla).

```
Mi primer programa en Java.
La suma de 2 y 3 = 5
7 + 8 = 15
```

Esta salida se presenta en la pantalla cuando se ejecutan las tres líneas siguientes:

```
System.out.println("Mi primer programa en Java.");
System.out.println("La suma de 2 y 3 = " + 5);
System.out.println("7 + 8 = " + (7 + 8));
```

Para explicar cómo sucede esto, considere la instrucción:

```
System.out.println("Mi primer programa en Java.");
```

Este es un ejemplo de una instrucción de *salida* en Java. Hace que el programa evalúe lo que se encuentra en el paréntesis y despliega el resultado en la pantalla. En general, cualquier cosa entre comillas dobles, denominada *cadena*, se evalúa por sí misma, es decir, su valor es la propia cadena. Por tanto, la instrucción causa que el sistema presente la línea siguiente en la pantalla:

```
Mi primer programa en Java.
```

(En general, cuando se imprime una cadena, se hace sin las comillas dobles.) Ahora considere la instrucción:

```
System.out.println("La suma de 2 y 3 = " + 5);
```

En esta instrucción de salida, los paréntesis contienen la cadena `"La suma de 2 y 3 = ", +` (el signo más) y el número `5`. Aquí el símbolo `+` se utiliza para concatenar (unir) los operandos. En este caso, el sistema automáticamente convierte el número `5` en una cadena, une esta con la primera cadena y presenta la línea siguiente en la pantalla:

```
La suma de 2 y 3 = 5
```

Ahora considere la instrucción:

```
System.out.println("7 + 8 = " + (7 + 8));
```

En esta instrucción de salida, los paréntesis contienen la cadena `"7 + 8 = "`, `+` (el signo más) y la expresión `(7 + 8)`. En la expresión `(7 + 8)`, observe el paréntesis alrededor de `7 + 8`. Esto causa que el sistema sume los números `7` y `8`, que resulta en `15`. Después el número `15` se convierte en la cadena `"15"` y luego se une con la cadena `"7 + 8"`. Por tanto, la salida de esta instrucción es:

```
7 + 8 = 15
```

En este y en el capítulo siguiente, hasta que se explique cómo construir un programa en Java de manera apropiada, se utilizarán instrucciones como las anteriores para explicar conceptos.

Antes de cerrar esta sección, analicemos algunos otros rasgos del anterior programa en Java. La unidad básica de un programa en Java es una **`class`**. En general, cada **`class`** en Java consiste en uno o más métodos. Hablando en términos generales, un método es una secuencia de enunciados o instrucciones cuyo objetivo es realizar algo. La primera línea del programa es:

```
public class ASimpleJavaProgram
```

`ASimpleJavaProgram` es el nombre de la **`class`** en Java. La segunda línea del programa consiste en la llave izquierda, la cual se correlaciona con la segunda llave derecha (la última). Estas llaves en conjunto marcan el inicio y el final de (el cuerpo de) la **`class`** `ASimpleJavaProgram`. La tercera línea consiste en:

```
public static void main(String[] args)
```

Este es el encabezado del método llamado `main`. Una **`class`** en Java puede tener a lo máximo un método `main`. Si una **`class`** en Java contiene un programa de aplicación, como el anterior, debe contener el método `main`. Cuando se ejecuta (corre) un programa (aplicación) en Java, la ejecución siempre inicia con el método `main`.

La octava línea consiste en una llave izquierda (la segunda llave izquierda del programa). Esto marca el inicio del (cuerpo del) método `main`. La primera llave derecha (en la 12a línea del programa) se correlaciona con esta llave izquierda y marca el fin del (cuerpo del) método `main`. Este último tiene una identación para separarlo.

En la siguiente sección aprenderá acerca de la finalidad de las líneas que se muestran en color verde en el programa.

Elementos de un programa en Java

Como se declaró en el capítulo anterior, los dos tipos de programas en Java son applets y programas de aplicación en Java. Los primeros son programas diseñados para correr en un navegador Web. Los segundos no lo requieren. Para introducir los componentes básicos de Java en algunos de los siguientes capítulos se desarrollan programas de aplicación en Java. Los applets en Java se consideran más adelante.

Si nunca ha visto un programa escrito en un lenguaje de programación, el programa en Java `ASimpleJavaProgram`, presentado en la sección anterior, puede parecer que está escrito en un idioma extraño. Para hacer oraciones con sentido en cualquier idioma extraño, se debe aprender su alfabeto, sus palabras y gramática. Lo mismo es válido en un lenguaje de programación. Para escribir programas significativos, se deben aprender los símbolos especiales, las palabras y reglas de sintaxis del lenguaje de programación. Las **reglas de sintaxis** indican cuáles de los enunciados (instrucciones) son legales o aceptados por el lenguaje de programación y cuáles no. También se deben aprender las **reglas de la semántica**, las cuales determinan el significado de las instrucciones. Las reglas, los símbolos, las palabras especiales y los significados del lenguaje de programación permiten escribir programas para resolver problemas.

Lenguaje de programación: conjunto de reglas, símbolos y palabras especiales utilizadas para construir programas.

En el resto de esta sección aprenderá acerca de algunos de los símbolos especiales utilizados en un programa de Java. Se introducen símbolos adicionales conforme se encuentran otros conceptos en capítulos posteriores. De manera similar, la sintaxis y las reglas de la semántica se introducen y se explican a lo largo del libro.

Comentarios

El programa que escriba debe ser claro no sólo para usted, sino también para el lector del mismo. Parte de una buena programación es la inclusión de comentarios en el programa. Es común que los comentarios se utilicen para identificar a los autores del programa, proporcionar la fecha cuando se escribió o modificó, dar una explicación breve del programa y explicar el significado de sus instrucciones clave. En los ejemplos de programación para los programas que escribiremos, no se incluirá la fecha de escritura, lo que es consistente con la convención estándar para escribir libros de este tipo.

Los comentarios son para el lector, no para el compilador. Por tanto, cuando un compilador compila un programa para verificar si hay errores de sintaxis, ignora por completo los comentarios. A lo largo de este libro los comentarios se muestran en color verde.

El programa `ASimpleJavaProgram` presentado en la sección anterior, contiene los siguientes comentarios:

```
//*********************************************************
// Este es un programa simple en Java. Presenta tres lineas
// de texto, incluyendo la suma de dos numeros.
//*********************************************************
```

Un programa en Java tiene dos tipos comunes de comentarios: en una sola línea y en líneas múltiples.

Los **comentarios en una sola línea** inician con `//` y se pueden colocar en cualquier parte de la línea. Todo lo que se encuentre en esa línea después de `//` lo ignora el compilador. Por ejemplo, considere la instrucción siguiente:

```
System.out.println("7 + 8 = " + (7 + 8));
```

Se pueden poner comentarios al final de esta línea como se muestra:

```
System.out.println("7 + 8 = " + (7 + 8)); //imprime: 7 + 8 = 15
```

Este comentario podría ser significativo para un programador principiante.

Los **comentarios en líneas múltiples** están contenidos entre `/*` y `*/`. El compilador ignora todo lo que aparezca entre `/*` y `*/`. El siguiente es un ejemplo de un comentario en líneas múltiples:

```
/*
    Usted puede incluir comentarios que pueden
    ocupar varias lineas.
*/
```

Símbolos especiales

Los siguientes son algunos de los símbolos especiales:

```
+   -   *   /

.   ;   ?   ,

<=  !=  ==  >=
```

La primera fila incluye símbolos matemáticos para la adición, sustracción, multiplicación y división. La segunda fila consiste en signos de puntuación tomados de la gramática inglesa. Observe que la coma es un símbolo especial. En Java las comas se utilizan para separar elementos en una lista. Mientras que los puntos y comas se emplean para terminar una instrucción. La tercera fila contiene símbolos utilizados para hacer comparaciones. Observe que un espacio en blanco, que no se muestra antes, también es un símbolo especial. Un símbolo en blanco se crea presionando la barra de espacio (sólo una vez) en el teclado. La tercera fila consiste en símbolos compuestos de hasta dos caracteres, pero que se consideran individuales. Ningún carácter puede estar entre los dos caracteres en estos símbolos, ni siquiera un espacio en blanco.

Palabras reservadas (palabras clave)

Una segunda categoría de símbolos son las palabras reservadas. Algunas de estas incluyen las siguientes:

```
int, float, double, char, void, public, static, throws, return
```

Las palabras reservadas también se llaman **palabras clave**. Las letras en una palabra reservada siempre son minúsculas. Igual que los símbolos especiales, cada palabra reservada se considera un símbolo individual. Además, las palabras reservadas no se pueden redefinir dentro de algún programa; es decir, no se pueden utilizar para algo diferente de su uso propuesto. Para ver una lista de palabras reservadas en Java, consulte el apéndice A.

A lo largo de este libro las palabras reservadas se muestran en color `azul`.

Identificadores

Una tercera categoría de símbolos son los **identificadores**. Estos son nombres de cosas, como variables, constantes y métodos, que aparecen en los programas. Algunos identificadores están predefinidos, otros los define el usuario. Todos deben obedecer las reglas de los identificadores de Java.

Identificador: un identificador en Java consiste en letras, dígitos, el carácter guión bajo (_) y el signo de dólar ($) y debe iniciar con una letra, un guión bajo o el signo de dólar.

Los identificadores pueden contener letras, dígitos, el carácter guión bajo (_) y el signo de dólar $; no se permite otro símbolo para formar un identificador.

NOTA

Java es sensible a las letras mayúsculas y minúsculas: las letras mayúsculas y las minúsculas se consideran diferentes. Así pues, el identificador `NUMBER` no es el mismo que el identificador `number` o que el identificador `Number`. De manera similar, los identificadores `X` y `x` son diferentes.

En Java los identificadores pueden ser de cualquier longitud. Algunos identificadores predefinidos que encontrará con frecuencia son `print`, `println` y `printf`, los cuales se utilizan cuando se genera una salida y `nextInt`, `nextDouble`, `next` y `nextLine`, que se emplean para ingresar datos. A diferencia de las palabras reservadas, los identificadores predefinidos se pueden redefinir, pero no sería muy prudente hacerlo.

EJEMPLO 2-1

Los siguientes son identificadores legales en Java:

```
first
conversion
payRate
counter1
$Amount
```

En la tabla 2-1 se muestran algunos identificadores ilegales y se explica por qué lo son.

TABLA 2-1 Ejemplos de identificadores ilegales

Identificador ilegal	Descripción
`salario Empleado`	No puede haber un espacio entre `empleado` y `Salario`.
`¡Hola!`	El signo de admiración no se puede utilizar en un identificador.
`uno+dos`	El símbolo + no se puede utilizar en un identificador.
`2o`	Un identificador no puede comenzar con un dígito.

Tipos de datos

El objetivo de un programa en Java es manipular datos. Diferentes programas manipulan distintos datos. Un programa diseñado para calcular el cheque de pago de un empleado sumará, restará, multiplicará y dividirá números; algunos de los números pueden representar las horas trabajadas y el pago por hora. De manera similar, un programa diseñado para ordenar alfabéticamente una lista de clase manipulará nombres. No se esperaría que la receta de una tarta de cereza ayude a cocinar galletas. De igual forma, no se manipularían caracteres alfabéticos con un programa diseñado para realizar cálculos aritméticos. Además, no se multiplicarían o restarían nombres. Para reflejar esas diferencias subyacentes, Java categoriza los datos en diferentes tipos y sólo se pueden realizar ciertas operaciones sobre un tipo de dato particular. Al principio puede parecer confuso, pero al tener cuidado con los tipos, Java tiene verificaciones automáticas para proteger contra errores.

Tipos de datos: conjunto de valores unidos a un conjunto de operaciones sobre esos valores.

Tipos de datos primitivos

Los tipos de datos primitivos son fundamentales en Java. Hay tres categorías de tipos de datos primitivos:

- **Integrales,** son un tipo de dato que trata con enteros o números sin un punto decimal (y caracteres)
- **De punto flotante,** es un tipo de dato que trata con números decimales
- **Booleanos,** es un tipo de dato que trata con valores lógicos

Los tipos de datos integrales se clasifican adicionalmente en cinco categorías: `char`, `byte`, `short`, `int` y `long`.

¿Por qué hay tantas categorías de tipos de datos integrales? Cada tipo de dato tiene un conjunto de valores diferente asociado con él. Por ejemplo, el tipo de dato `int` se utiliza para representar enteros entre –2147483648 (= -2^{32}) y 2147483647 (= $2^{32} - 1$). El tipo de dato `short` se maneja para representar enteros entre –32768 (= -2^{15}) y 32767 (= $2^{15} - 1$).

Qué tipo de dato se debe emplear, depende de la magnitud de un número con el que tenga que tratar un programa. En los primeros días de la programación las computadoras y la memoria principal eran muy costosas. Sólo una cantidad pequeña de memoria estaba disponible para ejecutar programas y manipular datos. Como resultado, los programadores tenían que optimizar el uso de la memoria. Debido a que escribir un programa y hacerlo que funcione ya es en sí un proceso complicado, no tener que preocuparse por el tamaño de la memoria resulta una cosa menos en qué pensar. Para utilizar de manera efectiva la memoria, un programador puede consultar el tipo de dato utilizado en un programa y determinar qué tipo de dato usar. (Las restricciones de memoria aún pueden ser una preocupación en el caso de programas escritos para aplicaciones como un reloj de pulsera.)

En la tabla 2-2 se presenta el intervalo de valores posibles asociados con los cinco tipos de datos integrales y el tamaño de la memoria designada para manipular esos valores.

TABLA 2-2 Valores y asignación de memoria para tipos de datos integrales

Tipo de datos	Valores	Almacenamiento (en bytes)
`char`	0 a 65535 (= 216 – 1)	2 (16 bits)
`byte`	–128 (= -2^7) a 127 (= $2^7 - 1$)	1 (8 bits)
`short`	–32768 (= -2^{15}) a 32767 (= $2^{15} - 1$)	2 (16 bits)
`int`	–2147483648 (= -2^{31}) a 2147483647 (= $2^{31} - 1$)	4 (32 bits)
`long`	–9223372036854775808 (= -2^{63}) a 9223372036854775807 (= $2^{63} - 1$)	8 (64 bits)

El tipo de datos integrales de uso más común es **`int`**. Observe que la explicación siguiente del tipo de dato **`int`** también se aplica a los tipos integrales **`byte`**, **`short`** y **`long`**.

TIPO DE DATO `int`

En esta sección se describe el tipo de dato **`int`**, pero esta explicación también se aplica a otros tipos de datos integrales. Los enteros en Java, igual que en matemáticas, son números como los siguientes:

```
-6728,  -67,  0,  78,  36782,  +763
```

Observe las siguientes dos reglas de estos ejemplos:

- Los enteros positivos no requieren un signo + enfrente de ellos.
- No se utilizan comas dentro de un entero. Recuerde que en JAVA las comas se utilizan para separar elementos en una lista. Así pues, `36,782` se interpreta como dos enteros: `36` y `782`.

TIPO DE DATO `char`

Como se indica en la tabla 2-2, el tipo de dato **`char`** tiene 65 536 valores, de 0 a 65 535. Sin embargo, su propósito principal es representar caracteres individuales, es decir, letras, dígitos y símbolos especiales. Por tanto, el tipo de dato **`char`** puede representar cualquier tecla del teclado. Al utilizar el tipo de dato **`char`** se encierra cada carácter representado dentro de comillas simples. Ejemplos de valores que pertenecen al tipo de dato **`char`** incluyen los siguientes:

```
'A', 'a', '0', '*', '+', '$', '&', ' '
```

Observe que un espacio en blanco es un carácter y se escribe ' ', con un espacio entre las comillas simples.

El tipo de dato **`char`** *posibilita que sólo un símbolo* se coloque entre las comillas simples. Por tanto, el valor `'abc'` no es de tipo **`char`**. Además, aunque `!=` y símbolos especiales similares se consideran uno solo, no se consideran valores posibles del tipo de dato **`char`** cuando están

encerrados entre comillas simples. Todos los símbolos individuales ubicados en el teclado que se pueden imprimir se consideran valores posibles del tipo de dato **char**.

Como se indicó en el capítulo 1, cada carácter tiene una representación específica en la memoria de una computadora y hay varios esquemas de codificación diferentes para los caracteres. Java utiliza el conjunto de caracteres Unicode, el cual contiene 65 536 valores numerados del 0 al 65 535. La posición del primer carácter es 0, la del segundo carácter es 1 y así sucesivamente. Otros conjuntos de datos de caracteres son el American Standard Code for Information Interchange (ASCII) y el Extended Binary-Coded Decimal Interchange Code (EBCDIC). El conjunto de caracteres ASCII tiene 128 valores. El ASCII es un subconjunto de Unicode, es decir, los primeros 128 caracteres de Unicode son los mismos que los caracteres en ASCII. El conjunto de caracteres EBCDIC tiene 256 valores y lo desarrolló IBM.

Cada uno de los 65 536 valores del conjunto de caracteres Unicode representa un carácter diferente. Por ejemplo, el valor `65` representa `'A'`, y el valor `43` representa `'+'`. Por tanto, cada carácter tiene un valor entero no negativo específico en el conjunto de caracteres Unicode, el cual se denomina **secuencia de intercalación** del carácter. Se deduce que la secuencia de intercalación de `'A'` es `65`. La secuencia de intercalación se utiliza cuando se comparan caracteres. Por ejemplo, el valor que representa `'B'` es `66`, por tanto `'A'` es menor que `'B'`. De manera similar, `'+'` es menor que `'A'` ya que `43` es menor que `65`.

El 14o carácter en el conjunto de caracteres Unicode (y en ASCII) se denomina carácter de nueva línea y se representa `'\n'`. (Observe que la posición del carácter de nueva línea en los conjuntos de caracteres Unicode y ASCII es 13 debido a que la posición del primer carácter es 0.) Aunque el carácter de nueva línea es una combinación de dos caracteres, se trata como uno solo. De manera similar, el carácter de tabulador horizontal se representa en Java como `'\t'` y el carácter nulo se representa como `'\0'` (una diagonal invertida seguida de cero). (Más adelante en este capítulo se explican estos caracteres especiales.) Además, los primeros 32 caracteres en los conjuntos de caracteres Unicode y ASCII no son imprimibles. (Consulte el apéndice C para una lista de estos caracteres.)

TIPO DE DATO `boolean`

El tipo de dato **boolean** (booleano) tiene sólo dos valores: **true** y **false** (**verdadero** y **falso**). Además, **verdadero** y **falso** se llaman valores lógicos (booleanos). El objetivo principal de este tipo de dato es manipular expresiones lógicas (booleanas). Una expresión que se evalúa como **verdadera** o **falsa** se denomina **expresión lógica (booleana)**. Las expresiones lógicas (booleanas) se definen formalmente y se explican en detalle en el capítulo 4. En Java, **boolean**, **true** y **false** son palabras reservadas. La memoria asignada para el tipo de datos **booleanos** es 1 bit.

TIPO DE DATO DE PUNTO FLOTANTE

Para tratar con números decimales, Java proporciona el tipo de dato de punto flotante. Para facilitar nuestro análisis de este tipo de dato, se repasará un concepto de un curso de álgebra de la preparatoria o de la universidad.

Puede que esté familiarizado con la notación científica. Por ejemplo:

$43872918 = 4.3872918 * 10^{7}$
$.0000265 = 2.65 * 10^{-5}$
$47.9832 = 4.7983 * 10^{1}$

Para representar números reales, Java utiliza una forma de notación científica denominada **notación de punto flotante.** En la tabla 2-3 se muestra cómo Java podría imprimir un conjunto de números reales. En la notación de punto flotante en Java, la letra e representa el exponente.

TABLA 2-3 Ejemplos de números reales impresos en Java en notación de punto flotante

Número real	Notación de punto flotante en Java
75.924	7.592400e+01
0.18	1.800000e–01
0.0000453	4.530000e–05
–1.482	–1.482000e+00
7800.0	7.800000e+03

Java proporciona dos tipos de datos para representar números decimales: `float` y `double`. Igual que para los tipos de datos integrales, los tipos de datos `float` y `double` difieren en el conjunto de valores.

`float:` el tipo de dato `float` se utiliza en Java para representar cualquier número real entre –3.4E+38 y 3.4E+38. La memoria asignada para el tipo de dato `float` es de 4 bytes.

`double:` el tipo de datos `double` se utiliza en Java para representar cualquier número real entre –1.7E+308 y 1.7E+308. La memoria asignada para el tipo de dato `double` es de 8 bytes.

Además del conjunto de valores hay una diferencia más entre los tipos de datos `float` y `double`. El número máximo de cifras significativas, es decir, el número de lugares decimales, en valores `float` es 6 o 7. El número máximo de cifras significativas en valores que pertenecen al tipo `double` es por lo general 15. El número máximo de cifras significativas se denomina **precisión.** En ocasiones los valores `float` se denominan de **precisión simple** y los valores del tipo `double` se nombran de **precisión doble.**

NOTA

En Java, por designación, los números de punto flotante son considerados del tipo `double`. Por tanto, si se utiliza el tipo de datos `float` para representar números de punto flotante en un programa, se podría obtener una advertencia o un mensaje de error, como "truncado de double a float" o "posible perdida de datos". Para evitar esos mensajes, se debe utilizar el tipo de datos `double`. Para fines de ilustración y evitar esos mensajes en los ejemplos de programación, en este libro principalmente se utilizan tipos de datos `double` para representar números de punto flotante. Sin embargo, si se necesita utilizar claramente un valor `float` en un programa, se especifica que el valor decimal es un valor `float` empleando la letra f al final del número. Por ejemplo, 28.75f representa un valor `float` en tanto que 28.75 representa un valor `double`.

PROGRAMACIÓN JAVA

DEL ANÁLISIS DE PROBLEMAS AL DISEÑO DE PROGRAMAS

Diseñado para un primer curso de Java, *Programación Java del análisis de problemas al diseño de programas 5a. Ed.*, motivará a los alumnos mientras construyen una piedra angular para el plan de estudios de Ciencias de la Computación. Con un enfoque en el aprendizaje de los alumnos, este texto aborda la programación utilizando la última versión de Java, e incluye ejercicios de programación y programas actualizados. El estilo de escritura atractiva y clara ayudará a los estudiantes a aprender conceptos clave a través de explicaciones concisas y prácticas en este lenguaje complejo y de gran alcance.

CARACTERÍSTICAS:

+ Diagramas visuales: Más de 240 diagramas visuales ayudan a la comprensión de los lectores, e ilustran claramente los conceptos difíciles.

+ Código de programación con descripciones: el código de programación utilizado en los ejemplos se acompaña de una descripción de lo que cada línea del código hace, llevando a los lectores paso a paso a través del proceso de programación.

+ Ejemplos de programación: amplios ejemplos de programación muestran las etapas precisas y concretas de Entrada, Salida, Programa de Análisis y Diseño de Algoritmos, y un listado completo de programas, que desafían a los lectores para escribir programas Java con un resultado especificado.

ISBN-13: 978-607481926-7
ISBN-10: 607481926-2

CENGAGE Learning®

Visite nuestro sitio en http://latinoamerica.cengage.com