# 1.1 Su primer programa

En esta sección, nuestro plan es llevarlo al mundo de la programación Java guiándolo a través de los pasos básicos necesarios para ejecutar un programa simple. La *plataforma Java (en*  lo sucesivo abreviada *Java*) es una colección de aplicaciones, no muy diferente de muchas de las otras aplicaciones que está acostumbrado a usar (como su procesador de textos, programa de correo electrónico y navegador web). Al igual que con cualquier aplicación, debe asegurarse de que Java esté instalado correctamente en su computadora. Viene precargado en muchas computadoras, o puede descargarlo fácilmente. También necesita un editor de texto y una aplicación de terminal. Su primera tarea es encontrar las instrucciones para instalar dicho entorno de programación Java en *su* computadora visitando <http://introcs.cs.princeton.edu/java>

Nos referimos a este sitio como el sitio del *libro*. Contiene una gran cantidad de información complementaria sobre el material de este libro para su referencia y uso durante la programación.

**Programación en Java** Para introducirte en el desarrollo de programas Java, dividimos el proceso en tres pasos. Para programar en Java, necesitas:

* *Cree un*  programa escribiéndolo en un archivo llamado, por ejemplo, MiPrograma.java.
* *Compílelo* escribiendo javac MyProgram.java en una ventana de terminal.
* *Ejecútelo (o* ejecútelo*) escribiendo* java MyProgram en la ventana del terminal.

En el primer paso, comienzas con una pantalla en blanco y terminas con una secuencia de caracteres escritos en la pantalla, al igual que cuando redactas un mensaje de correo electrónico o un ensayo. Los programadores utilizan el término código para referirse al texto del programa y el término *codificación* para referirse al acto de crear y editar el código. En el segundo paso, utiliza una aplicación del sistema que *compila* su programa (lo traduce a una forma más adecuada para la computadora) y coloca el resultado en un archivo llamado MyProgram.class. En el tercer paso, transfiere el control de la computadora del sistema a su programa (que devuelve el control al sistema cuando termina). Muchos sistemas tienen varias formas diferentes de crear, compilar y ejecutar programas. Elegimos la secuencia dada aquí porque es la más simple de describir y usar para programas pequeños.

*Creación de un programa.* Un programa Java no es más que una secuencia de caracteres, como un párrafo o un poema, almacenados en un archivo con una extensión .java. Para crear uno, por lo tanto, simplemente necesita definir esa secuencia de caracteres, de la misma manera que lo hace para el correo electrónico o cualquier otra aplicación informática. Puede usar cualquier *editor de texto* para esta tarea, o puede usar uno de los entornos de *desarrollo integrado más sofisticados*  descritos en el sitio del libro. Estos entornos son excesivos para el tipo de programas que consideramos en este libro, pero no son difíciles de usar, tienen muchas características útiles y son ampliamente utilizados por los profesionales.

*Compilación de un programa.* Al principio, podría parecer que Java está diseñado para ser mejor entendido por la computadora. Por el contrario, el lenguaje está diseñado para ser mejor entendido por el programador, es decir, por ti. El lenguaje de la computadora es mucho más primitivo que Java. Un *compilador* es una aplicación que traduce un programa del lenguaje Java a un lenguaje más adecuado para su ejecución en el ordenador. El compilador toma un archivo con una extensión .java como entrada (su programa) y produce un archivo con el mismo nombre pero con una extensión .class (la versión del lenguaje de la computadora). Para usar su compilador Java, escriba en una ventana de terminal el comando javac seguido del nombre de archivo del programa que desea compilar.

*Ejecución (ejecución) de un programa.* Una vez compilado el programa, puede ejecutarlo (o ejecutarlo). Esta es la parte emocionante, donde su programa toma el control de su computadora (dentro de las limitaciones de lo que permite Java). Quizás sea más exacto decir que su computadora sigue sus instrucciones. Es aún más exacto decir que una parte de Java conocida como Java *Virtual Machine* (*JVM*, para abreviar) dirige a su computadora para que siga sus instrucciones. Para utilizar la JVM para ejecutar el programa, escriba el comando java seguido del nombre del programa en una ventana de terminal.

*Utilice cualquier editor de texto para tipo javac HelloWorld.java tipo Java HelloWorld*

*Crea tu programa Para compilar el programa Para ejecutar el programa*

HelloWorld.java HelloWorld.class"Hola, Mundo"

*editor*

*compilador*

*JVM*

*Su programa lenguaje-informático salida*

*(un archivo de texto) versión de su programa*

*Desarrollo de un programa Java*

%

**javac HelloWorld.java**

%

**Java HelloWorld**

Hola mundo

***Programa 1.1.1***

***H***

***ello, World***

clase pública HelloWorld

{

public static void main(String[] args)

{

Imprime "Hola, Mundo" en la ventana del terminal.

System.out.println("Hola, Mundo");

}

}

*Este código es un programa Java que realiza una tarea simple. Tradicionalmente es la primera vez para principiantes*

*programa. El siguiente cuadro muestra lo que sucede cuando compila y ejecuta el programa. El*

*La aplicación de terminal proporciona un símbolo del sistema (*

*%*

*en este libro) y ejecuta las órdenes*

*que escriba (*

*Javac*

*Y entonces*

*Java*

*en el siguiente ejemplo). Nuestra convención es destacar en*

*Negrita: el texto que escribe y muestra los resultados en una cara normal. En este caso, el resultado es*

*que el programa imprime el mensaje*

*Hola mundo*

*en la ventana del terminal.*

El programa 1.1.1 es un ejemplo de un programa Java completo. Su nombre es HelloWorld, lo que significa que su código reside en un archivo llamado HelloWorld.java (por convención en Java). La única acción del programa es imprimir un mensaje en la ventana del terminal. Para dar continuidad, usaremos algunos términos estándar de Java para describir el programa, pero no los definiremos hasta más adelante en el libro: el programa 1.1.1 consiste en una sola *clase* llamada HelloWorld que tiene un solo *método* llamado main(). (Cuando nos referimos a un método en el texto, usamos () después del nombre para distinguirlo de otros tipos de nombres.) Hasta la sección 2.1, todas nuestras clases tendrán esta misma estructura. Por el momento, puedes pensar en "clase" como "programa".

La primera línea de un método especifica su nombre y otra información; el resto es una secuencia de *instrucciones* encerradas entre llaves, con cada instrucción típicamente seguida de un punto y coma. Por el momento, se puede pensar en "programación" como "especificar un nombre de clase y una secuencia de sentencias para su método main()", con el corazón del programa consistiendo en la secuencia de sentencias del método main() (su cuerpo*).* El programa 1.1.1 contiene dos de estas declaraciones:

• El primer enunciado es un *comentario*, que sirve para documentar el programa. En Java, un comentario de una sola línea comienza con dos caracteres '/' y se extiende hasta el final de la línea. En este libro, mostramos los comentarios en gris. Java ignora los comentarios, ya que solo están presentes para los lectores humanos del programa. • La segunda instrucción es una *declaración impresa*. Llama al método llamado System.out.println() para imprimir un mensaje de texto, el especificado entre las comillas dobles coincidentes, en la ventana del terminal.

En las siguientes dos secciones, aprenderá acerca de muchos tipos diferentes de instrucciones que puede usar para crear programas. Por el momento, solo usaremos comentarios y declaraciones impresas, como las de HelloWorld.

Cuando escribes java seguido de un nombre de clase en la ventana de tu terminal, el sistema llama al método main() que definiste en esa clase y ejecuta sus sentencias en orden, una por una. Por lo tanto, escribir java HelloWorld hace que el sistema llame al método main() en el programa 1.1.1 y ejecute sus dos instrucciones. La primera instrucción es un comentario, que Java ignora. La segunda instrucción imprime el

mensaje especificado a la ventana del terminal.

{

*main()*

*método*

*cuerpo*

*nombre*

*Declaraciones*

*archivo de texto llamado*

*HelloWorld.java*

Hola Mundo

Clase Pública

public static void main(String[] args)

{

Imprime "Hola, Mundo" en la ventana del terminal.

System.out.print("Hola, mundo");

}

}

*Anatomía de un programa*

Desde la década de 1970, ha sido una tradición que el primer programa de un programador principiante imprima Hello, World. Por lo tanto, debe escribir el código del programa 1.1.1 en un archivo, compilarlo y ejecutarlo. Al hacerlo, estarás siguiendo los pasos de muchos otros que han aprendido a programar. Además, comprobará que tiene un editor y una aplicación de terminal utilizables. Al principio, realizar la tarea de imprimir algo en una ventana de terminal puede no parecer muy interesante; Sin embargo, después de reflexionar, verá que una de las funciones más básicas que necesitamos de un programa es su capacidad para decirnos lo que está haciendo.

Por el momento, todo el código de nuestro programa será igual que el programa 1.1.1, excepto con una secuencia diferente de sentencias en main(). Por lo tanto, no es necesario comenzar con una página en blanco para escribir un programa. En su lugar, puede

* Copie HelloWorld.java en un nuevo archivo que tenga un nuevo nombre de programa de su elección, seguido de .java.
* Reemplace HelloWorld en la primera línea con el nuevo nombre del programa.
* Reemplace las instrucciones comment e print por una secuencia diferente de instrucciones.

Su programa se caracteriza por su secuencia de instrucciones y su nombre. Cada

El programa Java debe residir en un archivo cuyo nombre coincida con el que sigue a la palabra clase en la primera línea, y también debe tener una extensión .java.

*Errores.* Es fácil difuminar las distinciones entre editar, compilar y ejecutar programas. Debes mantener estos procesos separados en tu mente cuando estés aprendiendo a programar, para comprender mejor los efectos de los errores que inevitablemente surgen.

Puede corregir o evitar la mayoría de los errores examinando cuidadosamente el programa a medida que lo crea, de la misma manera que corrige los errores ortográficos y gramaticales cuando redacta un mensaje de correo electrónico. Algunos errores, conocidos como errores *en tiempo de compilación* , se identifican cuando compilas el programa, porque impiden que el compilador haga la traducción. Otros errores, conocidos como errores *en tiempo de ejecución* , no aparecen hasta que se ejecuta el programa.

En general, los errores en los programas, también conocidos comúnmente como *bugs, son la pesadilla de la* existencia de un programador: los mensajes de error pueden ser confusos o engañosos, y la fuente del error puede ser muy difícil de encontrar. Una de las primeras habilidades que aprenderás es identificar errores; También aprenderá a ser lo suficientemente cuidadoso al codificar, para evitar hacer muchos de ellos en primer lugar. Puede encontrar varios ejemplos de errores en las preguntas y respuestas al final de esta sección.

% j

**avac UseArgument.java**

%

**java UseArgument Alice**

Hola, Alicia. ¿Cómo estás?

%

**java useargument bob**

Hola, Bob. ¿Cómo estás?

***Programa 1.1.2***

***U***

***Cantar un argumento de línea de comandos***

public class UseArgument

{

public static void main(String[] args)

{

System.out.print("Hola, ");

System.out.print(args[0]);

System.out.println(". ¿Cómo estás?");

}

}

*Este programa muestra la forma en que podemos controlar las acciones de nuestros programas: proporcionando*

*Un argumento en la línea de comandos. Hacerlo nos permite adaptar el comportamiento de nuestros programas.*

**Entrada y salida** Por lo general, queremos proporcionar *información* a nuestros programas, es decir, datos que puedan procesar para producir un resultado. La forma más sencilla de proporcionar datos de entrada se ilustra en UseArgument (programa 1.1.2). Cada vez que se ejecuta el programa UseArgument, se acepta el *argumento de la línea de comandos* que se escribe después del nombre del programa y se imprime de nuevo en la ventana del terminal como parte del mensaje. El resultado de la ejecución de este programa depende de lo que escriba después del nombre del programa. Al ejecutar el programa con diferentes argumentos de línea de comandos,

Produce diferentes resultados impresos. Discutiremos con más detalle el mecanismo que usamos para pasar argumentos de línea de comandos a nuestros programas más adelante, en la sección 2.1. Por ahora es suficiente entender que args[0] es el primer argumento de la línea de comandos que se escribe después del nombre del programa, args[1] es el segundo, y así sucesivamente. Por lo tanto, puede usar args[0] dentro del cuerpo de su programa para representar la primera cadena que escriba en la línea de comandos cuando se ejecute, como en UseArgument.

Además del método System.out.println(), UseArgument llama al método System.out.print(). Este método es igual que System.out.println(), pero imprime solo la cadena especificada (y no un carácter de nueva línea).

Una vez más, llevar a cabo la tarea de hacer que un programa imprima lo que escribimos en él puede no parecer interesante al principio, pero al reflexionar te darás cuenta de que otra función básica de un programa es su capacidad para responder a la información básica del usuario para controlar lo que hace el programa. El modelo simple que representa UseArgument será suficiente para permitirnos considerar el mecanismo básico de programación de Java y abordar todo tipo de problemas computacionales interesantes.

Dando un paso atrás, podemos ver que UseArgument no hace ni más ni menos que implementar una función que asigna una cadena de caracteres (el argumento de la línea de comandos) a otra cadena de caracteres (el mensaje impreso de nuevo en la ventana del terminal). Al usarlo, podríamos pensar en nuestro programa Java como una caja negra que convierte nuestra cadena de entrada en una cadena de salida.

Este modelo es atractivo porque no solo es simple, sino también lo suficientemente general como para permitir *una cadena de entrada completa*

Alice

ción, en principio, de cualquier tarea computacional. Por ejemplo, el compilador Java en sí no es más que un programa que toma una cadena de caracteres como entrada de *caja negra*  (un archivo .java) y produce otra cadena de caracteres de *cadena*  de salida como salida (el archivo .class correspondiente ).

Más tarde, serás capaz de escribir programas que se adapten a la hola, Alice. ¿Cómo estás? A pesar de que no llegamos a programas tan complicados como un compilador, no llegamos a una variedad de tareas interesantes. Para *una vista de pájaro de un programa Java* el momento, viviremos con varias limitaciones en

El tamaño y tipo de entrada y salida a nuestros programas; en la sección 1.5, verá cómo incorporar mecanismos más sofisticados para la entrada y salida del programa. En particular, verás que podemos trabajar con cadenas de entrada y salida arbitrariamente largas y otros tipos de datos como sonido e imágenes.

***Preguntas y respuestas***

**Pregunta.** ¿Por qué Java?

**Un.** Los programas que estamos escribiendo son muy similares a sus contrapartes en varios otros idiomas, por lo que nuestra elección de idioma no es crucial. Usamos Java porque está ampliamente disponible, abarca un conjunto completo de abstracciones modernas y tiene una variedad de comprobaciones automáticas de errores en los programas, por lo que es adecuado para aprender a programar. No existe un lenguaje perfecto, y sin duda programará en otros lenguajes en el futuro.

**Pregunta.** ¿Realmente tengo que escribir los programas en el libro para probarlos? Creo que los ejecutó y que producen el resultado indicado.

**Un.** Todo el mundo debería escribir y ejecutar HelloWorld. Su comprensión se ampliará enormemente si también ejecuta UseArgument, lo prueba en varias entradas y lo modifica para probar diferentes ideas propias. Para ahorrarte algo de escritura, puedes encontrar todo el código de este libro (y mucho más) en el sitio web. Este sitio también tiene información sobre cómo instalar y ejecutar Java en su computadora, respuestas a ejercicios seleccionados,

enlaces web y otra información adicional que puede resultarle útil durante la programación.

**Pregunta.** ¿Cuál es el significado de las palabras público, estático y vacío?

**Un.** Estas palabras clave especifican ciertas propiedades de main() que aprenderá más adelante en el libro. Por el momento, solo incluimos estas palabras clave en el código (porque son obligatorias) pero no nos referimos a ellas en el texto.

**Pregunta.** ¿Cuál es el significado de las secuencias de caracteres //, /\* y \*/ en el código?

**Un.** Denotan *comentarios*, que son ignorados por el compilador. Un comentario es texto entre /\* y \*/ o al final de una línea después de //. Los comentarios son indispensables porque ayudan a otros programadores a entender tu código e incluso pueden ayudarte a entender tu propio código en retrospectiva. Las limitaciones del formato de libro exigen que usemos los comentarios con moderación en nuestros programas; En su lugar, describimos cada programa a fondo en el texto y las figuras que lo acompañan. Los programas en el sitio web se comentan en un grado más realista.

**Pregunta.** ¿Cuáles son las reglas de Java con respecto a las tabulaciones, los espacios y los caracteres de nueva línea?

**Un.** Estos caracteres se conocen como caracteres *de espacio en blanco* . Los compiladores de Java consideran que todos los espacios en blanco en el texto del programa son equivalentes. Por ejemplo, podríamos escribir HelloWorld de la siguiente manera:

public class HelloWorld { public static void main ( String [] args) { System.out.println("Hola, Mundo") ; } }

Pero normalmente nos adherimos a las convenciones de espaciado y sangría cuando escribimos

Java, al igual que sangramos párrafos y líneas de forma coherente cuando escribimos prosa o poesía.

**Pregunta.** ¿Cuáles son las reglas con respecto a las comillas?

**Un.** El material entre comillas dobles es una excepción a la regla definida en la pregunta anterior: normalmente, los caracteres entre comillas se toman

literalmente para que Puede especificar con precisión lo que se imprime. Si coloca cualquier número de espacios sucesivos dentro de las comillas, obtendrá ese número de espacios en la salida. Si omite accidentalmente una comilla, el compilador puede confundirse mucho, ya que necesita esa marca para distinguir entre los caracteres de la cadena y otras partes del programa.

**Pregunta.** ¿Qué sucede cuando omite una llave o escribe mal una de las palabras, como public o static o void o main?

**Un.** Depende precisamente de lo que hagas. Estos errores se denominan errores *de sintaxis* y, por lo general, son detectados por el compilador. Por ejemplo, si hace que un programa sea malo que sea exactamente igual que HelloWorld, excepto que omite la línea que contiene la primera llave izquierda (y cambia el nombre del programa de HelloWorld a malo), recibirá el siguiente mensaje útil:

% **javac Bad.java**

Bad.java:1: error: '{' esperado

clase pública Malo ^

1 error

A partir de este mensaje, puede suponer correctamente que necesita insertar una llave izquierda. Sin embargo, es posible que el compilador no pueda decirle exactamente qué error cometió, por lo que el mensaje de error puede ser difícil de entender. Por ejemplo, si omite la segunda llave izquierda en lugar de la primera, recibirá el siguiente mensaje:

% **javac Bad.java**

Bad.java:3: error: ';' expected public static void main(String[] args) ^

Bad.java:7: error: class, interface o enum expected

}

^

2 errores

Una forma de acostumbrarse a este tipo de mensajes es introducir intencionadamente errores en un programa sencillo y luego ver qué pasa. Independientemente de lo que diga el mensaje de error, debe tratar al compilador como a un amigo, porque solo está tratando de decirle que algo anda mal con su programa.

**Pregunta.** ¿Qué métodos Java están disponibles para mí?

**Un.** Hay miles de ellos. Te los presentamos de manera deliberada (comenzando en la siguiente sección) para evitar abrumarte con opciones.

**Pregunta.** Cuando ejecuté UseArgument, recibí un extraño mensaje de error. ¿Cuál es el problema?

**Un.** Lo más probable es que hayas olvidado incluir un argumento de línea de comandos:

% **java useargumento**

Hola, Excepción en el hilo "main" java.lang.ArrayIndexOutOfBoundsException: 0 at UseArgument.main(UseArgument.java:6)

Java se queja de que ejecutó el programa pero no escribió un argumento de línea de comandos como se prometió. Aprenderá más detalles sobre los índices de matriz en la sección 1.4. Recuerde este mensaje de error, es probable que lo vuelva a ver. Incluso los programadores experimentados se olvidan de escribir argumentos de línea de comandos en ocasiones.

***Ejercicios***

**1.1.1** Escriba un programa que imprima el mensaje "Hola, mundo" 10 veces.

**1.1.2** Describa lo que sucede si omite lo siguiente en HelloWorld.java:

1. público
2. estático
3. vacío
4. args

**1.1.3** Describa lo que sucede si escribe mal (por ejemplo, omitiendo la segunda letra) lo siguiente en HelloWorld.java:

1. público
2. estático
3. vacío
4. args

**1.1.4** Describa lo que sucede si coloca las comillas dobles en la instrucción print de HelloWorld.java en líneas diferentes, como en este fragmento de código:

System.out.println("Hola,

Mundo");

**1.1.5** Describa lo que sucede si intenta ejecutar UseArgument con cada una de las siguientes líneas de comandos:

1. java UseArgument java
2. java UseArgument @!&^%
3. java UseArgument 1234
4. java UseArgument.java Bob
5. java UseArgument Alice Bob

**1.1.6** Modifique UseArgument.java para crear un programa UseThree.java que tome tres nombres como argumentos de línea de comandos e imprima una oración adecuada con los nombres en el orden inverso al dado, de modo que, por ejemplo, java UseThree Alice Bob Carol imprime Hi Carol, Bob y Alice.

*Esta página se ha dejado en blanco intencionadamente*

# 1.2 Tipos de datos incorporados

|  |  |
| --- | --- |
| 1.2.1 | Concatenación de cadenas . 20 |
| 1.2.2 | Multiplicación y división de enteros 23 |
| 1.2.3 | Fórmula cuadrática . 25 |
| 1.2.4 | Año bisiesto.. 28 |
| 1.2.5 | Conversión para obtener un entero aleatorio . . 34  *Programas de esta sección* |

Al programar en Java, siempre debe tener en cuenta el tipo de datos que su programa está procesando. Los programas de la sección 1.1 procesan cadenas de caracteres, muchos de los programas de esta sección procesan números, y consideramos muchos otros tipos más adelante en el libro. Comprender las distinciones entre ellos es tan importante que definimos formalmente la idea:  *un tipo de datos* es un conjunto de valores  *y un* conjunto de operaciones definidas sobre esos valores. Está familiarizado con varios tipos de números, como los números enteros y los números reales, y con las operaciones definidas en ellos, como la suma y la multiplicación. En matemáticas, estamos acostumbrados a pensar que los conjuntos de números son infinitos; En los programas informáticos tenemos que trabajar con un número finito de posibilidades. Cada operación que realizamos está bien definida *solo* para el conjunto finito de valores en un tipo de datos asociado.

Hay ocho tipos *primitivos de*  datos en Java, principalmente para diferentes tipos de números. De los ocho tipos primitivos, la mayoría de las veces usamos estos: int para números enteros; double para números reales; y booleano para valores verdaderos-falsos. Hay otros tipos de datos disponibles en las bibliotecas Java: por ejemplo, los programas de la sección 1.1 utilizan el tipo String para las cadenas de caracteres. Java trata el tipo String de manera diferente a otros tipos porque su uso para la entrada y la salida es esencial. En consecuencia, comparte algunas características de los tipos primitivos; por ejemplo, algunas de sus operaciones están integradas en el lenguaje Java. Para mayor claridad, nos referimos a los tipos primitivos y a String colectivamente como tipos integrados. Por el momento, nos concentramos en programas que se basan en la computación con tipos incorporados. Más adelante, aprenderá sobre los tipos de datos de la biblioteca Java y la creación de sus propios tipos de datos. De hecho, la programación en Java a menudo se centra en la construcción de tipos de datos, como verá en el capítulo 3.

Después de definir los términos básicos, consideramos varios programas de ejemplo y fragmentos de código que ilustran el uso de diferentes tipos de datos. Estos fragmentos de código no hacen mucha computación real, pero pronto verá código similar en programas más largos. Comprender los tipos de datos (valores y operaciones en ellos) es un paso esencial para comenzar a programar. Prepara el escenario para que comencemos a trabajar con programas más intrincados en la siguiente sección. Cada programa que escribas usará código como los pequeños fragmentos que se muestran en esta sección.

|  |  |  |
| --- | --- | --- |
| *tipo* | *Conjunto de valores Operadores comunes* | *Valores literales de ejemplo* |
| Int | Enteros + - \* / % | 99 12 2147483647 |
| doble | Números de coma flotante + - \* / | 3,14 2,5 6,022e23 |
| booleano | Valores booleanos && || ! | Verdadero Falso |
| carbonizar | Caracteres | 'A' '1' '%' '\n' |
| Cuerda | Secuencias de caracteres +  *Tipos de datos integrados básicos* | "AB" "Hola" "2.5" |

**Terminología** Para hablar de tipos de datos, necesitamos introducir algo de terminología.

Para ello, comenzamos con el siguiente fragmento de código:

int a, b, c; a = 1234; b = 99; c = a + b;

La primera línea es una *instrucción de declaración* que declara los nombres de tres *variables* que utilizan los *identificadores* a, b y c y su tipo es int. Las siguientes tres líneas son *instrucciones de asignación* que cambian los valores de las variables, utilizando *los literales* 1234 y 99, y la expresión *a + b*, con el resultado final de que c tiene el valor 1333.

*Literales.* Un *literal* es una representación en código Java de un valor de tipo de datos. Usamos secuencias de dígitos como 1234 o 99 para representar valores de tipo int; agregamos un punto decimal, como en 3.14159 o 2.71828, para representar valores de tipo double; usamos las palabras clave true o false para representar los dos valores de tipo booleano; y usamos secuencias de caracteres encerrados entre comillas coincidentes, como "Hola, Mundo", para representar valores de tipo String.

*Operadores.* Un *operador* es una representación en código Java de una operación de tipo de datos. Java usa + y \* para representar la suma y la multiplicación de números enteros y de punto flotante; Java usa &&, ||, y ! para representar operaciones booleanas, etcétera. Más adelante en esta sección describiremos los operadores más utilizados en los tipos integrados.

*Identificadores.* Un *identificador* es una representación en código Java de un nombre (por ejemplo, para una variable). Cada identificador es una secuencia de letras, dígitos, guiones bajos y símbolos de moneda, el primero de los cuales no es un dígito. Por ejemplo, las secuencias de caracteres abc, Ab$, abc123 y a\_b son identificadores Java legales, pero Ab\*, 1abc y a+b no lo son. Los identificadores distinguen entre mayúsculas y minúsculas, por lo que Ab , ab y AB son nombres diferentes. Algunas *palabras reservadas*, como public, static, int, double, String, true, false y null , son especiales y no se pueden usar como identificadores.

*Variables.* Una *variable* es una entidad que contiene un valor de tipo de datos, al que podemos referirnos por su nombre. En Java, cada variable tiene un tipo específico y almacena uno de los valores posibles de ese tipo. Por ejemplo, una variable int puede almacenar el valor 99 o 1234, pero no 3.14159 o "Hola, mundo". Diferentes variables del mismo tipo pueden almacenar el mismo valor. Además, como su nombre indica, el valor de una variable puede *cambiar* a medida que se desarrolla un cálculo. Por ejemplo, usamos una variable llamada suma en varios programas de este libro para mantener la suma continua de una secuencia de números. Creamos variables usando *declaraciones* y calculamos con ellas en *expresiones*, como se describe a continuación.

*Declaraciones de declaración.* Para crear una variable en Java, se utiliza  *el nombre de la variable de tipo, una declaración de declaración,* o simplemente *una declaración para abreviar Una declaración incluye un*  *tipo seguido de un nombre de variable*. Java reserva suficiente memoria para almacenar un valor de tipo de datos del valor especificado

doble total;

type y asocia el nombre de la variable con esa área de la instrucción MemDeclaration ory, de modo que pueda acceder al valor cuando se utilice la variable en código posterior. Por economía, puede declarar varias variables  *de Anatomía de una declaración* del mismo tipo en una sola instrucción de declaración.

*Convenciones de nomenclatura de variables.* Los programadores suelen seguir convenciones estilísticas a la hora de nombrar las cosas. En este libro, nuestra convención es dar a cada variable un nombre significativo que consiste en una letra minúscula seguida de letras minúsculas, letras mayúsculas y dígitos. Usamos letras mayúsculas para marcar las palabras de un nombre de variable de varias palabras. Por ejemplo, usamos los nombres de las variables i, x, y, sum,

isLeapYear, y outDegrees, entre muchos otros. Los programadores se refieren a este estilo de nomenclatura como *caso de camello*.

*Variables constantes.* Usamos el término oxímoron  *variable constante* para describir una variable cuyo valor no cambia durante la ejecución de un programa (o de una ejecución del programa a la siguiente). En este libro, nuestra convención es dar a cada variable constante un nombre que consiste en una letra mayúscula seguida de letras mayúsculas, dígitos y guiones bajos. Por ejemplo, podríamos usar los nombres de variables constantes SPEED\_OF\_LIGHT y DARK\_RED.

*Expresiones.* Una *expresión* es una combinación de literales, variables y operaciones que Java *evalúa* para producir un valor. En el caso de los tipos primitivos, las expresiones suelen tener el mismo aspecto que las fórmulas matemáticas, utilizando *operadores* para especificar las operaciones de tipo de datos que se van a realizar en un *operando más*. La mayoría de los operadores que usamos son operadores *binarios* que toman exactamente dos operandos, como x - 3 o 5 \* x. Cada operando puede ser cualquier expresión, quizás entre paréntesis. Por ejemplo, podemos escribir 4 \* (x - 3) o 5 \* x - 6 y Java entenderá lo que queremos decir. Una expresión es una directiva para realizar

*Operandos*

*(y expresiones)*

4 \* ( x - 3 )

*operador*

|  |
| --- |
| una secuencia de operaciones; La expresión es una representación del valor resultante.  *Precedencia del operador.* Una expresión es la abreviatura de una secuencia de operaciones: ¿en qué orden se deben aplicar los operadores? Java tiene reglas *de precedencia naturales y bien definidas*  que especifican completamente este orden. Para las operaciones aritméticas, la multiplicación y la división se realizan antes de la suma y la resta, de modo que a - b \* c y a - (b \* c) representan la misma secuencia de operaciones. Cuando los operadores aritméticos tienen la misma precedencia, el orden está determinado por la *asociatividad izquierda*, de modo que a - b - c y (a - b) - c representan la misma secuencia de operaciones. Puede usar paréntesis para anular las reglas, por lo que puede escribir a - (b - c) si eso es lo que desea. Es posible que en el futuro se encuentre con algún código Java que dependa sutilmente de las reglas de precedencia, pero en este libro usamos paréntesis para evitar dicho código. Si está interesado, puede encontrar todos los detalles sobre las reglas en el sitio web.  *Declaraciones de asignación.* Una instrucción de asignación asocia un valor de tipo de datos con una variable. Cuando escribimos c = a + b en Java, no estamos expresando matemáticas. |

*Anatomía de una expresión*

En su lugar, están expresando una *acción*: establezca el valor de la variable c como el valor de a más el valor de b. Es cierto que el valor de c es matemáticamente igual al valor de a + b inmediatamente después de que se haya ejecutado la declaración de asignación, pero el objetivo de la declaración es cambiar (o inicializar) el valor de c. El lado izquierdo de una instrucción de asignación debe ser una sola variable; El lado derecho puede ser cualquier expresión que genere un valor de un tipo compatible. Así, por ejemplo, tanto 1234 = a; y a + b = b + a; son sentencias no válidas en Java. En resumen, el *significado de = decididamente no es el mismo que en las ecuaciones matemáticas*.

*Declaración de declaración*

*nombre de la variable a, b; literal a* = 1234 ;

*asignación*

*enunciadob = 99;* int c = a + b;

*Inicialización en línea*

*declaración*

*Uso de un tipo de datos primitivo*

*Inicialización en línea.* Antes de poder utilizar una variable en una expresión, primero debe declarar la variable y asignarle un valor inicial. Si no se realiza ninguna de las dos cosas, se producirá un error en tiempo de compilación. Por economía, puede combinar una instrucción de declaración con una instrucción de asignación en una construcción conocida como instrucción de *inicialización en línea*. Por ejemplo, el código siguiente declara dos variables a y b, y las inicializa en los valores 1234 y 99, respectivamente: int a = 1234; int b = 99;

La mayoría de las veces, declaramos e inicializamos una variable de esta manera  *en*  el punto de su primer uso en nuestro programa.

|  |  |  |
| --- | --- | --- |
| b son variables int): int t = a; a = b; b = t;  Para ello, utilice un método tradicional para examinar el comportamiento del programa: estudie una tabla de los valores de las variables después de cada instrucción (una tabla de este tipo se conoce como *seguimiento*). | 1. b   int a, b; *indefinido indefinido*  a = 1234; 1234 *indefinido*   1. = 99; 1234 99 int t = a; 1234 99 a = b; 99 99 b = t; 99 1234   *Tu primer rastro* | t  1234  1234  1234 |

*Seguimiento de los cambios en los valores de las variables.* Como comprobación final de su comprensión del propósito de las instrucciones de asignación, convénzase de que el siguiente código *intercambia* los valores de a y b (supongamos que a y

*Seguridad de tipos.* Java requiere que declares el tipo de cada variable. Esto permite

Java para comprobar si hay errores de discrepancia de tipos en tiempo de compilación y alertarle de posibles errores en el programa. Por ejemplo, no se puede asignar un valor double a un int

variable, multiplique una cadena con un valor booleano o utilice una variable no inicializada dentro de una expresión. Esta situación es análoga a asegurarse de que las cantidades tengan las unidades adecuadas en una aplicación científica (por ejemplo, no tiene sentido sumar una cantidad medida en pulgadas a otra medida en libras).

A continuación, consideramos estos detalles para los tipos integrados básicos que usará con más frecuencia (cadenas, enteros, números de punto flotante y valores verdadero-falso), junto con el código de ejemplo que ilustra su uso. Para comprender cómo usar un tipo de datos, debe conocer no solo su conjunto definido de valores, sino también qué operaciones puede realizar, el mecanismo de lenguaje para invocar las operaciones y las convenciones para especificar literales.

**Caracteres y cadenas** El tipo char representa *valores individualescaracteres* ual caracteres alfanuméricos o símbolos, como los que *se utilizan típicamente*'a'

tipo. Hay 216 valores de caracteres diferentes posibles, pero solemos restringir la atención a los que representan letras, números,

*Los símbolos de tipo de datos char incorporados en Java*  y los caracteres de espacio en blanco como tabulación y nueva línea. Puede especificar un literal char encerrando un carácter entre comillas simples; por ejemplo, 'a' representa la letra a. Para tabulación, nueva línea, barra invertida, comillas simples y comillas dobles, usamos las secuencias de *escape especiales \t, \n, \\*, \' y \", respectivamente. Los caracteres se codifican como enteros de 16 bits utilizando un esquema de codificación conocido como *Unicode*, y también hay secuencias de escape para especificar caracteres especiales que no se encuentran en el teclado (consulte el sitio del libro). Por lo general, no realizamos ninguna operación directamente en los caracteres que no sea asignar valores a

Variables.

|  |  |
| --- | --- |
| El tipo String representa secuencias de caracteres. Puede especificar un literal String encerrando una secuencia de caracteres entre comillas dobles, como "Hola, mundo". El tipo de datos String *no es*  un tipo primitivo, pero Java a veces lo trata como tal. Por ejemplo, el operador de *concatenación* (+) toma dos operandos String y produce un tercer String que se forma anexando los caracteres de la variable | *valoressecuencias de caracteres típicos*"Hola, mundo" *literales* " \* "  *Operador+ de OperationConcatenate*  *Tipo de datos String integrado en Java* |

segundo operando a los caracteres del primer operando.

|  |  |
| --- | --- |
| *expresión valor*  "Hola" + "Bob" "Hola, Bob"  "1" + " 2 " + "1" "1 2 1"  "1234" + " + " + "99" "1234 + 99"  "1234" + "99" "123499"  *Expresiones típicas de String* | programa, duplica el tamaño de la salida. Específicamente, si el programa imprime *n* líneas, la *enésima línea contiene 2n1 números. Por ejemplo, si tuviera que agregar instrucciones de esta manera para que el programa imprima 30 líneas, imprimiría más de mil millones de* números. |

La operación de concatenación (junto con la capacidad de declarar variables String y usarlas en expresiones e instrucciones de asignación) es lo suficientemente potente como para permitirnos atacar algunas tareas informáticas no triviales. Por ejemplo, Ruler (programa 1.2.1) calcula una tabla de valores de la *función ruler* que describe las longitudes relativas de las marcas en una regla. Una característica notable de este cálculo es que ilustra lo fácil que es crear un programa corto que produzca una gran cantidad de resultados. Si extiende este programa de la manera obvia para imprimir cinco líneas, seis líneas, siete líneas, etc., verá que cada vez que agregue dos declaraciones a esto

%

**javac Ruleta.java**

%

**java Regla**

1

2 1

1

2 1 3 1 2 1

1

1

2 1 3 1 2 1 4 1 2 1 3 1 2

1

***Programa 1.2.1***

***S***

***Concatenación de TRING***

Gobernante de clase pública

{

public static void main(String[] args)

{

Regla de cadena1 = "1";

Cadena ruler2 = ruler1 + " 2 " + ruler1;

Cadena ruler3 = ruler2 + " 3 " + ruler2;

Cadena ruler4 = ruler3 + " 4 " + ruler3;

System.out.println(ruler1);

System.out.println(ruler2);

System.out.println(ruler3);

System.out.println(ruler4);

}

}

*Este programa imprime las longitudes relativas de las subdivisiones en una regla. La enésima línea de salida*

*es la longitud relativa de las marcas en una regla subdividida en intervalos de 1/2*

*n*

*de una pulgada. Para*

*Por ejemplo, la cuarta línea de salida proporciona las longitudes relativas de las marcas que indican intervalos*

*de un dieciseisavo de pulgada en una regla.*

*La función de regla para n = 4*

1

1

2 1 3 1 2 1 4 1 2 1 3 1 2

Nuestro uso más frecuente (con diferencia) de la operación de concatenación es juntar los resultados del cálculo para la salida con System.out.println(). Por ejemplo, podríamos simplificar UseArgument (programa 1.1.2) reemplazando sus tres sentencias en main() con esta única sentencia:

System.out.println("Hola, " + args[0] + ". ¿Cómo estás?");

Hemos considerado el tipo String en primer lugar precisamente porque lo necesitamos para la salida (y los argumentos de la línea de comandos) en programas que procesan no solo cadenas, sino también otros tipos de datos. A continuación, consideramos dos mecanismos convenientes en Java para convertir números en cadenas y cadenas en números.

*Conversión de números en cadenas para la salida.* Como se mencionó al principio de esta sección, el tipo String integrado de Java obedece a reglas especiales. Una de estas reglas especiales es que se puede convertir fácilmente un valor de cualquier tipo en un valor String: siempre que usemos el operador + con un String como uno de sus operandos, Java convierte automáticamente el otro operando en un String, produciendo como resultado la String formada a partir de los caracteres del primer operando seguidos de los caracteres del segundo operando.

Por ejemplo, el resultado de estos dos fragmentos de código

Cadena a = "1234"; Cadena a = "1234";

Cadena b = "99"; u b = 99;

Cadena c = a + b; Cadena c = a + b; son ambos iguales: asignan a C el valor "123499". Usamos esta conversión automática libremente para formar valores de cadena para usar con System.out.print() y System.out.println(). Por ejemplo, podemos escribir enunciados como este:

System.out.println(a + " + " + b + " = " + c);

Si a, b y c son variables int con los valores 1234, 99 y 1333, respectivamente, esta instrucción imprime la cadena 1234 + 99 = 1333.

*Conversión de cadenas en números para la entrada.* Java también proporciona métodos de biblioteca que convierten las cadenas que escribimos como argumentos de línea de comandos en valores numéricos para tipos primitivos. Utilizamos los métodos de la biblioteca Java Integer.parseInt() y Double.parseDouble() para este propósito. Por ejemplo, escribir Integer.parseInt("123") en el texto del programa equivale a escribir el literal int 123. Si el usuario escribe 123 como primer argumento de la línea de comandos, el código Integer.parseInt(args[0]) convierte el valor de cadena "123" en el valor int 123. Verá varios ejemplos de este uso en los programas de esta sección.

Con estos mecanismos, nuestra vición de cada programa Java como una caja negra que toma argumentos de cadena y produce resultados de cadena sigue siendo válida, pero ahora podemos interpretar esas cadenas como números y usarlas como base para cálculos significativos.

**Enteros** El Int type representa números enteros (números naturales) entre

–2147483648 (2  31) y 2147483647 (2 311). Estos límites se derivan del hecho de que los números enteros se representan en binario con 32 dígitos binarios; Hay 232 posibles

valores. (El término *El dígito binario es*  omnipresente en las ciencias de la computación, y casi siempre usamos la abreviatura bit : un bit es 0 o 1). El rango de int posible

values es asimétrico porque el cero se incluye con los valores positivos. Puede ver las preguntas y respuestas al final de esta sección para obtener más detalles sobre la representación de números,

Pero en el contexto presente basta con saber que un int es uno de los conjuntos finitos de valores en el rango que acabamos de dar. Puede especificar un literal int con una secuencia de dígitos decimales del 0 al 9 (que, cuando se interpretan como números decimales, se encuentran dentro del intervalo definido). Usamos ints con frecuencia porque surgen naturalmente cuando estamos implementando programas.

Los operadores aritméticos estándar para la suma/resta (+ y -), la multiplicación (\*), la división (/) y el resto (%) para el tipo de datos int están integrados en Java. Estos operadores toman dos operandos int y producen un resultado int, con una excepción significativa: no se permite la división o el resto por cero. Estas operaciones se definen como en la escuela primaria (teniendo en cuenta que todos los resultados deben ser enteros): dados dos valores enteros a y b, el valor de a / b es el número de veces que b entra en a con la parte fraccionaria descartada, y el valor de a % b *es*  el resto

|  |  |  |
| --- | --- | --- |
| *expresión* | *valor* | *comentario* |
| 99 | 99 | *Entero literal* |
| +99 | 99 | *signo positivo* |
| -99 | -99 | *signo negativo* |
| 5 + 3 | 8 | *adición* |
| 5 - 3 | 2 | *sustracción* |
| 5 \* 3 | 15 | *multiplicación* |
| 5 / 3 | 1 | *Sin parte fraccionaria* |
| 5 % 3 | 2 | *resto* |
| 1 / 0 |  | *Error en tiempo de ejecución* |
| 3 \* 5 - 2 | 13 | *\* tiene precedencia* |
| 3 + 5 / 2 | 5 | */ tiene precedencia* |
| 3 - 5 - 2 | -4 | *Asociativo de izquierda* |
| ( 3 - 5 ) - 2 -4 | | *Mejor estilo* |
| 3 - ( 5 - 2 ) 0 | | *inequívoco* |

*Expresiones int típicas*

que se obtiene cuando se divide a por b. Por ejemplo, el valor de 17 / 3 es 5, y el

valor de 17 % 3 es 2. Los resultados int que obtenemos de las operaciones aritméticas son justo lo que esperamos, excepto que si el resultado es demasiado grande para caber en la representación de 32 bits de int, entonces se truncará de una manera bien definida. Esta situación es conocida

*valoresenteros entre*  2 31 y 2 311

*literales típicos1234 99 0 1000000*

*operationssign sumar restar multiplicar dividir operadores de resto+ - + - \* / %*

*Tipo de datos int integrado en Java*

%

**Java IntOps.java**

%

**java IntOps 1234 99**

\* 99 = 122166

1234

1234

/ 99 = 12

% 99 = 46

1234

46

12 \* 99 +

1234 =

***Programa 1.2.2***

***Yo***

***Multiplicación y división de Nteger***

IntOps de clase pública

{

public static void main(String[] args)

{

int a = Integer.parseInt(args[0]);

int b = Integer.parseInt(args[1]);

int p = a \* b;

int q = a / b;

int r = a % b;

System.out.println(a + " \* " + b + " = " + p);

System.out.println(a + " / " + b + " = " + q);

System.out.println(a + " % " + b + " = " + r);

System.out.println(a + " = " + q + " \* " + b + " + " + " + r);

}

}

*La aritmética para números enteros está integrada en Java. La mayor parte de este código está dedicado a la tarea de obtener el*

*valores de entrada y salida; La aritmética real está en las declaraciones simples en el medio del programa*

*que asignan valores a*

*p*

*,*

*q*

*y*

*r*

*.*

como *desbordamiento*. En general, tenemos que tener cuidado de que tal resultado no sea malinterpretado por nuestro código. Por el momento, calcularemos con números pequeños, por lo que no tiene que preocuparse por estas condiciones de contorno.

El programa 1.2.2 ilustra tres operaciones básicas (multiplicación,

sion, y el resto) para manipular números enteros,. También muestra el uso de Integer.parseInt() para convertir valores de cadena en la línea de comandos en valores int, así como el uso de la conversión automática de tipos para convertir valores int en valores de cadena para la salida.

Otros tres tipos incorporados son representaciones diferentes de enteros en Java.

Los tipos long, short y byte son los mismos que int, excepto que usan 64, 16 y 8 bits respectivamente, por lo que el rango de valores permitidos es diferente. Los programadores usan long cuando trabajan con números enteros enormes, y los otros tipos para ahorrar espacio. Puedes encontrar una tabla con los valores máximos y mínimos para cada tipo en el sitio del libro, o puedes averiguarlos por ti mismo a partir del número de bits.

|  |  |  |
| --- | --- | --- |
| números de puntos en cualquier representación digital por ordenador. Los números de coma flotante se aproximan suficientemente a los números reales  bueno, que podemos usarlos en aplicaciones, pero a menudo tenemos que lidiar con el hecho de que  No siempre podemos hacer cálculos exactos.  Puede especificar un literal doble con una secuencia de dígitos con un separador decimal. Por ejemplo, el literal 3.14159 representa una aproximación de seis dígitos a . Alterna- | *expresión*  3.141 + 2.0  3.141 - 2.0  3.141 / 2.0  5.0 / 3.0  10.0 % 3.141  1.0 / 0.0  Matemáticas.sqrt(2.0)  Math.sqrt(-1.0) | *valor*  5.141  1.111  1.5705  1.6666666666666667  0.577  Infinidad  1.4142135623730951  En |
| *Expresiones dobles típicas* | |

**Números de coma flotante El**  tipo doble representa números de coma flotante , para su uso en aplicaciones científicas y comerciales. La representación interna es como la notación científica, de modo que podemos calcular con números en un rango enorme. Usamos números de coma flotante para representar números reales, ¡pero decididamente no son lo mismo que los números reales! Hay infinitos números reales, pero solo podemos representar un número finito de números flotantes.

De hecho, se especifica un literal doble con una notación como la notación científica: el literal

6.022E23 representa el número 6.022  1023. Al igual que con los enteros, puede usar estas convenciones para escribir literales de punto flotante en los programas o para proporcionar números de punto flotante como argumentos de cadena en la línea de comandos.

Los operadores aritméticos +, -, \* y / se definen para double. Más allá de estos operadores incorporados, la biblioteca Java Math define la función de raíz cuadrada, las funciones trigonométricas, las funciones logarítmicas/exponenciales y otras funciones comunes para números de coma flotante. Para utilizar una de estas funciones en una expresión, escriba el nombre de la función seguido de su argumento entre paréntesis. Por ejemplo,

*valoresnúmeros reales (especificados por el estándar IEEE 754)*

*Literales típicos3.14159 6.022E23 2.0 1.4142135623730951 OperacionesSumar, Restar, Multiplicar y Dividir Operadores+ - \* /*

*Tipo de datos doble integrado en Java*

%

**javac Quadratic.java**

%

**java Cuadrático -3.0 2.0**

2.0

1.0

%

**java Cuadrático -1.0 -1.0**

1.618033988749895

-0.6180339887498949

%

**java Quadratic 1.0 1.0**

En

En

***Programa 1.2.3***

***Quadr***

***Fórmula atic***

Clase pública Cuadrática

{

public static void main(String[] args)

{

double b = Double.parseDouble(args[0]);

double c = Double.parseDouble(args[1]);

doble discriminante = b \* b - 4.0 \* c;

double d = Math.sqrt(discriminante);

System.out.println((-b + d) / 2.0);

System.out.println((-b - d) / 2.0);

}

}

*Este programa imprime las raíces del polinomio x*

*2*

*+ bx + c*

,

*usando la fórmula cuadrática. Para*

*Ejemplo, las raíces de x*

*2*

*– 3x + 2 son 1 y 2, ya que podemos factorizar la ecuación como (x – 1)(x – 2);*

*Las raíces de X*

*2*

*– x – 1 son*



*y 1 –*



*Dónde*



*es la proporción áurea; y las raíces de x*

*2*

*+ x + 1*

*no son números reales.*

ample, el código Math.sqrt(2.0) se evalúa como un valor doble que es aproximadamente la raíz cuadrada de 2. Discutimos el mecanismo detrás de esta disposición con más detalle en la sección 2.1 y más detalles sobre la biblioteca Math al final de esta sección.

Al trabajar con números de coma flotante, una de las primeras cosas con las que se encontrará es el problema de la *precisión*. Por ejemplo, la impresión 5.0/2.0 da como resultado

2.5 como se esperaba, pero la impresión 5.0/3.0 da como resultado 1.666666666666666666666666. En la sección 1.5, aprenderá el mecanismo de Java para controlar el número de dígitos significativos que ve en la salida. Hasta entonces, trabajaremos con el formato de salida predeterminado de Java.

El resultado de un cálculo puede ser uno de los valores especiales Infinito (si el número es demasiado grande para ser representado) o NaN (si el resultado del cálculo no está definido). Aunque hay una gran cantidad de detalles a considerar cuando los cálculos involucran estos valores, puede usar double de una manera natural y comenzar a escribir programas Java en lugar de usar una calculadora para todo tipo de cálculos. Por ejemplo, programa

La figura 1.2.3 muestra el uso de valores dobles en el cálculo de las raíces de una ecuación cuadrática utilizando la fórmula cuadrática. Varios de los ejercicios al final de esta sección ilustran aún más este punto.

Al igual que con long, short y byte para enteros, hay otra representación para números reales llamada float. Los programadores a veces usan float para ahorrar espacio cuando la precisión es una consideración secundaria. El tipo doble es útil para unos 15 dígitos significativos; el tipo flotante es bueno sólo para unos 7 dígitos. No usamos float en este libro.

|  |  |
| --- | --- |
| El tipo booleano representa el valor de la verdad. |  |
| De esta manera, la mayoría de las personas que se encuentran en el mundo de Tiene solo dos valores: verdadero y falso. Estos son también los dos literales booleanos posibles. Cada variable booleana tiene uno de estos dos valores, y cada operación booleana tiene operandos y un resultado que toma solo uno de estos dos valores . Esta simplicidad es engañosa: los valores booleanos se encuentran en la base de la informática. | *ValoresVerdadero o falso*  *literalstrue false*  *operacionesy o no operadores*&& || !  *Tipo de datos booleano integrado en Java* |

## Booleanos

Las operaciones más importantes definidas para los booleanos son *y* (&&), *o* (||), y *no* (!), que tienen definiciones conocidas:

* A && B es verdadero si ambos operandos son verdaderos, y falso si cualquiera de ellos es falso. • A || B es falso si ambos operandos son falsos, y verdadero si cualquiera de ellos es verdadero.
* ! a es verdadero si a es falso, y falso si a es verdadero.

A pesar de la naturaleza intuitiva de estas definiciones, vale la pena especificar completamente cada posibilidad para cada operación en tablas conocidas como tablas de *verdad*. La función *not* tiene un solo operando: su valor para cada uno de los dos valores posibles del operando es

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | *un* | *!un* | | verdadero | falso | | falso | verdadero | | |  |  |  |  | | --- | --- | --- | --- | | *un* | *b* | *A & B* | *A || b* | | falso | falso | falso | falso | | falso | verdadero | falso | verdadero | | verdadero | falso | falso | verdadero | | verdadero | verdadero | verdadero | verdadero | |

*Definiciones de tabla de verdad de operaciones booleanas*

especificado en la segunda columna. Cada  *una de las funciones and* y *or* tiene dos operandos: hay cuatro posibilidades diferentes para los valores de operando, y los valores de las funciones para cada posibilidad se especifican en las dos columnas de la derecha.

Podemos usar estos operadores con paréntesis para desarrollar expresiones arbitrariamente complejas, cada una de las cuales especifica una función booleana bien definida. A menudo, la misma función aparece en diferentes formas. Por ejemplo, las expresiones (a & & b) y !(! A || !b) son equivalentes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| un | b | A & B | !un | !b | !a || !b ! (!a || !b) |
| falso | falso | falso | verdadero | verdadero | verdadero falso |
| falso | verdadero | falso | verdadero | falso | verdadero falso |
| verdadero | falso | falso | falso | verdadero | verdadero falso |
| verdadero | verdadero | verdadero | falso | falso | falso verdadero |

*Prueba de la tabla de verdad de que a & & b y !(! A || !b) son idénticos*

El estudio de la manipulación de expresiones de este tipo se conoce como *lógica booleana*. Este campo de las matemáticas es fundamental para la computación: juega un papel esencial en el diseño y funcionamiento del propio hardware informático, y también es un punto de partida para los fundamentos teóricos de la computación. En el contexto actual, estamos interesados en las expresiones booleanas porque las usamos para controlar el comportamiento de nuestros programas. Normalmente, una condición de interés determinada se especifica como una expresión booleana y se escribe un fragmento de código de programa para ejecutar un conjunto de instrucciones si esa expresión es verdadera y un conjunto diferente de instrucciones si la expresión es falsa. La mecánica de hacerlo es el tema de la sección 1.3.

**Comparaciones** Algunos operadores *de tipo mixto* toman operandos de un tipo y producen un resultado de otro tipo. Los operadores más importantes de este tipo son los operadores de comparación ==, !=, <, <=, > y >= , que se definen para cada tipo numérico primitivo y producen un resultado booleano. Dado que las operaciones se definen solo con respecto a los tipos de datos, cada uno de estos símbolos representa muchas operaciones, una para cada tipo de datos. Se requiere que ambos operandos sean del mismo tipo.

*¿Discriminante no negativo?* (b\*b - 4.0\*a\*c) >= 0.0

*¿Principios de siglo?* (año % 100) == 0 *¿Mes legal?* (mes >= 1) && (mes <= 12)

*Expresiones de comparación típicas*

***Programa 1.2.4***

***L***

***Año EAP***

Clase pública LeapYear

{

public static void main(String[] args)

{

int año = Integer.parseInt(args[0]);

booleano esAño bisiesto;

isLeapYear = (año % 4 == 0);

isLeapYear = isLeapYear && (año % 100 != 0);

isLeapYear = isLeapYear || (año % 400 == 0);

System.out.println(isLeapYear);

}

}

*Este programa comprueba si un número entero corresponde a un año bisiesto en el calendario gregoriano. Un*

*año bisiesto si es divisible por 4 (2004), a menos que sea divisible por 100, en cuyo caso no lo es*

*(1900)*

*, a menos que sea divisible por 400, en cuyo caso es*

*(2000).*

%

**javac LeapYear.java**

%

**Año bisiesto de Java 2004**

verdadero

%

**Año bisiesto de Java 1900**

falso

%

**Java Año bisiesto 2000**

verdadero

Incluso sin entrar en los detalles de la representación numérica, está claro que las operaciones para los diversos tipos son bastante diferentes. Por ejemplo, una cosa es comparar dos enteros para comprobar que (2 <= 2) es verdadero, pero otra muy distinta es comparar dos dobles para comprobar si (2,0 <= 0,002e3) es verdadero. Aún así, estas operaciones están bien definidas y son útiles para escribir código que pruebe condiciones como (b \* b - 4.0 \* a \* c) > = 0.0, que se necesita con frecuencia, como verá.

|  |  |
| --- | --- |
| Las operaciones de comparación, junto con la lógica booleana, proporcionan la base para la toma de decisiones en los programas Java. El programa 1.2.4 es un ejemplo de su uso, y puedes encontrar otros ejemplos en los ejercicios al final de esta sección. Más importante aún, en la sección 1.3 veremos el papel que juegan las expresiones booleanas en programas más sofisticados. | *operador significado verdadero falso*  == *igual* 2 == 2 2 == 3  != *no son iguales* 3 != 2 2 != 2  < *menos que* 2 < 13 2 < 2  <= *menor o igual que* 2 <= 2 3 <= 2  > *mayor que* 13 > 2 2 > 13  >= *mayor o igual que* 3 >= 2 2 >= 3  *Comparaciones con operandos int y un resultado booleano* |

Las operaciones de comparación tienen una prioridad más baja que los operadores aritméticos y una prioridad más alta que los operadores booleanos, por lo que no necesita los paréntesis en una expresión como (b \* b - 4.0 \* a \* c) > = 0.0, y podría escribir una expresión como mes > = 1 y mes < = 12 sin paréntesis para probar si el valor de la variable int mes está entre 1 y 12. (Sin embargo, es mejor usar los paréntesis).

**Métodos de biblioteca y API** Como hemos visto, muchas tareas de programación implican el uso de métodos de biblioteca Java además de los operadores incorporados. El número de métodos de biblioteca disponibles es enorme. A medida que aprendas a programar, aprenderás a usar más y más métodos de biblioteca, pero es mejor al principio restringir tu atención a un conjunto relativamente pequeño de métodos. En este capítulo, ya ha utilizado algunos de los métodos de Java para imprimir, para convertir datos de un tipo a otro y para calcular funciones matemáticas (la biblioteca Java Math). En capítulos posteriores, aprenderá no solo cómo usar otros métodos, sino también cómo crear y usar sus propios métodos.

Para mayor comodidad, resumiremos constantemente los métodos de biblioteca que necesita saber cómo usar en tablas como esta:

void System.out.print(String s) *Impresión s*

void System.out.println(Cadena s) *Impresión s, seguido de una nueva línea* void System.out.println() *Imprimir una nueva línea*

*Nota: Se puede usar cualquier tipo de datos como argumento (y se convertirá automáticamente en String).*

*Métodos de la biblioteca Java para imprimir cadenas en el terminal*

Dicha tabla se conoce como una aplicación *de programación inter-*

*Nombre de la biblioteca*

*cara* (*API* ). Cada método se describe mediante una línea en la API

Clase pública Matemáticas que especifica la información que necesita saber para usar el método

. . .

*Método de firma Nombre* Método. El código de las tablas *no es el*  código que se escribe para usar el método; se conoce como la firma del método.

Doble SQRT (Doble A)

La firma especifica el tipo de argumentos, los

|  |  |
| --- | --- |
| *Tipo de valor devuelto Tipo de argumento*  . . .  *Anatomía de la firma de un método* | od y el tipo de resultado que calcula el método (el *valor devuelto*).  En el código, puede llamar a un método escribiendo su nombre |

seguido de argumentos, encerrados entre paréntesis y separados por comas. Cuando Java ejecuta su programa,

Digamos que *llama (*o *evalúa*) el método con los argumentos dados y que el método *devuelve* un valor. Una llamada al método es un

expresión, por lo que puede usar una llamada de método en *nombre de la biblioteca nombre del método de* la misma manera que usa variables y liter-

als para construir expresiones más complicadas. double d = Math.sqrt(b\*b - 4.0\*a\*c);

|  |  |
| --- | --- |
| Por ejemplo, puede escribir expresiones como Math.sin(x) \* Math.cos(y) y así sucesivamente. Un argumento también es una expresión, por lo que puede | *Tipo de valor devuelto argumento*  *Uso de un método de biblioteca* |

escriba código como Math.sqrt(b\*b - 4.0\*a\*c) y Java sabe lo que quiere decir: evalúa la expresión del argumento y pasa el valor resultante al método.

Las tablas de la API en la página opuesta muestran algunos de los métodos más utilizados en la biblioteca matemática de Java , junto con los métodos Java que hemos visto para imprimir texto en la ventana del terminal y para convertir cadenas en tipos primitivos. En la tabla siguiente se muestran varios ejemplos de llamadas que utilizan estos métodos de biblioteca:

|  |  |  |  |
| --- | --- | --- | --- |
| *Llamada al método* | *biblioteca* | *Tipo de valor devuelto* | *valor* |
| Integer.parseInt ("123") | Entero | Int | 123 |
| Double.parseDouble("1.5") | Doble | doble | 1.5 |
| Math.sqrt (5.0 \* 5.0 - 4.0 \* 4.0) | Matemática | doble | 3.0 |
| Math.log(Math.E) | Matemática | doble | 1.0 |
| Math.random() | Matemática | doble | *aleatorio en* [0, 1] |
| Matemáticas.redondo(3.14159) | Matemática | largo | 3 |
| Matemáticas.max(1.0, 9.0) | Matemática | doble | 9.0 |

*Llamadas típicas a métodos de biblioteca Java*

Clase pública Matemáticas

Doble ABS (Doble A) *valor absoluto de un* Doble Max (Doble A, Doble B) *máximo de un y b* Doble Min (Doble A, Doble B) *mínimo de un y b*

*Nota 1: abs(), max() y min() también se definen para int, long y float.*

|  |  |
| --- | --- |
| Doble pecado (doble theta) | *Seno de theta* |
| Doble COS (doble theta) | *Coseno de Theta* |
| Doble bronceado (doble theta) | *tangente o theta* |

*Nota 2: Los ángulos se expresan en radianes. Utilice toDegrees() y toRadians() para convertir. Nota 3: Utilice asin(), acos() y atan() para las funciones inversas.*

Doble exp(Doble A) exponencial (E A) Doble log(Doble A) *Logaritmo natural (Loge A, o Ln A) Doble PoW(Doble A, Doble B) Eleva*  A a la B-ésima potencia (AB*) Ronda larga (Doble A) Redondea A al número entero más cercano Doble aleatorio()* en *[0, 1*  *)*  doble sqrt(doble a) *raíz cuadrada de a*

doble E *Valor de E (constante)* doble PI *valor de*  *(constante)*

*Consulte booksite para conocer otras funciones disponibles.*

*Extractos de la biblioteca matemática de Java*

|  |  |
| --- | --- |
| void System.out.print(String s) | *Imprimir S* |
| void System.out.println(Cadena s) | *print s, seguido de una nueva línea* |
| void System.out.println() | *Imprimir una nueva línea* |

*Métodos de la biblioteca Java para imprimir cadenas en el terminal*

int Integer.parseInt(Cadena s) *convertir s a un Int valor*

double Double.parseDouble(String s) *convertir s a un doble valor*

long Long.parseLong(Cadena s) *convertir s a un largo valor*

*Métodos de la biblioteca Java para convertir cadenas en tipos primitivos*

Con tres excepciones, los métodos de la página anterior son *puros*: dados los mismos argumentos, siempre devuelven el mismo valor, sin producir ningún *efecto secundario* observable. El método Math.random() es impuro porque devuelve potencialmente un valor diferente cada vez que se le llama; los métodos System.out.print() y System.out.println() son impuros porque producen efectos secundarios: imprimir cadenas en el terminal. En las API, usamos una frase verbal para describir el comportamiento de un método que produce efectos secundarios; de lo contrario, usamos un sintagma nominal para describir el valor devuelto. La palabra clave void designa un método que no devuelve un

valor (y cuyo propósito principal es producir efectos secundarios).

La biblioteca Math también define los valores constantes Math.PI (para ) y Math.E (para e), que puede utilizar en sus programas*.* Por ejemplo, el valor de

Math.sin(Math.PI/2) es 1.0 y el valor de Math.log(Math.E) es 1.0 (porque Math.sin () toma su argumento en radianes y Math.log() implementa la función logaritmo natural).

Estas API son típicas de la documentación en línea que es el estándar en la programación moderna. La extensa documentación en línea de las API de Java es utilizada rutinariamente por programadores profesionales, y está disponible para usted (si está interesado) directamente desde el sitio web de Java o a través de nuestro sitio de libros. No es necesario ir a la documentación en línea para entender el código de este libro o para escribir código similar, porque presentamos y explicamos en el texto todos los métodos de biblioteca que usamos en APIs como estas y los resumimos en los endpapers. Y lo que es más importante, en los capítulos 2 y 3 aprenderás en este libro cómo desarrollar tus propias APIs e implementar métodos para tu propio uso.

**Una de** las reglas principales de la programación moderna es que siempre debe tener en cuenta el tipo de datos que procesa el programa. Solo conociendo el tipo se puede saber con precisión qué conjunto de valores puede tener cada variable, qué literales se pueden usar y qué operaciones se pueden realizar. Por ejemplo, supongamos que desea calcular el promedio de los cuatro enteros 1, 2, 3 y 4. Naturalmente, me viene a la mente la expresión (1 + 2 + 3 + 4) / 4, pero produce el valor int 2 en lugar del valor doble 2.5 debido a las convenciones de conversión de tipos. El problema se deriva del hecho de que los operandos son valores int, pero es natural esperar un valor doble para el resultado, por lo que la conversión de int a doble es necesaria en algún momento. Hay varias formas de hacerlo en Java.

*Conversión de tipos implícita.* Puede utilizar un valor int siempre que se espere un valor double, ya que Java convierte automáticamente los enteros en dobles cuando corresponde. Por ejemplo, 11\*0.25 se evalúa como 2.75 porque 0.25 es un doble y ambos operandos deben ser del mismo tipo; por lo tanto, 11 se convierte en un doble y luego el resultado de dividir dos dobles es un doble. Como otro ejemplo, Math.sqrt(4) se evalúa como 2.0 porque 4 se convierte en un doble, como esperaba Math.sqrt(), que luego devuelve un valor doble. Este tipo de conversión se denomina *promoción automática* o *coerción*. La promoción automática es apropiada porque su intención es clara y se puede hacer sin pérdida de información. Por el contrario, una conversión que puede implicar la pérdida de información (por ejemplo, asignar un valor double a un valor int

variable) conduce a un error en tiempo de compilación.

|  |  |  |  |
| --- | --- | --- | --- |
| Para ello, se puede hacer explícito el uso de un dispositivo llamado *cast*. Para convertir una expresión de un tipo primitivo a otro, anteponga el nombre de tipo deseado entre paréntesis. Por ejemplo, la expresión (int) 2.71828 es una conversión de double a int que produce un int con valor 2. Los métodos de conversión definidos para los repartos arrojan información de una manera razonable (para obtener una lista completa, consulte el sitio web). Por ejemplo, al convertir un número de coma flotante en un entero, se descarta la parte fraccionaria redondeando hacia cero. RandomInt (Aleatorio)  (programa 1.2.5) es un ejemplo de que | *expresión*  (1 + 2 + 3 + 4) / 4.0  Matemáticas.sqrt(4)  "1234" + 99  11 \* 0.25  (int) 11 \* 0,25  11 \* (int) 0.25  (int) (11 \* 0.25)  (int) 2.71828  Matemáticas.redondo(2.71828)  (int) Matemáticas.redondo(2.71828)  Integer.parseInt ("1234") | *Tipo de expresión*  double double double String double double int int int int  long int int | *Valor de expresión*  2.5  2.0  "123499"  2.75  2.75  0  2  2  3  3  1234 |
| *Conversiones de tipos típicas* | |

*Reparto explícito.* Java tiene algunas convenciones de conversión de tipos integradas para tipos primitivos que puede aprovechar cuando es consciente de que puede perder información. Tienes que hacer tu inten-

utiliza un molde para un cálculo práctico.

La conversión tiene mayor prioridad que las operaciones aritméticas: cualquier conversión se aplica al valor que le sigue inmediatamente. Por ejemplo, si escribimos int value = (int) 11 \* 0.25, la conversión no es de ayuda: el literal 11 ya es un número entero, por lo que la conversión (int) no tiene ningún efecto. En este ejemplo, el compilador genera un posible mensaje de error de pérdida de precisión porque habría una pérdida

**% javac RandomInt.java**

**% java RandomInt 1000**

548

**% java RandomInt 1000**

141

**% java RandomInt 1000000**

135032

***Programa 1.2.5***

***C***

***De esta manera, se puede obtener un número entero aleatorio***

clase pública RandomInt

{

public static void main(String[] args)

{

int n = Integer.parseInt(args[0]);

double r = Math.random();

uniforme entre 0.0 y 1.0

valor int = (int) (r \* n);

uniforme entre 0 y n-1

System.out.println(valor);

}

}

*Este programa utiliza el método Java*

*Math.random()*

*para generar un número aleatorio*

*r*

*entre*

*0.0*

*(inclusive) y*

*1.0*

*(exclusivo); luego se multiplica*

*r*

*por el argumento de la línea de comandos*

*n*

*Llegar*

*un número aleatorio mayor o igual que*

*0*

*y menos de*

*n*

*; a continuación, utiliza una conversión para truncar el*

*result para ser un número entero*

*valor*

*entre*

*0*

*y*

*N-1*

*.*

de precisión en la conversión del valor resultante (2,75) en un int para la asignación a valor. El error es útil porque el cálculo previsto para este código es probable (int) (11 \* 0,25), que tiene el valor 2, no 2,75.

*Conversión de tipos explícitos.* Puede utilizar un método que tome un argumento de un tipo (el valor que se va a convertir) y genere un resultado de otro tipo. Ya hemos utilizado los métodos de biblioteca Integer.parseInt() y Double.parseDouble() para convertir los valores de String en valores int y double , respectivamente. Hay muchos otros métodos disponibles para la conversión, entre otros tipos. Por ejemplo, el método de biblioteca Math.round() toma un argumento doble y devuelve un resultado largo: el entero más cercano al argumento. Así, por ejemplo, Math.round(3.14159) y Math.round(2.71828) son de tipo long y tienen el mismo valor (3). Si desea convertir el resultado de Math.round() en un int, debe usar una conversión explícita.

 Los programadores principiantes tienden a encontrar que la conversión de tipos es una molestia, pero los programadores experimentados saben que prestar especial atención a los tipos de datos es la clave del éxito en la programación. También puede ser una clave para evitar el fracaso: en un famoso incidente en 1985, un cohete francés explotó en el aire debido a un problema de conversión de tipos. Si bien es posible que un error en su programa no cause una explosión, vale la pena tomarse el tiempo para comprender de qué se trata la conversión de tipos. Después de haber escrito solo unos pocos programas, verá que la comprensión de los tipos de datos lo ayudará no solo a componer código compacto, sino también a hacer explícitas sus intenciones y evitar errores sutiles  *de Explosión de cohetes Ariane 5* en sus programas.

**Resumen**  *Un tipo de datos es un conjunto de valores y un conjunto de operaciones sobre esos valores.* Java tiene ocho tipos de datos primitivos: booleano, char, byte, short, int, long , float y double. En el código Java, usamos operadores y expresiones como los de expresiones matemáticas familiares para invocar las operaciones asociadas con cada tipo. El tipo booleano se usa para calcular con los valores lógicos true y false; el tipo char es el conjunto de valores de caracteres que escribimos; y los otros seis tipos numéricos se usan para calcular con números. En este libro, la mayoría de las veces usamos bool-

ean, int y double; no usamos short ni float. Otro tipo de datos que usamos con frecuencia, String, no es primitivo, pero Java tiene algunas facilidades integradas para Strings que son como las de los tipos primitivos.

Al programar en Java, tenemos que tener en cuenta que cada operación se define solo en el contexto de su tipo de datos (por lo que es posible que necesitemos conversiones de tipos) y que todos los tipos solo pueden tener un número finito de valores (por lo que es posible que tengamos que vivir con resultados imprecisos).

El tipo booleano y sus operaciones—&&, ||, y !—son la base para la toma de decisiones lógicas en los programas Java, cuando se utilizan junto con los operadores de comparación de tipos mixtos ==, !=, <, > , <= y >=. Específicamente, usamos expresiones booleanas para controlar las construcciones condicionales (if) y de bucle (for y while) de Java, que estudiaremos en detalle en la siguiente sección.

Los tipos numéricos y las bibliotecas de Java nos dan la capacidad de usar Java como una calculadora matemática extensa. Escribimos expresiones aritméticas utilizando los operadores incorporados +, -, \*, / y % junto con métodos Java de la biblioteca Math.

Aunque los programas de esta sección son bastante rudimentarios para los estándares de lo que podremos hacer después de la siguiente sección, esta clase de programas es bastante útil por derecho propio. Utilizarás tipos primitivos y funciones matemáticas básicas ampliamente en la programación Java, por lo que el esfuerzo que dediques ahora a comprenderlos sin duda valdrá la pena.

***Preguntas y respuestas (cuerdas)***

**Pregunta.** ¿Cómo almacena Java las cadenas internamente?

**Un.** Las cadenas son secuencias de caracteres codificadas con Unicode, un estándar moderno para codificar texto. Unicode admite más de 100.000 caracteres diferentes, incluidos más de 100 idiomas diferentes, además de símbolos matemáticos y musicales.

**Pregunta.** ¿Puede usar < y > para comparar valores de cadena?

**Un.** No. Esos operadores solo se definen para valores de tipo primitivo.

**Pregunta.** ¿Qué tal == y != ?

**Un.** Sí, pero es posible que el resultado no sea el esperado, debido a los significados que estos operadores tienen para los tipos no primitivos. Por ejemplo, hay una distinción entre una cadena y su valor. La expresión "abc" == "ab" + x es falsa cuando x es una cadena con valor "c" porque los dos operandos se almacenan en lugares diferentes de la memoria (aunque tengan el mismo valor). Esta distinción es esencial, ya que

Lo sabrás cuando lo discutamos con más detalle en la sección 3.1.

**Pregunta.** ¿Cómo puedo comparar dos cadenas como palabras en un libro, índice o diccionario?

**Un.** Aplazamos la discusión del tipo de datos String y los métodos asociados hasta la sección 3.1, donde introducimos la programación orientada a objetos. Hasta entonces, la operación de concatenación de cadenas es suficiente.

**Pregunta.** ¿Cómo puedo especificar un literal de cadena que es demasiado largo para caber en una sola línea?

**Un.** No puedes. En su lugar, divida el literal de cadena en literales de cadena independientes y concátelos, como en el ejemplo siguiente:

String dna = "ATGCGCCCACAGCTGCGTCTAAACCGGACTCTG" +

"AAGTCCGGAAATTACACCTGTTAG";

## Preguntas y respuestas (números enteros)

**Pregunta.** ¿Cómo almacena Java los enteros internamente?

**Un.** La representación más simple es para pequeños enteros positivos, donde el *sistema numérico binario* se usa para representar cada número entero con una cantidad fija de memoria de computadora.

**Pregunta.** ¿Qué es el sistema numérico binario?

**Un.** En el *sistema numérico binario*, representamos un número entero como una secuencia de *bits*. Un bit es un solo dígito binario (base 2), ya sea 0 o 1, y es la base para representar la información en las computadoras. En este caso los bits son coeficientes de potencias de 2. Específicamente, la secuencia de bits *bnbn*–1...*b2b1b0* representa el entero

*BN2N +* BN–*12N–*1 + ...  + B222 + B121 + B020

Por ejemplo, 1100011 representa el número entero

99 = 1· 64 + 1· 32 + 0· 16 + 0· 8 + 0· 4 + 1· 2 +1· 1

El sistema numérico decimal *más familiar*  es el mismo, excepto que los dígitos están entre 0 y 9 y usamos potencias de 10. Convertir un número a binario es un problema computacional interesante que consideraremos en la siguiente sección. Java utiliza 32 bits para representar los valores int. Por ejemplo, el entero decimal 99 podría representarse con el 00000000000000000000000001100011 de 32 bits. **Pregunta.** ¿Qué hay de los números negativos?

**Un.** Los números negativos se manejan con una convención conocida como *complemento a dos*, que no necesitamos considerar en detalle. Esta es la razón por la que el rango de valores int en Java es de –2147483648 (–231) a 2147483647 (231 – 1). Una consecuencia sorprendente de esta representación es que los valores int pueden volverse negativos cuando se hacen grandes y se *desbordan* (superan 2147483647). Si no ha experimentado este fenómeno, consulte el ejercicio 1.2.10. Una estrategia segura es usar el tipo int cuando sepa que los valores enteros serán inferiores a diez dígitos y el tipo long cuando crea que los valores enteros podrían llegar a ser de diez dígitos o más.

**Pregunta.** Parece incorrecto que Java simplemente deje que ints se desborde y dé valores incorrectos.

¿No debería Java comprobar automáticamente si hay desbordamiento?

**Un.** Sí, este tema es polémico entre los programadores. La respuesta corta por ahora es que la falta de dicha comprobación es una de las razones por las que estos tipos se denominan tipos de datos primitivos. Un poco de conocimiento puede ser de gran ayuda para evitar tales problemas. Una vez más, está bien usar el tipo int para números pequeños, pero cuando los valores llegan a los miles de millones, no se puede.

**Pregunta.** ¿Cuál es el valor de Math.abs(-2147483648)?

**Un.** -2147483648. Este resultado extraño (pero cierto) es un ejemplo típico de los efectos del desbordamiento de enteros y la representación del complemento a dos.

**Pregunta.** ¿A qué se refieren las expresiones 1/0 y 1 % 0 en Java?

**Un.** Cada uno genera una excepción en tiempo de ejecución, para la división por cero.

**Pregunta.** ¿Cuál es el resultado de la división y el resto de números enteros negativos?

**Un.** El cociente a / b se redondea hacia 0; el resto a % b se define de tal manera que

(a / b) \* b + a % b es siempre igual a a. Por ejemplo, -14 / 3 y 14 / -3 son ambos

-4, pero -14% 3 es -2 y 14% -3 es 2. Algunos otros lenguajes (incluido Python) tienen diferentes convenciones cuando se dividen por enteros negativos. **Pregunta.** ¿Por qué el valor de 10 ^ 6 no es 1000000 sino 12?

**Un.** El operador ^ no es un operador de exponenciación, como debes haber estado pensando. En su lugar, es la *exclusiva bit a bit o el*  operador, que rara vez es lo que quieres. En su lugar, puede usar el literal 1e6. También puedes usar Math.pow(10, 6) pero hacerlo es un desperdicio si estás subiendo 10 a una potencia conocida.

## Preguntas y respuestas (números de punto flotante)

**Pregunta.** ¿Por qué el tipo de los números reales se llama doble?

**Un.** El punto decimal puede "flotar" a través de los dígitos que componen el número real. Por el contrario, con los números enteros, el punto decimal (implícito) se fija después del dígito menos significativo.

**Pregunta.** ¿Cómo almacena Java internamente los números de coma flotante?

**Un.** Java sigue el estándar IEEE 754, que es compatible con el hardware de la mayoría de los sistemas informáticos modernos. La norma especifica que un número de punto flotante se almacena mediante tres campos: signo, mantisa y exponente. Si está interesado, consulte el sitio del libro para obtener más detalles. El estándar IEEE 754 también especifica cómo se deben manejar los valores especiales de punto flotante: cero positivo, cero negativo, infinito positivo, infinito negativo y NaN (no un número). En concreto, la aritmética de punto flotante nunca conduce a una excepción en tiempo de ejecución. Por ejemplo, la expresión

-0.0/3.0 se evalúa como -0.0, la expresión 1.0/0.0 se evalúa como infinito positivo y Math.sqrt(-2.0) se evalúa como NaN.

**Pregunta.** Quince dígitos para números de coma flotante ciertamente me parecen suficientes. ¿Realmente tengo que preocuparme mucho por la precisión?

**Un.** Sí, porque estás acostumbrado a las matemáticas basadas en números reales con una precisión infinita, mientras que el ordenador siempre se ocupa de aproximaciones finitas. Por ejemplo, la expresión (0,1 + 0,1 == 0,2) se evalúa como verdadera, pero la expresión

(0.1 + 0.1 + 0.1 == 0.3) se evalúa como falso! Escollos como este no son nada inusuales en la computación científica. Los programadores novatos deben evitar comparar dos números de punto flotante para la igualdad.

**Pregunta.** ¿Cómo puedo inicializar una variable doble a NaN o infinito?

**Un.** Java tiene constantes incorporadas disponibles para este propósito: Double.NaN, Double.POSITIVE\_INFINITY y Double.NEGATIVE\_INFINITY.

**Pregunta.** ¿Hay funciones en la biblioteca matemática de Java para otras funciones trigonométricas, como cosecante, secante y cotangente?

**Un.** No, pero puedes usar Math.sin(), Math.cos() y Math.tan() para calcularlos. Elegir qué funciones incluir en una API es un equilibrio entre la conveniencia de tener todas las funciones que necesita y la molestia de tener que encontrar una de las pocas que necesita en una larga lista. Ninguna opción satisfará a todos los usuarios, y los diseñadores de Java tienen muchos usuarios a los que satisfacer. Tenga en cuenta que hay muchas redundancias incluso en las API que hemos enumerado. Por ejemplo, puede usar Math.sin(x)/Math.cos(x) en lugar de Math.tan(x).

**Pregunta.** Es molesto ver todos esos dígitos al imprimir un doble. ¿Podemos hacer que System.out.println() imprima solo dos o tres dígitos después del punto decimal?

**Un.** Ese tipo de tarea implica una mirada más cercana al método utilizado para convertir de double a String. La función de la biblioteca Java System.out.printf() es una forma de hacer el trabajo, y es similar al método básico de impresión en el lenguaje de programación C y muchos lenguajes modernos, como se discutió en la sección 1.5. Hasta entonces, viviremos con los dígitos de más (lo cual no es del todo malo, ya que hacerlo nos ayuda a acostumbrarnos a los diferentes tipos primitivos de números).

## Preguntas y respuestas (variables y expresiones)

**Pregunta.** ¿Qué sucede si olvido declarar una variable?

**Un.** El compilador se queja cuando se hace referencia a esa variable en una expresión. Por ejemplo, IntOpsBad es el mismo que el programa 1.2.2, excepto que la variable p no está declarada (para ser de tipo int).

% javac IntOpsBad.java

IntOpsBad.java:7: error: no se puede encontrar el símbolo

p=a\*b;

^ símbolo: variable p ubicación: clase IntOpsBad

IntOpsBad.java:10: error: no se puede encontrar el símbolo

System.out.println(a + " \* " + b + " = " + p);

^ símbolo: variable p ubicación: clase IntOpsBad 2 errores

El compilador dice que hay dos errores, pero en realidad solo hay uno: falta la declaración de p . Si olvida declarar una variable que usa con frecuencia, recibirá bastantes mensajes de error. Una buena estrategia es corregir el *primer* error y comprobar esa corrección antes de abordar los posteriores. **Pregunta.** ¿Qué sucede si olvido inicializar una variable?

**Un.** El compilador comprueba esta condición y le mostrará un mensaje de error de que es posible que la variable no se haya inicializado si intenta usar la variable en una expresión antes de inicializarla.

**Pregunta.** ¿Hay alguna diferencia entre los operadores = y ==?

**Un.** ¡Sí, son bastante diferentes! El primero es un operador de asignación que cambia el valor de una variable y el segundo es un operador de comparación que produce un resultado booleano. Su capacidad para entender esta respuesta es una prueba segura de si entendió el material de esta sección. Piensa en cómo podrías explicarle la diferencia a un amigo.

**Pregunta.** ¿Se puede comparar un doble con un int?

**Un.** No sin hacer una conversión de tipos, pero recuerde que Java generalmente realiza la conversión de tipos requerida automáticamente. Por ejemplo, si x es un int con el valor 3, entonces la expresión (x < 3.1) es verdadera: Java convierte x en double (porque 3.1 es un literal doble) antes de realizar la comparación.

**Pregunta.** ¿Será el enunciado a = b = c = 17; ¿Asignar el valor 17 a las tres variables enteras A, B y C?

**Un.** Sí. Funciona porque una instrucción de asignación en Java también es una expresión (que se evalúa en su lado derecho) y el operador de asignación es asociativo a la derecha. Por una cuestión de estilo, en este libro no usamos tales *asignaciones encadenadas* .

**Pregunta.** ¿La expresión (a < b < c) probará si los valores de tres variables enteras a, b y c están en orden estrictamente ascendente?

**Un.** No, no se compilará porque la expresión a < b produce un valor booleano, que luego se compararía con un valor int. Java no admite *comparaciones encadenadas*. En su lugar, debe escribir (a < b & & b < c). **Pregunta.** ¿Por qué escribimos (a y b) y no (a y b)?

**Un.** Java también tiene un operador & que puedes encontrar si sigues cursos avanzados de programación.

**Pregunta.** ¿Cuál es el valor de Math.round(6.022e23)?

**Un.** Debes adquirir el hábito de escribir un pequeño programa Java para responder a estas preguntas tú mismo (y tratar de entender por qué tu programa produce el resultado que produce).

**Pregunta.** He oído que se refieren a Java como un  *lenguaje de tipado estático*. ¿Qué significa esto?

**Un.** El tipo estático significa que el tipo de cada variable y expresión se conoce en tiempo de compilación. Java también verifica y aplica restricciones de tipo en tiempo de compilación; por ejemplo, el programa no se compilará si intenta almacenar un valor de tipo double en una variable de tipo int o llamar a Math.sqrt() con un argumento String.

## Ejercicios

**1.2.1** Supongamos que a y b son variables int. ¿Qué hace la siguiente secuencia de declaraciones?

int t = a; b = t; a = b;

**1.2.2** Escriba un programa que use Math.sin() y Math.cos() para verificar que el valor de cos2  + sin2  *es aproximadamente 1 para cualquier*  ingresado como argumento de línea de comandos.

mento. Solo tienes que imprimir el valor. ¿Por qué los valores no siempre son exactamente 1?

**1.2.3** Supongamos que a y b son variables booleanas. Mostrar que la expresión

(!( a && b) && (a || b)) || ((a & & b) || ! (a || b)) se evalúa como verdadero.

**1.2.4** Supongamos que a y b son variables int. Simplifique la siguiente expresión: (!( a < b) && ! (a > b)).

**1.2.5** El  *operador* exclusivo o ^ para los operandos booleanos se define como verdadero si son diferentes, falso si son iguales. Proporcione una tabla de verdad para esta función.

**1.2.6** ¿Por qué 10/3 da 3 y no 3.3333333333?

*Solución.* Dado que tanto 10 como 3 son literales enteros, Java no ve la necesidad de la conversión de tipos y utiliza la división de enteros. Debes escribir 10.0/3.0 si quieres que los números sean literales dobles. Si escribe 10/3.0 o 10.0/3, Java realiza una conversión implícita para obtener el mismo resultado.

**1.2.7** ¿Qué imprime cada uno de los siguientes elementos?

1. System.out.println(2 + "bc");
2. System.out.println(2 + 3 + "bc");
3. System.out.println((2+3) + "bc");
4. System.out.println("bc" + (2+3));
5. System.out.println("bc" + 2 + 3); Explique cada resultado.

**1.2.8** Explique cómo usar el programa 1.2.3 para encontrar la raíz cuadrada de un número.

**1.2.9** ¿Qué imprime cada uno de los siguientes elementos?

1. System.out.println('b');
2. System.out.println('b' + 'c');
3. System.out.println((char) ('a' + 4)); Explique cada resultado.

**1.2.10** Supongamos que una variable a se declara como int a = 2147483647 (o equivalentemente, Integer.MAX\_VALUE). ¿Qué imprime cada uno de los siguientes?

1. System.out.println(a);
2. System.out.println(a+1);
3. System.out.println(2-a);
4. System.out.println(-2-a);
5. System.out.println(2\*a);
6. System.out.println(4\*a); Explique cada resultado.

**1.2.11** Supongamos que una variable a se declara como doble a = 3,14159. ¿Qué imprime cada uno de los siguientes?

1. System.out.println(a);
2. System.out.println(a+1);
3. System.out.println(8/(int) a);
4. System.out.println(8/a);
5. System.out.println((int) (8/a)); Explique cada resultado.

**1.2.12** Describa lo que sucede si escribe sqrt en lugar de Math.sqrt en el programa 1.2.3.

**1.2.13** Evalúe la expresión (Math.sqrt(2) \* Math.sqrt(2) == 2).

**1.2.14** Escriba un programa que tome dos enteros positivos como argumentos de la línea de comandos e imprima true si cualquiera de ellos divide uniformemente al otro.

**1.2.15** Escriba un programa que tome tres enteros positivos como argumentos de la línea de comandos e imprima false si alguno de ellos es mayor o igual que la suma de los otros dos y verdadero en caso contrario. (*Nota* : Este cálculo comprueba si los tres números podrían ser las longitudes de los lados de algún triángulo).

**1.2.16** Un estudiante de física obtiene resultados inesperados al usar el código

fuerza doble = G \* masa1 \* masa2 / r \* r;

para calcular los valores de acuerdo con la fórmula *F = Gm1m2*  *r* 2. Explique el problema y corrija el código.

**1.2.17** Indíquese el valor de la variable a después de la ejecución de cada una de las siguientes secuencias de sentencias:

int a = 1; booleano a = verdadero; int a = 2; a = a + a; A = !a; A = a \* a; a = a + a; A = !a; A = a \* a; a = a + a; A = !a; A = a \* a;

**1.2.18** Escriba un programa que tome dos argumentos enteros de la línea de comandos x e y e imprima la distancia euclidiana desde el punto (x, y) hasta el origen (0, 0).

**1.2.19** Escriba un programa que tome dos argumentos enteros de la línea de comandos a y b e imprima un número entero aleatorio entre a y b, inclusive.

**1.2.20** Escribe un programa que imprima la suma de dos números enteros aleatorios entre 1 y 6 (como los que puedes obtener al tirar los dados).

**1.2.21** Escriba un programa que tome un argumento doble de la línea de comandos t e imprima el valor de sin(2t)  sin(3t ).

**1.2.22** Escriba un programa que tome tres argumentos dobles de la línea de comandos  *x0, v0 y*  t e imprima el valor de x0 ** v0t *−*  g *t 2*  *2, donde* g es la constante 9.80665. (*Nota*: Este valor es el desplazamiento en metros después  *de t* segundos cuando un objeto es lanzado directamente hacia arriba desde la posición inicial *x0* a una velocidad *v0* metros por segundo).

**1.2.23** Escriba un programa que tome dos argumentos enteros de la línea de comandos m y d e imprima true si el día d del mes m está entre 3/20 y 6/20, false en caso contrario.

## Ejercicios creativos

**1.2.24** *Interés compuesto continuo.* Escribe un programa que calcule e imprima la cantidad de dinero que tendrías después de *t* años si invirtieras *P* dólares a una tasa de interés anual *r* (compuesta continuamente). El valor deseado viene dado por la fórmula *Pe rt*.

**1.2.25** *Sensación térmica.* Dada la temperatura *T (en*  grados Fahrenheit) y la velocidad del viento *v (*en millas por hora), el Servicio Meteorológico Nacional define la temperatura efectiva (la sensación térmica) de la siguiente manera:

*w = 35,74 * 0,6215 T  *(0,4275* T  *35,75*) v *0,16*

Escriba un programa que tome dos argumentos dobles de la línea de comandos, temperatura y velocidad, e imprima la sensación térmica. Utilice Math.pow(a, b) para calcular *ab*. *Nota*: La fórmula no es válida si *T* es mayor que 50 en valor absoluto o si *v* es mayor que 120 o menor que 3 (puede suponer que los valores que obtiene están en ese rango).

**1.2.26** *Coordenadas polares.* Escribe un programa que convierta de coordenadas cartesianas a polares. Su programa debe aceptar dos argumentos dobles de línea de comandos x e y imprimir las coordenadas polares r *y* . Utilice el método Math.atan2(y, x) para calcular el valor de arcotangente de y/x que está en el rango de  a .

x

y

*r*

*Coordenadas polares*

**1.2.27** *Números aleatorios gaussianos.* Escriba un programa RandomGaussian que imprima un número aleatorio *r* extraído de la *distribución gaussiana*. Una forma de hacerlo es usar la fórmula de *Box-Müller* *r =* sin(2  *v*) (2 ln *u*)1/2

donde *u* y *v* son números reales entre 0 y 1 generados por el método Math.random().

**1.2.28** *Comprobación del pedido.* Escriba un programa que tome tres argumentos dobles de la línea de comandos x, y y z e imprima true si los valores son estrictamente ascendentes o descendentes ( x < y < z o x > y > z *), y* false en caso contrario.

**1.2.29** *Día de la semana.* Escriba un programa que tome una fecha como entrada e imprima el día de la semana en el que cae esa fecha. El programa debe aceptar tres argumentos de línea de comandos int: m (mes), d (día) e y (año). Para m, use 1 para enero, 2 para febrero, y así sucesivamente. Para la salida, imprima 0 para el domingo, 1 para el lunes, 2 para el martes, y así sucesivamente. Utilice las siguientes fórmulas para el calendario gregoriano:

*y0 = y*  (14  *m*) / 12 *x = y0*  *y0* / 4  *y0* / 100  *y0* / 400 *m0 = m*  12  ((14  *m*) / 12)  2 *d0 =* (*d*  *x*  (31  *m0*) / 12) *%* 7

*Ejemplo:* ¿En qué día de la semana cayó el 14 de Febrero 2000?

*y0 =* 2000  1 = 1999

*x = 1999  1999 / 4  1999 /*  100   1999 / 400 = 2483 m0 = 2  12  1  2 = 12

*d0 =* (14  2483  (31  12) / 12) % 7 = 2500 % 7 = 1 *Respuesta* : Lunes.

**1.2.30** *Números aleatorios uniformes.* Escriba un programa que imprima cinco números aleatorios uniformes entre 0 y 1, su valor promedio y sus valores mínimo y máximo. Utilice Math.random(), Math.min() y Math.max().

**1.2.31** *Proyección de Mercator.* La  *proyección de Mercator* es una proyección conforme (que conserva el ángulo) que asigna la latitud  y la longitud  a coordenadas rectangulares (*x*, *y*). Es ampliamente utilizado, por ejemplo, en cartas náuticas y en los mapas que se imprimen desde la web. La proyección se define mediante las ecuaciones *x*     0  e y ** 1 /2 ln ((1  sin )  (1  sin )), donde 0 es la longitud del punto en el centro del mapa. Escriba un programa que tome 0 y la latitud y longitud de un punto de la línea de comandos e imprima su proyección.

**1.2.32** *Conversión de color.* Se utilizan varios formatos diferentes para representar el color. Por ejemplo, el formato principal de las pantallas LCD, las cámaras digitales y las páginas web, conocido como formato *RGB,* especifica el nivel de rojo (R), verde (G) y azul (B) en una escala entera de 0 a 255. El formato principal para la publicación de libros y revistas, conocido como formato *CMYK*, especifica el nivel de cian (C), magenta

(M), amarillo (Y) y negro (K) en una escala real de 0.0 a 1.0. Escriba un programa RGBtoCMYK que convierta RGB a CMYK. Tome tres números enteros (r, g y b) de la línea de comandos e imprima los valores CMYK equivalentes. Si los valores RGB son todos 0, entonces los valores CMY son todos 0 y el valor K es 1; De lo contrario, utilice estas fórmulas:

*W*  Max ( *r*  / 255, g  */ 255, b /* 255 )

*C  (W  ( R / 255))  W m  (W  (* G / 255))  W Y  (W  ( *B /* 255) )  W K  ** 1  **  W

**1.2.33** *Gran círculo.* Escriba un programa GreatCircle que tome cuatro argumentos dobles de la línea de comandos (x1, y1, x2 e y2) (la latitud y longitud, en grados, de dos puntos de la tierra) e imprima la distancia del círculo máximo entre ellos. La distancia del círculo máximo (en millas náuticas) viene dada por la siguiente ecuación: *d =* 60 arccos(sin(x1) sin(x2) * cos(x1) cos(*x2) cos(*y1* ** y2))

Tenga en cuenta que esta ecuación usa grados, mientras que las funciones trigonométricas de Java usan radianes. Utilice Math.toRadians() y Math.toDegrees() para convertir entre los dos. Utilice su programa para calcular la distancia del círculo máximo entre París (48,87° N y 2,33° W) y San Francisco (37,8° N y 122,4° W).

**1.2.34** *De tres tipos.* Escriba un programa que tome tres argumentos enteros de la línea de comandos y los imprima en orden ascendente. Utilice Math.min() y Math.max().

**1.2.35** *Curvas de dragón.* Escribe un programa para imprimir las instrucciones para dibujar las curvas del dragón del orden 0 al 5. Las instrucciones son cadenas de caracteres F, L y R , donde F significa "dibujar línea mientras se mueve 1 unidad F hacia adelante", L significa "girar a la izquierda" y R significa "girar a la derecha". Un

La curva de dragón de orden *n* se forma cuando se dobla una tira FLF

de papel en medio *n* veces, luego desdoblar en ángulos rectos. La clave para resolver este problema es tener en cuenta que una curva de orden FLFLFRF

*n es una curva de orden n1 seguida de una L* seguida de una curva de  *orden*  n1 *atravesada en orden inverso, y luego*

para encontrar una descripción similar para la curva inversa.FLFLFRFLFLFRF

*Curvas de dragón de orden 0, 1, 2 y 3*