Curso de programación Java

Artículo publicado originalmente en la revista Sólo Programadores

Este artículo será el primero de una serie orientada a introducir al lector en el lenguaje de programación Java. Este lenguaje de programación creado por Sun Microsystems a mediados de la década de los 90 es el lenguaje de programación más popular en la actualidad. Así lo demuestra el índice TIOBE.

El índice TIOBE trata de reflejar el "estado de salud" de los distintos lenguajes de programación. El índice reparte 100 puntos entre todos los lenguajes de programación existentes; el número de puntos que se lleva cada lenguaje depende del número de ofertas de trabajo que para dicho lenguaje haya en portales como Monster y similares, del número de respuestas que varios buscadores de Internet (entre ellos Google y Yahoo) devuelven al realizar consultas relacionadas con el lenguaje, del número de libros editados y vendidos sobre dicho lenguaje... y otras métricas similares. Como se puede ver en la figura 1, Java lidera dicho índice; y lo lleva liderando desde principios del 2001, con la excepción de unos pocos meses en torno a 2005, momento en el que fue temporalmente sobrepasado por C. Otro de los méritos que puede atribuirse es el liderar el ranking de Sourceforge de número de proyectos desarrollados en un determinado lenguaje. En abril del 2005 superó al actual tercer lenguaje de programación, C, y en noviembre del 2005 superó al actual segundo, C++.

Ranking del índice TIOBE con los 11 lenguajes de programación más populares

Java, al igual que su rival más directo (.NET), es mucho más que un simple lenguaje de programación, como podría ser el caso de C o C++. Es una plataforma de desarrollo de aplicaciones completa que proporciona contenedores web y lenguajes de script para la creación de páginas web dinámicas; soporte para componentes transaccionales distribuidos; mapeadores objetos relacionales para encargarse de la persistencia de la aplicación; colas de mensajería para el procesamiento asíncrono de tareas; soporte para desarrollo de aplicaciones para tarjetas inteligentes y otros dispositivos empotrados; soporte para desarrollo de aplicaciones para terminales móviles; un toolkit gráfico con soporte para animaciones 2D y 3D; soporte para el desarrollo de aplicaciones que se ejecutarán empotradas en un navegador; y un largo etcétera.

La plataforma se divide en tres grandes bloques. Uno de ellos, **Java SE** (anteriormente conocido como J2SE) es lo más parecido a un lenguaje de programación estándar. Proporciona una sintaxis para un lenguaje de programación, un entorno de ejecución para las aplicaciones creadas en dicho lenguaje y un extenso conjunto de librerías en las cuales se pueden apoyar dichas aplicaciones. Java SE será el objetivo de esta serie de artículos y es el punto por el que, habitualmente, los desarrolladores que quieren aprender Java se acercan a la plataforma.

Java ME (anteriormente conocido como J2ME) es una edición de Java orientada a programación de dispositivos móviles y empotrados. En esta categoría encajan desde las tarjetas inteligentes (como las que se emplean en los DNI electrónicos, en las tarjetas prepago o en las SIM de los teléfonos móviles) hasta terminales móviles de última generación, pasando por los reproductores del

formato de video de alta definición BlueRay, o software de control de coches. Java ME proporciona un subconjunto de las librerías y de las características del lenguaje de Java SE. Este subconjunto puede variar dependiendo de para qué dispositivos estemos programando; si estamos programando para terminales móviles de última generación están disponibles prácticamente todas las características de Java SE. Sin embargo, si estamos programando para tarjetas inteligentes el conjunto de librerías está considerablemente reducido, y el lenguaje de programación soporta menos características; por ejemplo, no soporta tipos de datos reales ya que estos dispositivos sólo tienen hardware para realizar aritmética de enteros.

Java ME es muy popular en la actualidad en dispositivos móviles, aunque a menudo pase desapercibido. Para que el lector se haga una idea de la presencia de esta tecnología, simplemente mencionar que se han manufacturado 3.500.000.000 de tarjetas inteligentes basadas en la tecnología Java, y más de 1.400.000.000 de teléfonos móviles con soporte para aplicaciones Java ME. Con toda probabilidad, tu teléfono móvil es capaz de ejecutar aplicaciones Java ME. Y, por cierto, la inmensa mayoría de los juegos y aplicaciones para terminales móviles que ves anunciados en televisión y en revistas están desarrollados en Java ME.

Si Java ME puede considerarse como un subconjunto de Java SE, **Java EE** (anteriormente conocido como J2EE) puede considerarse como un superconjunto. En este caso, no se extiende ni se modifica de ningún modo el lenguaje de programación. Se añade un conjunto amplio de librerías y una serie de contenedores de aplicaciones (contenedores web y de EJB). Estos contenedores proporcionan servicios, de un modo automático y transparente para el programador, a las aplicaciones que contienen, servicios como transaccionalidad, persistencia, autenticación, autorización, etcétera. Nuevamente, muchas de las web que visitas diariamente están hechas en Java: la web del Real Madrid, el propio GMail, o las web de los ministerios del gobierno español son algunos ejemplos.

Con el lenguaje de programación Java es posible programar todos estos dispositivos

1 El objetivo de esta serie de artículos: Java SE

Una vez terminada la revisión a vista de pájaro de las distintas partes que integran la plataforma, es momento de explicar exactamente qué vamos a abordar en esta serie de artículos. Nuestro objetivo será Java SE: presentaremos el lenguaje de programación Java e introduciremos las partes más esenciales de su librería estándar.

Los artículos están escritos suponiendo que el lector conoce algún lenguaje de programación y, por tanto, domina los conceptos básicos de la programación estructurada: el concepto de variable, estructuras de control de flujo, el concepto de función o procedimiento, etcétera. Esta serie de artículos no es, por tanto, un sustituto de un curso básico de programación.

Aunque no será imprescindible para seguir los artículos, si será aconsejable poseer conocimientos básicos de orientación a objetos: el concepto de método, de objeto, de clase, de paso de mensajes... serán expuestos con mucha brevedad. Nos centraremos más en presentar el soporte que Java proporciona para la programación orientada a objetos, más que en presentar la programación orientada a objetos en sí misma.

1.1 Java, el lenguaje

Ya hemos introducido, a vista de pájaro, lo que podríamos denominar "plataforma Java". En esta sección presentaremos brevemente cuáles son las principales características del lenguaje de

programación Java.

Java es un lenguaje *sencillo* de aprender. Su sintaxis es la de C++ “simplificada”. Los creadores de Java partieron de la sintaxis de C++ y trataron de eliminar todo lo que resultase complicado o fuese fuente de errores en este lenguaje. La herencia múltiple, la aritmética de punteros, la gestión de memoria dinámica (que en Java se realiza de modo transparente para el programador gracias al recogedor de basura) son ejemplos de puntos complicados del lenguaje C++ que en Java se han eliminado o simplificado. Esta decisión de diseño, intencionada, tenía como objetivo atraer a desarrolladores con conocimientos de C++ al nuevo lenguaje de programación, objetivo que fue alcanzado.

Java es completamente *independiente de la máquina y el sistema operativo* en el que se ejecuta. El código generado por el compilador Java es independiente de la arquitectura: podría ejecutarse en un entorno UNIX, Mac o Windows; en un procesador Sparc, x86, PowerPC, ARM, ... tanto de 32 como de 64 bits. El motivo de esto es que el que realmente ejecuta el código generado por el compilador no es el procesador del ordenador directamente, sino que se ejecuta mediante una máquina virtual, que es la que genera código máquina adecuado para cada arquitectura.

Además, Java es *portable* a nivel de código fuente. El párrafo anterior, aborda la portabilidad de Java a nivel de binario: un mismo programa compilado puede ejecutarse en distintas arquitecturas. Pero además en el lenguaje Java no hay aspectos dependientes de la implementación, todas las implementaciones de Java siguen los mismos estándares en cuanto a tamaño y almacenamiento de los datos, y en cuanto al modelo de memoria que ven las aplicaciones. Esto no ocurre así en otros lenguajes como C o C++. En estos dos lenguajes, por ejemplo, un entero puede tener un tamaño de 16, 32 o más bits, siendo la única limitación que el entero (

int
sea mayor o igual que un short
y menor o igual que un long int
. Así mismo, dependiendo de la implementación los datos pueden almacenarse en formato *little endian* o en *big endian*. Java lo hace siempre en el mismo formato.

Java es un lenguaje de programación *robusto y seguro*. Estas características surgen en buena medida de ser un lenguaje que no es ejecutado directamente por la CPU, sino que es ejecutado por la máquina virtual. Esta máquina virtual puede controlar los permisos que se le otorgan a una determinada aplicación y garantizar que dichos permisos no son violados. Por ejemplo, Java proporciona soporte para la creación de Applets. Estos son programas diseñados para ser ejecutados de modo automático cuando un usuario visita una página web. Si el Applet no estuviese restringido en lo que puede hacer al ejecutarse, podría comprometer nuestra máquina. Sin embargo, Java garantiza que ningún Applet puede escribir o leer de nuestro disco o mandar información del usuario que accede a la página a través de la red (como, por ejemplo, la dirección de correo electrónico). En general no le permite realizar cualquier acción que pudiera dañar la máquina o violar la intimidad del que visita la página web.

Java es un lenguaje de *alto rendimiento*. Sí, de alto rendimiento. Actualmente la velocidad de ejecución del código Java es semejante a la de C++, hay ciertas pruebas estándares de comparación (*benchmarks*) en las que Java gana a C++ y viceversa. Esto es así gracias al uso de compiladores *just in time*, compiladores que traducen los bytecodes de Java en código máquina para una determinada CPU, código que no precisa de la máquina virtual para ser ejecutado, y guardan el resultado de dicha conversión, volviéndolo a llamar en caso de volverlo a necesitar, con lo que se evita la sobrecarga de trabajo asociada a la interpretación del bytecode. Además, estos compiladores realizan optimizaciones que se basan en información que sólo está disponible en tiempo de

ejecución y que, por tanto, son imposibles de realizar en tiempo de compilación.

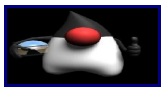
No obstante, por norma general el programa Java consume bastante más memoria que un programa C++ equivalente, ya que no sólo ha de cargar en memoria los recursos necesarios para la ejecución del programa, sino que además debe simular un sistema operativo y hardware virtuales (la máquina virtual). Sin embargo, dado el bajo precio de la memoria RAM en la actualidad y la gran cantidad de memoria que suele haber disponible, el mayor consumo de memoria a menudo es irrelevante.

Por último, en la línea del alto rendimiento, mencionar que Java posee soporte nativo para threads, lo que hace que escribir aplicaciones Java que tomen ventaja de procesadores de varios núcleos (lo más habitual en la actualidad es que al menos dispongamos de dos núcleos en una máquina) sea mucho más sencillo que en otros lenguajes de programación.

2 El kit de desarrollo Java, JDK

Para desarrollar aplicaciones Java es necesario un "*Java Development Kit*" o JDK. Existen múltiples JDKs, desarrollados por compañías diferentes, o desarrollados como proyectos de software libre. Yo recomiendo al lector que emplee el que distribuye de modo gratuito Sun Microsystems, que se puede descargar desde http://java.sun.com/j2se. La implementación de Java de Sun suele ser la más actualizada y, además, ¡es software libre!. No obstante, el lector podría emplear cualquier otro JDK. La portabilidad del lenguaje garantiza que todos funcionarán del mismo modo.

En la imagen se muestra al Duke, la mascota de Java. Y ¡él también se distribuye bajo una licencia libre!



Varios ejemplos de esta serie de artículos requerirán que se emplee un JDK 5 o superior. Java, como lenguaje, ha sufrido dos revisiones mayores en su historia. La primera fue la versión 1.2, en 1998. En la actualidad prácticamente nadie emplea JDKs anteriores al 1.2, por lo que no merece la pena hacer énfasis en las características del lenguaje antes de ese momento. La segunda, realizada en 2004, fue Java 5 (1.5 si se hubiese seguido con la numeración antigua, pero Sun decidió saltar de 1.4 a 5). Todavía existe bastante código escrito en Java 1.4.X y todavía existen muchos servidores de aplicaciones en producción que no soportan Java 5. Por ello cuando describamos una característica que sólo está presente en Java 5 avisaremos al lector. En cualquier caso, no existe ningún motivo para que el lector se descargue un JDK antiguo; lo más razonable es que se descargue la última versión (Java 6 en el momento de escribir este artículo).

Es de consenso que el entorno JDK no es el más adecuado para el desarrollo de aplicaciones Java, debido a funcionar única y exclusivamente mediante comandos de consola. Hoy en día la programación se suele ayudar de entornos visuales, como Eclipse y Netbeans, que facilitan enormemente la tarea. Mi preferencia personal hace que tienda a recomendar Netbeans, especialmente para un programador novato ya que este entorno de desarrollo requiere mucha menos configuración que Eclipse. Además, en artículos posteriores de esta serie se proporcionarán tutoriales flash cuyo objetivo es ayudar al usuario a comenzar a trabajar con Netbeans. En cualquier caso, ambos son excelentes entornos de desarrollo, completamente gratuitos y se distribuyen bajo licencia libre. Pueden descargarse desde http://netbeans.org y http://eclipse.org.

Aunque espero que para seguir esta serie de artículos el elector emplee alguno de estos entornos de desarrollo, siempre está bien comprender qué sucede cuando hacemos un clic sobre el botón de

"*Play*". En esta sección mostraremos cuáles son los componentes básicos del JDK y cómo emplearlos desde consola. Recomiendo al lector que para este primer artículo evite emplear un IDE y que trabaje con el JDK en la consola, para familiarizarse con él.

A partir de ahora supongo que el lector tiene un JDK instalado en su equipo. La descarga desde la página de Sun es un instalador estándar. Podemos aceptar todas las opciones por defecto. Una vez instalado, nuestro equipo podrá ejecutar aplicaciones Java. Sin embargo, por defecto, el kit de desarrollo no estará accesible desde la consola de comandos.

Para solucionar esto debemos incluir en la variable de entorno

PATH

el nombre del directorio de instalación del JDK. Para ello hacemos un clic derecho sobre el icono de "Mi PC" y seleccionamos "Propiedades". En la ventana que aparecerá vamos a la lengüeta de "Opciones avanzadas" y seleccionamos el botón "Variables de entorno". Si ya existe una variable llamada PATH

la seleccionamos y le damos al botón de "Modificar". Sino, creamos dicha variable mediante el botón "Nueva". El valor de dicha variable debe ser el directorio "bin" que se encuentra dentro del directorio de instalación del JDK. Si hemos instalado la última versión del JDK de Sun (la 6) en una máquina Windows y hemos aceptado todas las opciones por defecto este directorio será "C:\Archivos de programa\Java\jdk1.6.0\bin". Si en nuestro equipo ya estaba definida la variable PATH

vamos al final de la línea de su definición, añadimos un ";" y a continuación colocamos la ruta del directorio mencionado anteriormente.

2.1 Compilando nuestra aplicación: el comando javac

Ha llegado ya el momento (¡Por fin!) de comenzar a escribir algo de código fuente. Todavía no hemos explicado absolutamente nada de la sintaxis de Java; comenzaremos con ello en breve. Sin embargo, este primer programa, que mostramos en el listado 1, es muy simple de entender y no creo que lector tenga problemas comprendiendo lo que hace.

```
: T&iacute;pico programa de "hola mundo" desarrollado en Java. import java.util.*;
 /**Programa "hola mundo" que muestra la fecha actual.
   * @author Abraham Otero
   * @version 1.0
  */
  public class HolaMundo {
    /**El punto de entrada de la aplicaci&oacute;n
     * @param args array de cadenas de caracteres
     * @return no devuelve nada
    */
    public static void main(String[] args) {
     //Esta l&iacute;nea imprime por consola la cadena de caracteres &ldquo;Hola es:&rdquo;
      System.out.println("Hola es: ");
     //Esta sentencia imprime la fecha actual del equipo
      System.out.println(new Date());
    }
  }
```

La primera sentencia,

```
import java.util.*;
```

, importa el contenido de una librería, de la cual tomaremos un objeto que representa una fecha. Todas las sentencias comprendidas entre un `/**` y un `*/` son comentarios y, por tanto, serán ignorados por el compilador. Estos comentarios están en una sintaxis especial que es interpretada por una de las herramientas del JDK que veremos un poco más adelante. También es posible crear comentarios rodeando una o varias líneas con `/*` y `*/`; todo lo que se encuentre después de `/*` y hasta `*/` será un comentario. Por último, el tercer tipo de comentario que soporta Java es `//`. En este caso el comentario comienza en las dos barras y termina con el final de la línea.

La línea

```
public class HelloDate {
```

define una clase. Java es un lenguaje de programación orientado a objetos y en él todo, a excepción de los tipos de datos primitivos, son objetos. Por lo de ahora simplemente recomiendo al lector que ignore esta línea; las clases y los objetos serán abordados más en profundidad en el tercer artículo de la serie.

La línea

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

define el `main`, el punto de entrada, del programa. La primera sentencia que se encuentre dentro del bloque de código que comienza con la apertura de llave del `main` será la primera sentencia que se ejecute. En este caso la sentencia es `System.out.println("Hola es: ");`. En artículos posteriores explicaremos más en detalle el porqué de una sintaxis tan engorrosa como `System.out.println`. Por lo de la hora el lector lo único que necesita saber es que esa sentencia permite imprimir por la consola el texto que se le pase entre paréntesis, rodeado de comillas. La última sentencia ejecutable del `main` crea un objeto de tipo `Date`, esto es, crea un objeto de tipo fecha y lo muestra por consola.

Este es el momento en el que debemos coger un editor de texto cualquiera (el Bloc de notas de Windows es perfectamente válido) y teclear el código anterior. Deberemos guardarlo en un archivo con nombre

```
HolaMundo.java
```

. En Java una clase pública obligatoriamente tiene que guardarse en un archivo con el mismo nombre que la clase.

Para compilar nuestro primer programa necesitamos abrir una consola (en Windows, esto puede

hacerse desde el menú de inicio, seleccionando la entrada a "Ejecutar" y tecleando en la ventana que aparece

cmd

). Empleando el comando cd

, nos situaremos en el directorio donde hemos guardado el archivo HolaMundo.java. A continuación tecleamos:

```
javac HolaMundo.java
```

Javac

es el nombre del compilador de Java (su nombre viene de Java Compiler). Si hemos tecleado correctamente el código del ejemplo, el comando se ejecutará sin mostrar ningún mensaje por la consola y en el directorio aparecerá un archivo con nombre HolaMundo.class

; hemos compilado nuestro código con éxito. Si hay algún error en el código, el compilador nos advertirá de ello mostrando el correspondiente mensaje de error. Si nuestra consola no reconociese el comando javac

, o bien no hemos instalado todavía el JDK, o bien no hemos modificado adecuadamente la variable de entorno PATH.

2.2Lanzando la máquina virtual: el comando java

Los ficheros .class contienen **bytecode**, una especie de "ensamblador" de la plataforma Java. Este ensamblador puede ser interpretado por cualquier máquina virtual Java. La máquina virtual estará accesible desde nuestro

PATH

una vez hayamos instalado el JDK, sin necesidad de haber modificado ninguna variable de entorno. El comando para invocarla es java

.

Para indicarle a la máquina virtual que debe comenzar la ejecución de un determinado programa, el cual debe haber sido compilado con antelación, después del comando

java

debemos teclear el nombre de la clase que contiene el main

de nuestro programa. En nuestro caso el comando será:

```
java HolaMundo
```

No es necesario aquí suministrar la extensión del fichero, ya que siempre ha de ser un fichero .class. El resultado de la ejecución se muestra en la figura 4.

Resultado de compilar y ejecutar el código

HolaMundo

2.3 Generando documentación de nuestro código fuente

Una de las cosas que Java ha traído al mundo de la programación es la cultura de que todo código fuente debe tener una documentación asociada. Quizás haya sido por el hecho de que desde el día cero Java proporcionaba herramientas para generar documentación de modo semiautomático a partir de un código fuente con comentarios. O quizás fuese por el énfasis que los evangelistas del lenguaje hacían sobre este aspecto. Pero una de las cosas que todavía en la actualidad es muy envidiada por los desarrolladores de otros lenguajes es la documentación de nuestras librerías y la cultura de documentar el código fuente que existe dentro de la plataforma Java.

El JDK de Sun nos proporciona una herramienta que extrae los comentarios que comienzan por

/**

de un código fuente (los que comienzan por /*
y por //
son ignorados). En dichos comentarios puede emplearse una sintaxis especial que es interpretada por la herramienta. A partir de los comentarios, y con el soporte de dicha sintaxis si el programador la ha empleado, la herramienta genera una página HTML con documentación del código fuente.

El código que estamos usando de ejemplo posee varios de estos comentarios, y emplea la sintaxis especial. En los artículos que siguen a éste iremos presentando más detalles sobre dicha sintaxis. Aquí simplemente pretendemos mostrar cómo se emplea la herramienta, cuyo nombre es

javadoc
. Para generar documentación sobre nuestro código de ejemplo debemos teclear el comando:

```
javadoc HolaMundo.java
```

Este comando nos genera un montón de páginas HTML, no una sola. Habitualmente, suele emplearse para documentar programas complejos donde tiene sentido estructurar la documentación generada en distintas páginas web. La que nos interesa es la que tiene por nombre "index.html". Podemos ver su contenido en la figura 5. Como el lector podrá observar, los comentarios que introdujimos en el código fuente han sido extraídos por la herramienta e interpretados de modo adecuado para generar la documentación.

Documentación que genera el comando javadoc para nuestro ejemplo

2.4 El visor de Applets : el comando appletviewer

Todavía tardaremos varios números en explicar qué es un Applet. Pero, por completitud, presentaremos aquí la última herramienta del JDK que emplearemos en esta serie de artículos:

appletviewer
. Se trata de un comando que verifica el comportamiento de un Applet, es decir, de una aplicación Java diseñada para ejecutarse dentro de un navegador web. La entrada del comando ha de ser una página web que contenga una referencia al applet que deseamos probar. Su sintaxis es:

```
appletviewer mipagina.html
```

El comando ignora todo el contenido de la página web que no sean applets, y se limita a ejecutar éstos. Un ejemplo de página web "mínima" para poder probar un applet llamado

applet.class
sería el mostrado en el listado 2.

Página web con una etiqueta que empotra un Applet en el documento HTML.En el CD que acompaña esta revista lector podrá encontrar un archivo con nombre

Appplet.class
, junto con una página web con nombre Applet.html
. Puede copiar ambos al directorio de trabajo y probar con ellos el comando appletviewer
. También puede abrir la página web con su navegador web favorito y podrá comprobar como, si ya ha instalado el JDK, puede ver el Applet en el navegador.

3 Tipos de datos primitivos

En este apartado presentaremos los tipos de datos primitivos que existen en Java. Aquellos lectores que estén familiarizados con C, o C++, no encontrarán prácticamente nada nuevo en este apartado: los tipos son casi idénticos a los de estos lenguajes, salvo que aquí todos los enteros son signed siempre.

En Java toda variable declarada ha de tener su tipo, y además antes de poder emplearla hemos de inicializarla a un valor, si no el compilador se quejará y no generará los archivos .class. Esto no es necesario, por ejemplo, en C, siendo fuente de muchos errores el emplear en operaciones variables que nos hemos olvidado de inicializar. A continuación pasamos a describir los distintos tipos de datos primitivos que existen en Java.

3.1 Enteros

Almacenan, como su propio nombre indica, números enteros; esto es, números sin parte decimal.

Cabe destacar, como ya se indicó anteriormente, que por razones de portabilidad todos los datos en Java tienen el mismo tamaño y formato en todas las plataformas. En Java hay cuatro tipos de enteros, como se muestra en la tabla 1. Esta tabla también nos muestra el rango (valores mínimos y máximos) de cada tipo y el espacio que ocupan en memoria, medido en bytes.

TABLA 1: tipo de datos enteros en Java

Tipo	Tamaño (bytes)	Rango
byte	1	-128 a 127
short	2	-32768 a 32767
int	4	-2147483648 a 2147483647
long	8	-9223372036854775808 a 9223372036854775807

Para indicar que un literal entero es de tipo long debemos añadirle una

L

al final: el número 23

es un int

y el número 23L

es un long

.

3.2 Reales

Almacenan números reales, es decir, números con parte decimal. Como se muestra en la tabla 2, hay dos tipos diferentes de número real; se diferencian tanto en la precisión (el número de cifras decimales que son capaces de representar) como en el rango de valores que permiten representar.

TABLA 2: tipos de datos reales en Java

Tipo	Tamaño (bytes)	Rango
float	4	- 3.40282347E+38 a + 3.40282347E+38
double	8	- 17976931348623157OE+308 a + 17976931348623157OE+308

Si queremos indicar que un literal real es de tipo

float

debemos añadir una F

después de él: 2.3F
, sino por defecto será double
. Esto suele ser una causa habitual de quebraderos de cabeza para los programadores Java novatos que, cuando escriben una sentencia del tipo float f = 2.3;
no comprenden por qué el compilador da un error; estamos asignando un número double
, 2.3
, a una variable de tipo float
. En esta operación puede perderse información y el compilador nos avisa.

está codificado en un formato denominado Unicode. Unicode es una extensión de ASCII, ya que éste último sólo tenía capacidad para representar 256 símbolos distintos. Para poder representar todos los alfabetos (chino, japonés, ruso...) y una mayor cantidad de símbolos se creó el formato Unicode.

En Java, al igual que en C, se distingue la representación de los datos

 char
frente a las cadenas de caracteres. Los char
van entre comillas simples: char ch = ‘a’;
, mientras que las cadenas de caracteres usan comillas dobles.

3.4Datos lógicos: boolean

Se trata de un tipo de dato que solo puede tomar dos valores:

 true
y false
, que representan los valores lógicos cierto y falso, respectivamente. Por ejemplo, la sentencia
 bolean b = true;
inicia la variable b
al valor lógico que representa una condición que se cumple, esto es, una condición cierta. Es un tipo de dato bastante útil a la hora de realizar chequeos sobre condiciones. En C no hay un dato equivalente y para suplir su ausencia muchas veces se emplean enteros con valor 1 si la variable lógica toma el valor lógico cierto y 0 si la variable toma el valor lógico falso.

4 Definición de variables

Al igual que en C, y a diferencia de Fortran, Java requiere que se declaren los tipos de todas las variables empleadas. La sintaxis de declaración es la misma que C:

 int i;

Sin embargo, y a diferencia que en C, se requiere inicializar todas las variables antes de usarlas, si no el compilador genera un error y aborta la compilación. Se puede declarar e inicializar valor a una variable en una misma línea:

```
int i = 0;
```

En Java, después de cada línea de código siempre debe ir un ";". Declaración e inicialización pueden hacerse en líneas diferentes:

```
int i ;
      i = 0;
```

Es posible declarar e iniciar varias variables en una línea:

```
int i=3, j,k=2;
```

Los caracteres aceptados en el nombre de una variable son los comprendidos entre

A-Z
, a-z
, _
, $
y cualquier carácter que sea una letra en algún idioma. En Java, al igual que en todo lenguaje de programación, hay una serie de palabras reservadas que no pueden ser empleadas como nombres de variables (if
, . int
, char
, else
, goto
...); alguna de éstas son empleadas en la sintaxis del lenguaje, otras, como goto
no se emplean en la actualidad pero se han reservado por motivos de compatibilidad; por si se decide emplear en el futuro.

5 Reglas de conversión entre distintos tipos numéricos

Las normas de conversión entre tipos numéricos son las habituales en un lenguaje de programación: si en una operación se involucran varios datos numéricos de distintos tipos todos ellos se convierten al tipo de dato que permite una mayor precisión y rango de representación numérica; así, por ejemplo:

- Si cualquier operando es

double
todos se convertirán en double
.

- Si cualquier operando es

  float
  y no hay ningún double
  todos se convertirán a float
  .

- Si cualquier operando es

  long
  y no hay datos reales todos se convertirán en long
  .

- Si cualquier operando es

  int
  y no hay datos reales ni long
  todos se convertirán en int
  .

- En cualquier otro caso el resultado será también un

  int
  .

Java sólo tiene dos tipos de operadores para operar números enteros: uno que aplica para operar datos de tipo

long
, y otro que emplea para operar datos de tipo int
(esto también sucede con la mayor parte de las CPU actuales). De este modo cuando operemos un
byte
con un byte
, un short con un short

o un short
con un byte
Java empleará para dicha operación el operador de los datos tipo int
, por lo que el resultado de dicha operación será un int
siempre.

Estas conversiones son importantes a la hora de determinar en qué tipo de variable guardamos el resultado de la operación; ésta ha de tener un rango de representación mayor o igual al rango de representación de la variable con mayor rango de representación involucrada en la operación. Si es de rango superior no habrá problemas. Si no respetamos esta regla, el compilador generará un error.

Es posible convertir un dato de jerarquía “superior” a uno con jerarquía “inferior”, arriesgándonos a perder información en el cambio. Este tipo de operación (almacenar el contenido de una variable de jerarquía superior en una de jerarquía inferior) se denomina cast o molde. Para llevar a cabo un cast debemos poner delante de la variable cuyo tipo queremos cambiar, entre paréntesis, el tipo de la variable al cual queremos realizar el cambio; por ejemplo, la siguiente sentencia realiza un cast de

 double
a int
:

```
    int i = (int)3.4;
```

En el listado 2 mostramos un código Java donde se hace uso de un cast. Como podremos comprobar al ejecutar el programa, al realizar un cast la variable de mayor rango es truncada para ser almacenada en la de menor rango; es decir, 3.9999 al transformarse a entero mediante un cast da como resultado 3.

Para comprender el código del listado 2 es importante tener en cuenta que en Java cuando "se suma" un valor numérico a una cadena de caracteres lo que sucede es que se crea una nueva cadena de caracteres igual a la cadena de caracteres original concatenada con el valor; es decir

 "Edad: "+ 23
da como resultado "Edad: 23"
.

: Este código muestra un ejemplo de cast.

```
  //c&oacute;digo Ejemplo2.java del CD
      int i = 9,k;
     float j = 47.9F;
     System.out.println(&ldquo;i: &ldquo;+i + &ldquo; j: &ldquo; +j);
     k = (int)j; //empleo de un cast; k pasa a valer 47
     System.out.println(&ldquo;j: &ldquo; + j + &ldquo; k: &ldquo; +k);
     j = k;//no necesita cast
     System.out.println(&ldquo;j: &ldquo; + j + &ldquo; k: &ldquo; +k);
```

6 Operadores

En este apartado veremos los operadores aritméticos, relacionales y lógicos con los que cuenta Java.

6.1Operadores aritméticos

Los operadores aritméticos de Java son

+ , - , * , /
para suma, resta, producto y división. El operador /
representa la división de enteros si ambos operandos son enteros. El módulo de la división de dos enteros puede obtenerse mediante el operador %
. Por ejemplo, 7/4= 1; 7% 4=3
.

Además, existen los operadores decremento e incremento:

--
y ++
, respectivamente. La operación que realizan son incrementar y decrementar en una unidad a la variable a la que se aplican. Su acción es distinta según se apliquen antes (preincremento, ++a
) o después (postincremento a++
) de la variable. En el caso del operador preincremento, si la variable sobre la que se aplica forma parte de una expresión primero se incrementará el valor de la variable, y a continuación se evaluará la expresión. En el caso del operador postincremento, si la variable sobre la que se aplica forma parte de una expresión, primero se evaluará la expresión y a continuación se incrementará el valor de la variable. Los operadores de predecremento y postdecremento tienen un comportamiento análogo. El listado 3 ilustra estos distintos escenarios.

: Operadores de preincremento y postdecremento.

```
  //c&oacute;digo Ejemplo3.java del CD
      int i = 1;
      System.out.println("i : " + i);
      System.out.println("++i : " + ++i); // Pre-incremento, primero //incrementa y luego imprime por consola
      System.out.println("i++ : " + i++); // Post-incremento, primero imprime //&ldquo;2&rdquo; por consola y luego incrementa i.
      System.out.println("i : " + i);//i por lo tanto vale 3
      System.out.println("--i : " + --i); // Pre-decremento, primero //decrementa i y luego lo imprime p or consola
      System.out.println("i-- : " + i--); // Post-decremento, primero imprime //i por consola y luego de decrementa.
      System.out.println( "i : " + i);//Ahora i vale 1
```

6.2 Operadores relacionales

Los operadores relacionales son operadores que comprueban si se cumple una determinada relación, de igualdad, desigualdad, mayor que... entre dos valores numéricos. El resultado de la aplicación de cualquier operador relacional es siempre un

bolean

, es decir, la expresión que surge de comparar dos variables o literales mediante un operador relacional sólo puede tomar dos valores: true

y false

. En la tabla 3 se muestran los operadores relacionales disponibles en Java, y el listado 4 muestra su uso.

En este listado se generan números aleatorios empleando un objeto de tipo

Random

; en el tercer artículo de esta serie veremos cómo crear objetos desde Java. Para comprender el código el lector lo único que necesita saber es que el comando rand.nextInt()
genera un número aleatorio entero. El resultado de una posible ejecución de este programa se muestra en el listado 5.

Tabla 1: operadores relacionales

Operador	Operación que realiza
==	Test de igualdad
!=	Test de desigualdad
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que

: Uso de operadores relacionales en Java.

```
 //c&oacute;digo Ejemplo4.java del CD
 Random rand = new Random();
 //el m&eacute;todo nextInt() del objeto Random creado genera un n&uacute;mero aleatorio entero.

 //El m&oacute;dulo entre 100 de un entero aleatorio ser&aacute; un entero aleatorio entre 0 y 100.
    int i = rand.nextInt() % 100;
```

```
int j = rand.nextInt() % 100;
System.out.println("i = " + i);
System.out.println("j = " + j);
System.out.println("i > j es " + (i > j));
System.out.println("i < j es " + (i < j));
System.out.println("i >= j es " + (i >= j));
System.out.println("i <= j es " + (i <= j));
System.out.println("i == j es " + (i == j));
System.out.println("i != j es " + (i != j));
```

: Posible resultado de la ejecución del código del listado 4

```
i = 85
j = 4
i > j es true
i < j es false
i >= j es true
i <= j es false
i == j es false
i != j es true
```

6.3Operadores lógicos

Estos operadores se aplican sobre valores lógicos, es decir, sobre datos de tipo

boolean

. En Java hay tres operadores lógicos: la negación, el AND (Y) lógico y el OR (O) lógico. Estos operadores se muestran en la tabla 4. El código del listado 6 imprime la tabla de verdad de operadores lógicos AND y OR, y muestra el efecto del operador de negación.

Tabla 4: Operadores lógicos

Operador	Operación que realiza
!	Not lógico
&&	And lógico
||	Or lógico

: Este código muestra las tablas de verdad de los operadores lógicos AND y OR

```
boolean variableLogica = false, variableLogica2;
variableLogica2 = !variableLogica;
```

```
    System.out.println("variableLogica: "+ variableLogica + ", variableLogica2: "+variableLogica2)
;
    System.out.println("variableLogica&&variableLogica2: " +  (variableLogica&&variableLogica
2));
    System.out.println("variableLogica||variableLogica2: " +  (variableLogica||variableLogica2));
    System.out.println("\n\nTabla de verdad del operador &&:\n");
    System.out.println("false && false: " + (false && false));
    System.out.println("false && true: " + (false && true));
    System.out.println("true && false: " + (true && false));
    System.out.println("true && true: " + (true && true));



    System.out.println("\n\nTabla de verdad del operador ||:\n");
    System.out.println("false || false: "+ (false || false));
    System.out.println("false || true: " + (false || true));
    System.out.println("true || false: " + (true || false));
    System.out.println("true || true: " + (true || true));
```

7 Conclusiones

En este primer artículo de la serie hemos presentado qué es la plataforma Java y cuál es el papel que Java, el lenguaje de programación, juega en ella. Hemos visto cuáles son las características de Java como lenguaje de programación, y hemos aprendido a manejar las herramientas básicas del kit de desarrollo Java. También hemos visto cuáles son los tipos de datos primitivos con los que cuenta Java, cómo definir variables, y los operadores aritméticos, relacionales y lógicos definidos dentro del lenguaje.

Todavía nos queda mucho por andar; en el siguiente artículo de la serie presentaremos otros tipos de datos que se emplean muy comúnmente en el lenguaje, aunque esta vez no son primitivos: Strings, enumeraciones y arrays. Haremos un repaso rápido de la librería matemática que proporciona Java, y mostraremos las estructuras de control de flujo (bucles y condicionales) con las que cuenta el lenguaje. Os espero a todos el mes que viene.

Descargas

- Códigos del artículo