Elementos básicos

Contenido

Literales básicos

Independientemente de su complejidad, un programa siempre realiza operaciones con números, cadenas de texto y otros valores. Estos elementos se denominan **literales**. Hay muchos tipos diferentes de literales en Java, pero en este tema nos centraremos solo en algunos de ellos.

Números enteros

Se utilizan para contar cosas del mundo real. Ejemplos de literales de tipo número entero: `0`, `1`, `2`, `10`, `11`, `100`.

Si un valor entero contiene muchos dígitos, podemos añadir guiones para dividirlo en bloques para una mayor facilidad de lectura; por ejemplo: `1_000_000`. De esta manera es más legible que el mismo valor escrito como `1000000`.

Caracteres

Un solo carácter puede representar un dígito, una letra u otro símbolo.

Para escribir un carácter usamos comillas simples de la siguiente manera: `'A', 'B', 'C', 'x', 'y', 'z', '0', '1', '2', '9'`.

Los literales de caracteres pueden representar símbolos de un alfabeto, dígitos del `'0'` a `'9'`, espacios en blanco (`' '`) u otros caracteres o símbolos (`'$'`).

No confundas los caracteres que representan números (por ejemplo: `'9'`) con los números propiamente dichos (por ejemplo: 9). Un carácter no puede incluir dos o más dígitos o letras porque representa solo un símbolo. Los dos ejemplos siguientes son incorrectos : `'abc', '543',` y lo son porque estos literales contienen demasiados caracteres.

Al conjunto de caracteres con los que trabaja una aplicación se le denomina charset, y todos ellos son una ampliación del Código ASCII:

http://www.asciitable.com/

Cadenas de caracteres (Strings)

Una cadena es una secuencia de cualquier conjunto de caracteres individuales. Las cadenas representan información textual, como textos publicitarios (`“¡Yo no soy tonto!”`), la dirección de una página web (`“https://es.wikipedia.org/”`) o el nombre de usuario que utilizas para acceder mediante login a un sitio web (`“fulanit@”`).

Para escribir un String se utilizan comillas dobles (no comillas simples, esas son para los caracteres). Algunos ejemplos válidos: `“texto”`, `“Quiero aprender Java”`, `“123456”`, `“yo@educa.madrid.org”`. Un String formado por un único carácter como `“A”` es perfectamente válido, pero no es lo mismo que el carácter `‘A’`.

Como puedes ver, los Strings pueden incluir letras, dígitos, espacios en blanco y otros caracteres.

Recuerda:

No confundas estos literales:

- 123 es un número entero, “123” es una cadena.
- ‘A’ es un carácter, “A” es una cadena.
- ‘1’ es un carácter, 1 es un número entero.

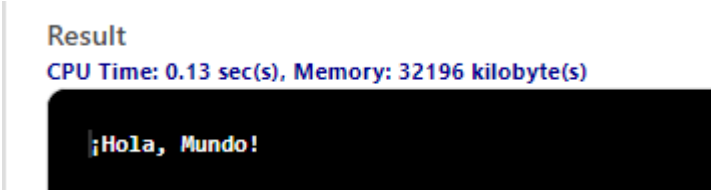
Visión general de un programa básico en Java

En este tema crearemos nuestro primer programa Java. Nuestro programa simplemente imprimirá "¡Hola, mundo!" en la pantalla (una tradición de la mayoría de los programadores cuando aprenden nuevos lenguajes). Nuestro código puede no parecer demasiado emocionante al principio, sin embargo, aprenderemos acerca de la plantilla básica que todos los programas Java deben seguir.

El código Java de este programa es el siguiente:

```
1  public class Main {
2      public static void main(String[] args) {
3          System.out.println("¡Hola, Mundo!");
4      }
5  }
```

Escribe exactamente esto en el editor y luego haz clic en el botón "Ejecutar" ( ). En el terminal verás el resultado.



Terminología básica

Veamos algo de terminología básica para comprender cualquier programa:

- **Programa:** secuencia de instrucciones que se ejecutan una tras otra de manera secuencial, en el orden en que están escritas.

- **Instrucción:** acción (como imprimir un texto) terminada por punto y coma (;).

- **Bloque:** grupo de cero, una o más instrucciones encerradas entre un par de llaves {...}. Hay dos bloques de este tipo en el programa anterior: el delimitado por la clase y el del método main.

- **Método:** secuencia de instrucciones que representa una operación de alto nivel (también conocida como subprograma o procedimiento).

- **Sintaxis:** conjunto de reglas que definen cómo se debe escribir un programa para que sea válido; cada lenguaje de programación tiene su propia sintaxis.

- **Palabra clave**: palabra que tiene un significado especial en el lenguaje de programación (`public`, `class` y muchas otras). Estas palabras no se pueden utilizar como nombres de variables para tu propio programa.

- **Identificador o nombre**: palabra que se refiere a algo en un programa (como una variable o el nombre de una función).

- **Comentario**: explicación textual de lo que hace el código. Los comentarios de Java comienzan con //, si son de una línea, o vienen delimitados por /*….*/ si se extienden en varias líneas.

El programa "Hola Mundo"

El programa "Hola, Mundo" ilustra los elementos básicos de los programas Java. Por ahora, comentaremos sólo los más importantes.

1. La clase **`public`**. Es la unidad básica de un programa. Cada programa Java debe tener al menos una clase. La definición de una clase consta de la palabra clave **class** seguida del nombre de la clase, en mayúscula. Una clase puede tener cualquier nombre, como por ejemplo **`App`**, **`Main`** o **`Programa`**, pero **no puede comenzar con un dígito**. El cuerpo de la clase tiene que estar encerrado entre un par de llaves `{...}`.

```
1  public class Main {
2      // ...
3  }
```

2. El método principal. Para que el programa se pueda ejecutar, colocamos un método llamado **`main`** dentro de una clase. Es el punto de entrada para un programa Java. Nuevamente, las llaves `{...}` encierran el cuerpo del método, que contiene instrucciones de programación.

```
1  public class Main {
2      public static void main(String[] args) {
3          // Aquí van las instrucciones
4      }
5  }
```

Las palabras clave `public`, `static` y `void` se discutirán más adelante, así que simplemente recuerda escribirlas por ahora. El nombre de este método (**`main`**) está predefinido y siempre debe ser el mismo. La capitalización importa: si nombras a este método como Main , MAIN o de cualquier otra forma, el programa no puede iniciarse.

El elemento `String[] args` representa una secuencia de argumentos pasados al programa desde el exterior. No te preocupes por ello de momento.

3. Imprimir "¡Hola, mundo!" . El cuerpo del método consta de instrucciones de programación que determinan qué debe hacer el programa después de iniciarse. Nuestro programa imprime la cadena "¡Hola, mundo!" usando la siguiente instrucción:

```
3          System.out.println("¡Hola, Mundo!"); // Cada instrucción debe terminar con ;
```

Esta es una de las cosas más importantes que debe comprender el programa "Hola, Mundo". Invocamos un método llamado `println` que lo que hace es mostrar en pantalla una cadena seguida de una nueva línea. Observa que el texto queda impreso sin comillas dobles.

(Importante: "¡Hola, mundo!" no es una palabra clave o un identificador; es solo un texto para imprimir como podríamos poner cualquier otra cosa).

Palabras clave

Como puedes ver, incluso un programa Java simple consta de muchos elementos, incluidas palabras clave que forman parte del lenguaje. En total, Java tiene más de 50 palabras clave.

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

main está fuera de la lista porque no es una palabra clave

Visualizar datos en el terminal

Visualización de texto usando println () y print ()

La salida estándar es un receptor al que un programa puede enviar información (texto). Es compatible con todos los sistemas operativos habituales. Java proporciona un objeto especial `System.out` para trabajar con la salida estándar. A menudo lo usaremos para imprimir por pantalla.

El método `println` muestra la cadena pasada seguida de una nueva línea en la pantalla (**print-line**). Como ejemplo, el siguiente fragmento de código imprime cuatro líneas.

```
System.out.println("Yo ");
System.out.println("sé ");
System.out.println("un montón de ");
System.out.println("Java.");
```

Resultado:

```
Yo
sé
un montón de
Java.
```

Si no se proporciona una cadena, se imprime una línea en blanco:

```
System.out.println("Java es un lenguaje de programación muy popular.");
System.out.println(); // imprime una línea en blanco
System.out.println("¡Se utiliza en todo el mundo!");
```

Resultado:

```
Java es un lenguaje de programación muy popular.


¡Se utiliza en todo el mundo!
```

El método `print` muestra el valor pasado y coloca el cursor después de él. Como ejemplo, el siguiente código genera todas las cadenas en una sola línea.

```
System.out.print("Yo ");
System.out.print("sé ");
System.out.print("un montón de ");
System.out.print("Java.");
```

Resultado:

```
Yo sé un montón de Java.
```

Fíjate en que los espacios entre palabras debemos incluirlos dentro de las cadenas de texto.

Impresión de números y caracteres

Ambos métodos (`println` y `print`) permiten imprimir no sólo cadenas, sino también números y caracteres.

Por ejemplo:

```
System.out.print(108);   // imprime un número
System.out.print('c');   // imprime un carácter que representa una letra
System.out.print("Q");   // imprime una cadena de caracteres (String)
System.out.println('3'); // imprime un carácter que representa un dígito

System.out.print(22);
System.out.print('E');
System.out.print(8);
System.out.println('1');
```

Resultado:

```
108cQ3
22E81
```

Observa que los caracteres se rodean de comillas simples y las cadenas con comillas dobles.

Comentarios

Dentro de un programa Java podemos escribir texto que será ignorado por el compilador de Java, conocido como comentario. Los comentarios permiten excluir código del proceso de compilación (deshabilitarlo) o escribir algo que aclare lo que hace un fragmento de código, para ti mismo o para otros desarrolladores (de hecho, se agradece mucho cuando debes trabajar a partir del código que ha escrito otra persona y ésta se preocupó por dejarlo bien comentado).

El lenguaje de programación Java admite tres tipos de comentarios.

Comentarios de una línea

El compilador de Java ignora cualquier texto desde `//` hasta el final de la línea.

Por ejemplo:

```
class Program {
    public static void main(String[] args) {
        // La línea que está debajo de ésta será ignorada
        // System.out.println("Hola, Mundo");
        // Imprime la cadena "Hola, Java"
        System.out.println("Hola, Java"); // Aquí va cualquier comentario
    }
}
```

Comentarios de varias líneas

El compilador ignora cualquier texto entre `/*` y el `*/` más cercano . Se puede utilizar para comentar una sola línea como para comentar múltiples líneas a la vez.

```
class Programa {
    public static void main(String[] args) {
        /* Este es un comentario en una sola línea */
        /*  Este es un ejemplo de comentario
            de varias líneas */
  }
}
```

Se pueden meter comentarios dentro de comentarios:

```
class Programa {
    public static void main(String[] args) {
        /*
        System.out.println("Hola"); // imprime "Hola"
        System.out.println("Java");  // imprime "Java"
        */
    }
}
```

El compilador ignora todo lo que está dentro de `/* ... */`.

Comentarios de la documentación de Java (javadoc)

El compilador ignora cualquier texto desde `/**` hasta `*/` igual que ignora los comentarios de varias líneas.

Este tipo de comentarios se pueden utilizar para generar automáticamente documentación sobre su código fuente utilizando la herramienta javadoc. Por lo general, estos comentarios se colocan sobre declaraciones de clases, interfaces, métodos, etc. Algunas etiquetas especiales como `@param` o `@return` se utilizan a menudo con funciones de control. Sin embargo, son opcionales y no nos ocuparemos de ellos por ahora, pero no te sorprendas si te encuentras con alguno.

```
class Programa {
    /**
     * El método main acepta un array de argumentos de tipo String
     *
     * @param args desde la línea de comandos
     */
    public static void main(String[] args) {
        // esto no hace nada
    }
}
```

Introducir datos por teclado

La **entrada estándar** (`System.in`) es un flujo de datos que ingresamos a un programa y es soportada (y gestionada) por el sistema operativo. De forma predeterminada, la entrada estándar obtiene datos de la entrada del teclado, pero es posible redirigirla a un fichero en su lugar.

En realidad, los programas no siempre utilizan la entrada estándar, pero aquí la vamos a usar mucho sobre todo al principio, mientras vamos aprendiendo a programar. La forma típica de resolver problemas de programación es la siguiente:

1. Leer datos de la entrada estándar (`System.in`);
2. Procesar datos para obtener un resultado;
3. Enviar el resultado a la salida estándar (`System.out`).

Leyendo datos con un Scanner

La forma más sencilla de obtener datos de la entrada estándar es utilizar la clase `Scanner`, que permite introducir al programa valores de diferentes tipos (cadenas, números, etc.) de la entrada estándar, que en nuestro caso es el teclado.

Para poder usar esta clase, antes debemos importarla añadiendo la siguiente instrucción en la parte superior del archivo de código fuente:

```
import java.util.Scanner;
```

Y ahora ya sí que podemos utilizarla. Y lo haremos añadiendo la siguiente instrucción después de la importación:

```
Scanner scanner = new Scanner(System.in);
```

Con esta línea, creamos un objeto de tipo `Scanner`, y a partir de entonces ya podemos utilizar sus métodos. Aprenderemos más sobre la creación de objetos más adelante. `System.in` indica que el programa leerá el texto que escriba en la entrada estándar. Por ahora, siempre escribiremos exactamente esta instrucción, aunque el nombre que le demos al objeto (en este caso `scanner` sí que puede ser el que nosotros queramos).

Hay varias formas de leer cadenas con un objeto de tipo `Scanner`.

- Si la entrada es un número entero o una sola palabra, se usar el método `next()`. Como ejemplo, el siguiente fragmento de código lee el nombre del usuario e imprime un mensaje de saludo :

```
String nombre = scanner.next();

System.out.println("¡Hola, " + nombre +"!");
```

En el ejemplo anterior, si ponemos Lalala escribirá “Hello, Lalala!”. Pero si el nombre es compuesto, como por ejemplo “María Lalala”, este método sólo recogerá lo que encuentre hasta el primer espacio. En este caso escribiría “Hello, María!”. En este caso deberíamos haber utilizado el método que describimos en el siguiente punto.

- Si la entrada contene más de una palabra, necesitamos un método que lea una lína completa, y eso es lo que hace `nextLine():`

```
String nombre = scanner.nextLine();

System.out.println("¡Hola, " + nombre +"!");
```

El término más correcto para referirnos a lo que hemos llamado "palabra" es token: es un fragmento de texto rodeado de espacios en blanco. Podemos decir ahora que el método `next()` recoge el siguiente token que encuentra, mientras que `nextLine()` lee todos los datos hasta el final de la línea actual.

- Si vamos a introducir números y queremos que los recoja directamente como un tipo concreto (`int`, `double`, etc.), por ejemplo para asignarlo a una variable de ese tipo, tenemos métodos concretos para hacerlo: `nextInt()`, `nextDouble()`, etc. Aunque aún no hemos trabajado con variables y tipos, dejamos aquí esto para cuando nos haga falta.

NOTA:

Cuando se acepta por teclado una cadena con caracteres en blanco debemos llamar al método nextLine(). Pero si previamente hemos llamado al método nextInt(),no nos permite aceptarla. Esto es debido a que después de ejecutar el método nextInt() queda almacenado en el objeto de la clase Scanner el caracter "Enter" y si llamamos inmediatamente al método nextLine() este almacena dicho valor de tecla y continúa con el flujo del programa.

Para solucionar este problema llamamos al método nextLine() dos veces, la primera retorna la tecla "Enter" y la segunda se queda esperando que aceptemos el resto de campos (tener en cuenta que esto es necesario solo si previamente se llamó al método nextInt() o nextFloat().)

Si la cadena a aceptar no tiene blancos, podemos utilizar el método next()

```java
// introducimos un número por teclado
System.out.println("Introduce un número: ");
int num = sc.nextInt();
// lo imprimimos
System.out.println(num);

// para evitar que después de nextInt() "salte", añadimos un nextLine
sc.nextLine(); /* prueba a ejecutar el código comentando esta línea, para
que veas que es necesaria */

// ahora vamos a introducir un String
System.out.println("Introduce un String:");
String s = sc.nextLine();
System.out.println(s);
```

Variables y tipos de datos

Variables: qué son y cómo se utilizan

Una **variable** es un elemento que utilizamos para almacenar un valor de un **tipo** determinado (una cadena, un número entero, un número decimal, un carácter…). Cada variable tiene un **nombre** (también conocido como **identificador)** para distinguirla de otras. Para poder usar una variable, lo primero que tenemos que hacer es declararla.

La forma general de **declarar una variable** es la siguiente:

```
TipoDeDato identificador = valor_inicial;
```

Ejemplo:

```java
int x=10;
int y=25;
int z=x+y;
```

Lo que queda a la izquierda del igual es la **declaración**, que **comprende el tipo de dato y el nombre de la misma (**en el ejemplo hay tres declaraciones de variables: `int x`, `int y` e `int z`**).**

Para guardar un valor en esa variable tenemos que inicializarla. **Inicializar una variable** consiste en darle un valor determinado que más adelante podemos cambiar por otro a lo largo del programa. Es lo que queda a la derecha del igual tras la declaración (en nuestro ejemplo hemos inicializado la variable x con el valor 10, la variable y con el valor 25 y la variable z con el resultado de sumar lo que valga x en este momento más lo que valga y en este momento).

Si declaramos una variable pero no la inicializamos, el valor que tiene esa variable es lo que se conoce como **valor por defecto**, que en el caso de las variables numéricas es 0.

De modo que tenemos:

- **`TipoDeDato`**: el tipo (o tipo de dato) de una variable determina qué posibles operaciones se pueden realizar con la variable y qué valores se pueden almacenar en ella.
- **`Identificador:`** es el nombre que queramos darle a la variable. Generalmente comienza con una letra, y nunca puede empezar por un dígito. Intenta elegir simpre nombres significativos y legibles para que el código sea fácil de entender.

- **Operador de asignación (=):** asigna el valor que ponemos a la izquierda a la variable que nombramos a la derecha.
- **Inicialización:** es el valor (o también puede ser el resultado de una expresión) que asignamos a la variable.

Un ejemplo de variable en forma de cadena de texto (tipo String):

```
String lenguaje = "java";
```

(¡Ojo! Java es *case sensitive*, esto es, distingue entre mayúsculas y minúsculas; la variable lenguaje es diferente de la variable Lenguaje).

Acceder al valor de una variable

Una vez que se ha declarado una variable, se puede acceder a su valor y modificarlo usando su nombre. En el siguiente ejemplo, declaramos una variable y luego la mostramos:

```
String dayOfWeek = "Monday";
System.out.println(dayOfWeek); // Monday
```

También es posible asignar un valor de una variable a otra:

```
int one = 1;
int num = one;
System.out.println(num); // 1
```

Una característica importante de las variables es que se pueden cambiar. No es necesario volver a declarar una variable para cambiar su valor; simplemente le asignamos un nuevo valor usando el operador =.

Declaremos una variable llamada dayOfWeek e imprimamos su valor antes y después de cambiar:

```
String dayOfWeek = "Monday";
System.out.println(dayOfWeek); // Monday

dayOfWeek = "Tuesday";
System.out.println(dayOfWeek); // Tuesday
```

Las variables tienen una restricción importante: solo puede asignar un valor del mismo tipo que el tipo de la variable inicial (en general, veremos más adelante que hay algunas excepciones concretas). Por ejemplo, el siguiente código no es correcto:

```
int number = 10;
number = 11; // ok
number = "twelve"; // esto no va a funcionar
```

Formas alternativas para declarar una variable.

Hay varias formas alternativas de declaración que se utilizan con menos frecuencia en la práctica. Aquí hay varios de ellos en ejemplos particulares.

- Declarar varias variables del mismo tipo en una única línea:

```
String language = "java", version = "8 or newer";
```

- Separación de declaración e inicialización en líneas diferentes:

```
int age; // declaration
age = 35; // initialization
```

Inferencia de tipos.

Desde Java 10, se puede escribir **var** en lugar de un tipo específico para forzar la inferencia automática de tipos según el tipo de valor asignado:

```
var nombreVariable = inicialización;
```

Por ejemplo:

```
var language = "Java"; // String
var version = 10; // int
```

Nosotros no vamos a usar la inferencia de tipos por si tuviéramos que programar en versiones anteriores de Java.

Reglas para nombrar variables

Cada **variable** tiene un **nombre** que la identifica de forma única entre otras variables. Dar un buen nombre a una variable no es tan simple como parece. Los programadores experimentados ponen mucho cuidado en esto para que sus programas sean fáciles de entender, porque un programador debe dedicar mucho tiempo a leer y comprender el código escrito por otros. Si las variables tienen nombres incorrectos, incluso tu propio código te parecerá confuso al cabo de unos meses.

Intenta siempre dar nombres descriptivos y concisos a todas las variables.

Java tiene algunas reglas para nombrar variables:

- los nombres distinguen entre mayúsculas y minúsculas;
- un nombre puede incluir letras, números y dos caracteres especiales ($, _);
- no se permiten espacios en blanco en los nombres de variables.
- un nombre no puede comenzar con un dígito;
- un nombre no puede ser una palabra clave (`class, static, int`, etc, son nombres ilegales).

A continuación se muestran algunos nombres válidos de variables:

```
number, $ident, bigValue, _val, abc, k, var
```

Para mantener la compatibilidad con versiones anteriores, la palabra var se puede usar como nombre de variable incluso después de haberse lanzado Java 10.

Y aquí hay algunos nombres de variables inválidos:

```
@ab, 1c, !ab, class
```

Desde Java 9, el carácter único _ es un nombre no válido para una variable, pero _a y __(doble _) son nombres legales.

Convenciones de nomenclatura para variables.

Además, existen las siguientes convenciones para nombrar variables:

- si un nombre de variable es una sola palabra que debe estar en minúsculas (por ejemplo: `number, price`);
- si el nombre de una variable incluye varias palabras, debe estar en `lowerCamelCase`, es decir, la primera palabra debe estar en minúsculas y cada palabra después de la primera debe tener su primera letra escrita en mayúsculas (por ejemplo:) `numberOfCoins;`
- los nombres de las variables no deben comenzar con los caracteres `_` y `$`, aunque están permitidos;
- elija un nombre que tenga sentido, por ejemplo, `score` tiene más sentido que `s`, aunque ambos son válidos.

Estas convenciones son opcionales, pero se recomienda encarecidamente seguirlas porque hacen que su código más legible para todos.

Referencias

https://hyperskill.org