Introducción a Java

Hola mundo

¿Cómo escribir un programa Java?

Un programa en código Java no es más que un documento de texto con sentencias y sintaxis de Java que puedan ser comprendidas por el compilador.

Este fichero de código fuente ha de tener la extensión `.java` para que pueda ser reconocido por los IDE (Entornos integrados de desarrollo).

Así, para crear nuestro primer programa Java lo único que hemos de hacer es escribir las siguientes líneas de código en un fichero **de nombre** `HolaMundo.java` (es necesario el nombre `HolaMundo` como explicaremos más adelante):

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

¿Cómo compilar / ejecutar un programa?

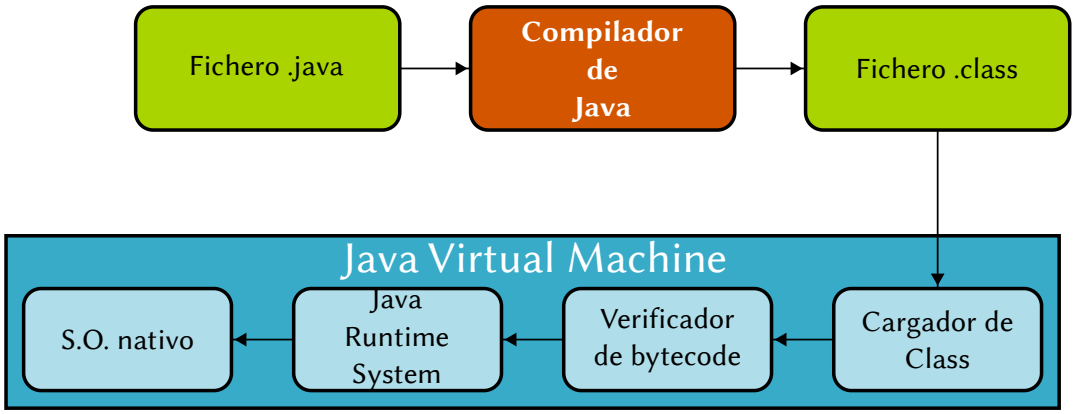
¿Qué significa compilar?

Compilar un programa consiste en *convertir* el **código fuente** en **código binario**. El código fuente es el programa tal como nosotros lo hemos escrito en el editor de código, de manera que es legible por los seres humanos. Por el contrario el **código binario** *normalmente* es únicamente comprensible por el **Sistema Operativo** que puede procesarlo y enviarlo al procesador del ordenador para que se ejecuten las instrucciones que escribió el programador.

Como podemos ver el **código binario (o simplemente binario)** de un programa es dependiente del **sistema operativo** y el tipo de **procesado**r (arquitectura) de la máquina en la que se quiera ejecutar.

En Java este proceso es algo diferente. En lugar de compilar para un sistema operativo / arquitectura lo que hace el compilador de Java es crear un binario para ser ejecutado en la **máquina virtual de java (JVM)**. Esta *máquina virtual de java* no es más que un programa que entiende el código binario de Java (Java bytecode).

De modo que un programa Java, una vez compilado, podrá ser ejecutado por **cualquier JVM**. Lo que necesitaremos para poder ejecutar un **bytecode** de Java en cualquier arquitectura es una **JVM** que corra sobre ella.



Entonces... para ejecutar un programa Java ¿Qué?

Para ejecutar un programa Java hemos de seguir los siguientes pasos:

1. Compilar a bytecode de Java: `javac -verbose HolaMundo.java`.
2. Enviarle el bytecode de Java (`HolaMundo.class`) a la máquina virtual: `java HolaMundo`.

Nota: al ejecutar el segundo paso **no se ha de incluir el .class** al indicar la clase que queremos ejecutar.

Comentarios

Cuando escribamos código en cualquier lenguaje hemos de tener presente que no seremos únicamente nosotros los que tendremos que entender y posiblemente modificar ese código. Para ello, cuando estimemos que algo puede ser confuso (para nuestro yo futuro o para otra persona), es conveniente **comentar el código**.

En Java hay tres tipos de comentarios:

- Comentario de una línea: Se indica escribiendo `//` antes de comenzar a escribir el comentario.

```
int a = 10;
int b = 10;
int c = a + b; // Esto no hace falta ni comentarlo.
```

- Comentario de varias líneas: Se indica escribiendo `/*` al comienzo de las líneas de comentario y `*/` al final:

```
/* Y esto es un ejemplo de como se puede escribir un comentario
que se extienda várias líneas. */
String str = "Hola mundo";
System.out.println(str)
```

- Comentarios para **Javadoc**: Los comentarios para Javadoc son similares a los comentarios para várias líneas pero comienzan con `/**` en lugar de `/*`.

Variables

Una variable es un ***nombre*** que estará enlazado con un ***espacio de memoria*** donde se podrá almacenar algún ***dato***.

Para poder utilizar una variable en Java habrá de **declararla** antes.

Declaración de una variable

Para declarar una variable lo primero que hemos de indicar es el **tipo de dato** de la misma. El tipo de dato determina que tipo de valores podrá contener la variable.

A continuación indicaremos el nombre de la variable.

Finalmente, y de manera opcional, podremos asignar un **valor inicial** al la variable utilizando el **operador de asignación** `=`.

```
<tipo de dato> <nombre de la variable> [= valor];
```

```
int x = 1;
float y = 1.5;
String texto = "Hola mundo."
```

Tipos de datos

Java es un lenguaje de **tipado estático**. Esto quiere decir que hay que indicar siempre el **tipo de dato** de una variable.

Por esto se dice también que Java es **fuertemente tipado**.

Un tipo de dato es una declaración mediante la que se indica el contenido que podrá tener una variable.

Consecuencias

Si intentamos guardar en una variable un dato que no encaja con el que hemos indicado en la declaración se producirá un **error de compilación**.

```
int x;
x = 12.5
```

```
.\tarea1\MiNombre.java:4: error: incompatible types: possible lossy conversion
from double to int
        x = 1.2;
            ^
1 error
```

Tipos primitivos

Estos tipos de datos de base tienen como única característica que **no son objetos**. El resto de elementos del lenguaje Java **son objetos**.

- **byte**: números enteros que se pueden representar mediante un byte (8 bits). Es decir, los números entre el -128 y el 127 (ambos inclusive).
- **short**: números enteros de 16 bits. Los valores enteros entre -32 768 y 32 767.
- **int**: similar a los anteriores pero usando 32 bits (-2^31)
- **long**: lo mismo pero con 64 bits.
- **float**: números reales expresados mediante 32 bits.
- **double**: números reales representados mediante 64 bits.
- **boolean**: dos valore (lógicos) posibles: **true** y **false**. Utiliza un bit de datos.

- **char**: representa un carácter Unicode de 16 bits.

¿Y no hay más tipos de datos?

El resto de tipos de datos en Java **son objetos**.

Algunos tipos de datos, aunque sean objetos, tienen comportamientos ***especiales*** para facilitar la escritura de código. Un ejemplo son los tipos **String** y **Array**.

Tipo `String`

El tipo **String** se utiliza para almacenar y realizar operaciones sobre cadenas de caracteres.

Para crear una variable que haga referencia a una cadena de caracteres escribiremos los siguiente:

```
String str = "Hola mundo.";
```

Esta es la forma abreviada de escribir el código. La forma ***correcta*** teniendo en cuenta que **String** es un objeto sería:

```
String str = new String("Hola mundo.");
```

Nombres de variables

Los nombres de las variables han de cumplir las siguientes reglas:

- Han de empezar por **minúscula** (opcional pero recomendado)**, subrayado o $**.
- Han de estar compuestas por caracteres **Unicode**.
- No pueden coincidir con el nombre de las **palabras reservadas** del lenguaje.

Operadores

Operadores *postfijo*

Estos operadores valen para incrementar (`++`) o decrementar (`--`) en una unidad el valor de una variable.

```
int x = 1;
x++; // x pasa a valer 2.
System.out.println(x); // Se mostrará el valor 2.
x--; // x pasa a valer 1.
```

Operadores unarios

Son similares a los anteriores pero con menor prioridad:

```
int x = 1;
++x; // x pasa a valer 2.
--x; // x pasa a valer 1.
```

Operadores aritméticos

Los operadores aritméticos sirven, como su nombre indica, para realizar operaciones aritméticas. La prioridad de ejecución entre ellos es la común entre las calculadores y el uso general:

```
int a = 10; // Operador de asignación =. Asigna a una variable un valor.
int b = 20;
int d = 30;

d = a * b; // Asignamos a c el producto de a por b.
d = a / b;
d = a % b; // Operador módulo que guarda en c el RESTO de la division de a entre
b.
d = a + b * c; // Primero multiplicamos b por c y al resultado le añadimos a.
d = a - b * c; // Le restamos a a el producto de b por c.
```

Operadores a nivel de bit

- `<<`: Desplaza el número a su izquierda tantas posiciones como indique el número a su derecha:

  ```
  int a = 1; // 0001 en binario.
  int b = 3;
  int c = a << b;
  System.out.println(c) // Mostrará el valor 8 (1000)
  ```

- `>>`: Igual que el anterior pero desplazando a la derecha:

  ```
  int a = 8; // 1000.
  int b = 3;
  inc c = a >> b;
  System.out.println(c); // Mostrará 1 en pantalla.
  ```

- `&`: **And** a nivel de bits.
- `|`: **Or** a nivel de bits.
- `^`: **Xor** a nivel de bits.

Operadores realacionales

Estos operadores sirven para comparar dos valores. Devolverán **true** o **false** dependiendo de si la comparación es *cierta* o no.

- `==`: El operador **igual** compara si dos valores son iguales.

  ```
  int a = 1;
  int b = 2;
  System.out.println(a == b); // Mostrará false.
  System.out.println(a == b - 1); // Mostrará true.
  ```

- `!=`: El operador **distinto** devolverá **cierto** cuando los dos valores comparados sean **distintos**:

```
int a = 1;
int b = 2;
System.out.println(a != b); // Mostrará true.
System.out.println(a != b - 1); // Mostrará false.
```

- `>` y `>=`: Los operadores **mayor** y **mayor o igual** realizan dicha comparación:

```
int a = 0;
int b = 10;
int c = 10;
boolean resultado = a > b; // resultado será igual a false.
resultado = b >= c; // resultado será igual a true.
```

- `<` y `<=`: Son similares a los anteriores.

Operadores lógicos

Estos operadores sirven para:

- **negar** (operador `!`) un valor:

```
boolean t = true; // Asignamos verdadero a t.
boolean f = !b; // Asignamos no verdadero (= falso) a f.
System.out.prinln(f); // Mostrará false por pantalla.
```

- Hacer la operación **y** (and) de dos valores (operador `&&`):

```
boolean t = true;
boolean f = false;
System.out.println(t && f); // true && false es igual a false.
System.out.pringln(t && t); // true && true es igual a true.
```

- Hacer la operación **o** (or) de dos valores (operador `||`):

Operador ternario

No es conveniente usarlo ya que vuelve el código difícil de seguir.

Tiene tres partes: <condición> **?** <resultado del **true**> **:** <resultado del **false**>

```
int a = 10;
int b = 20;
System.out.println(a == b ? "a igual a b" : "a distinto de b."); // Mostrará "a
distinto de b." por pantalla.
```

Operador de asignación

Sólo hay uno `=` y sirve para asignar un valor a una variable:

```
String a = "un valor";
String b = a;
boolean c = a == b; // ¿Cuál será el valor de c?
b = "un valor";
c = a == b; // ¿Y ahora?
b = new String("un valor");
c = a == b; // ¿¿Y AHORA??
```

INCISO: Entrada de datos

Para poder hacer ejercicios necesitamos interactuar con el programa o, al menos, poder indicar con qué datos va a trabajar (datos de entrada).

Para ello hemos de ver un par de formas en las que podemos pasar valores a nuestros programas.

Estas instrucciones van a ser muy superficiales, simplemente para poder realizar entrada de datos. Más adelante las analizaremos en más detalle.

Entrada por consola

Para leer de la consola (entrada por teclado) primero hemos de acceder a ella. Para hacer esto usaremos la clase `System`:

```
import java.io.Console;

public class InputOutput {
    public static void main(String[] args) {
        Console c = System.console(); // Accedemos al objeto Console del
sistema.
        System.out.print("Dime tu nombre: ");

        String nombre = c.readLine(); // Quedará esperando a que escribamos un
texto y lo guardará en nombre.

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

NOTA:

Los datos leídos con `readLine` siempre será **cadenas de caracteres**. Para poder usarlos como números habrá que convertirlos en `int`, `float`, etc.

Para ello hemos de usar los métodos / funciones siguientes:

- `Integer.parseInt()`: para convertir una cadena a un valor entero.
- `Float.parseFloat()`: para obtener un valor.
- `Double.parseDouble()`: ...

Veámoslo en un ejemplo:

```
import java.io.Console;

public class InputNumbers {
    public static void main(String[] args) {
        Console c = System.console(); // Accedemos al objeto Console del
sistema.
        System.out.print("Introduzca un número: ");

        String str_numero = c.readLine(); // Guaramos lo escrito por teclado en
numero.
        int numero = Integer.parseInt(str_numero);
        System.out.println("El valor " + str_numero + " + 10 vale " + (numero +
10) + ".");
    }
}
```

Argumentos

Cuando creamos el método `main` podemos ver que a continuación sigue la siguiente secuencia `(String[] args)`. Esta última parte está indicando el que método `main` recibe como argumento un **array de cadenas**.

Este método será utilizado por la máquina virtual de Java para pasarle al programa **los argumentos que indiquemos por consola en el momento de ejecutarlo**.

Veámoslo con un ejemplo:

```
public class Argumentos {
    public static void main(String[] args) {
        System.out.println("El primer argumento fue: " + args[0]);
        System.out.println("El segundo argumento fue: " + args[1]);
        System.out.println("El tercer argumento fue: " + args[2]);
    }
}
```

Si ahora compilamos e invocamos el programa:

```
javac Argumentos.java
java Argumentos Adiós Mundo Cruel
```

El resultado será:

```
El primer argumento fue: Adiós
El segundo argumento fue: Mundo
El tercer argumento fue: Cruel
```

Estructuras de control

En Java hay tres sentencias de control:

- Condicionales: Se determina la ejecución de cierto código fuente dependiendo de una condición:

    - `if else` / `else if`
    - `switch`
- Repetitiva: Se repetirá una secuencia de código tantas veces como se indique o hasta que se cumpla (o deje de cumplirse) una condición:
    - `for`
    - `while`
    - `do ... while`
- Salto incondicional: Se utilizan principalmente para **saltar** o **salir** de la ejecución de una **iteración** de un **bucle**:
    - `break`
    - `continue`

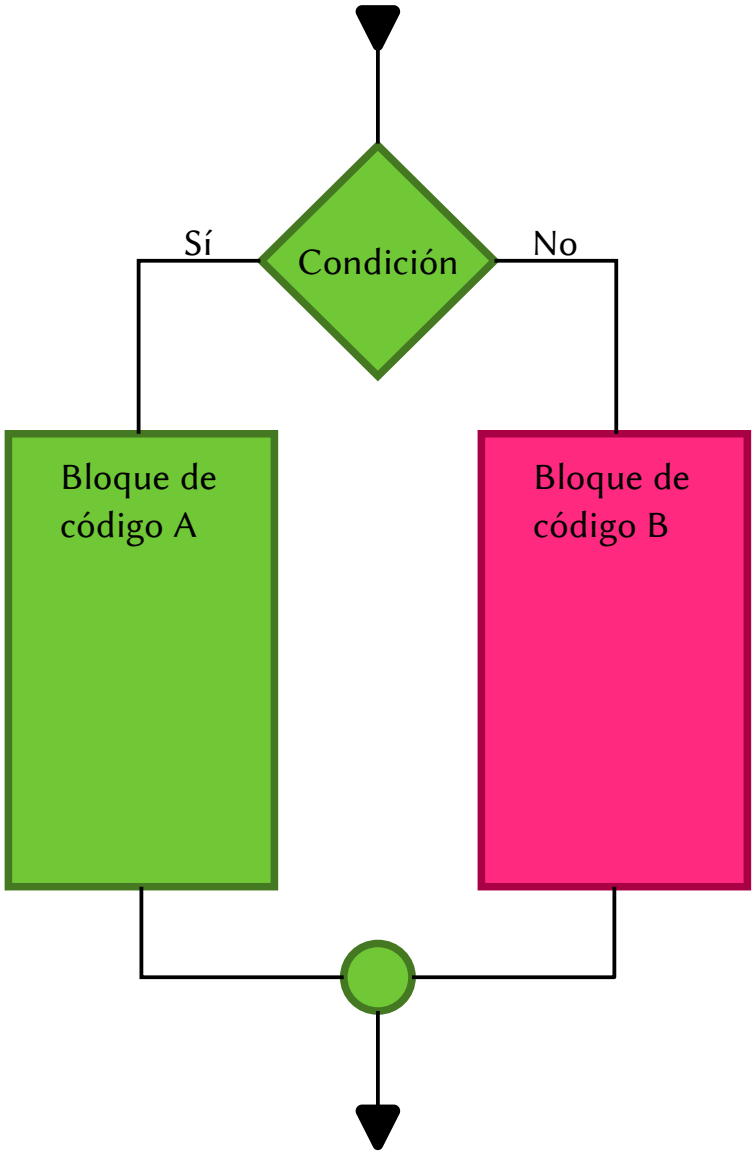
Nota:

A una estructura de control repetitiva se la llama también **bucle**.

Cada una de las veces que se ejecuta el código de un **bucle** se dice que se ha producido una **iteración**.

Estructuras condicionales

`if else` / `else if`



Esta estructura permitirá que se ejecute un bloque de código u otro dependiendo de la condición.

Veámoslo con un ejemplo:

```
if (age >= 18) {
    // Bloque de código A.
    System.out.println("Eres mayor de edad.");
} else {
    // Bloque de código B.
    System.out.println("Eres menor de edad.");
}
```

Variantes:

Una variante sería **omitir** la rama del **else**:

```
if (age < 18) {
    System.out.println("Eres menor.")
}
```

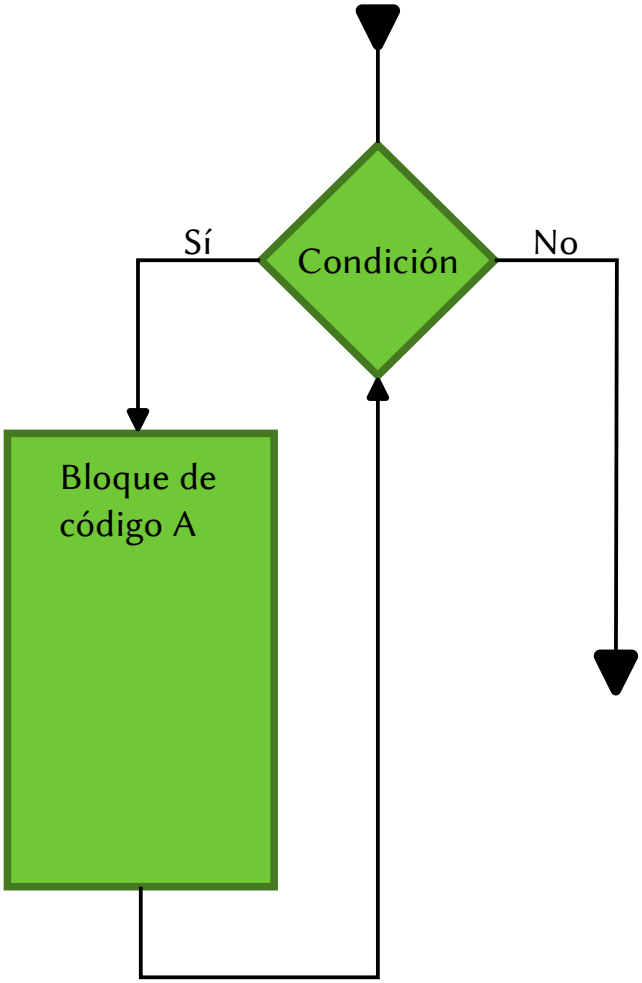
Otra variante consiste en **anidar ifs**:

```
if (edad < 12) {
    System.out.println("Eres muy jóven para la peli.");
} else if (edad >= 12 && edad < 18) {
    System.out.println("Esta peli es para ti.");
} else {
    System.out.println("Eres muy viejo para la peli.");
}
```

`switch`

Estructuras repetitivas (estructuras iterativas o bucles)

Estas estructuras determinan cuantas veces ha de repetirse un bloque de código. Puede indicarse un número fijo de repeticiones, como es el caso del `for` o establecer una condición para la repetición como es el caso del `while` y el `do while`.



`for`

El bucle `for` tiene la siguiente sintaxis:

```
for (<sentencia>; <condición>; <modificación de la condición>) {
    // Bloque de código a repetir
}
```

así, un bucle **for** que muestre los números del 1 al 100 sertía:

```
for (int i = 1; i <= 100; i += 1) {
    System.out.println(i);
}
```

`while`

El bucle `while` tiene la siguiente sintaxis:

```
while (<condición>) {
    // Bloque a repetir mientras se cumpla la condición.
}
```

El mismo código que muestre los números del 1 al 100 con un bucle **while** sería:

```java
int i = 1;
while (i <= 100) {
    System.out.println(i);
    i = i + 1;
}
```

`do while`

La única diferencia con los bucles anteriores es que en este **siempre se ejecuta el bloque de código** al menso **una vez**:

Su sintaxis es la siguiente:

```java
do {
    // Bloque a repetir
} while (<condición>);
```

El mismo ejemplo de los casos anteriores con un bucle **do while** sería:

```java
int i = 0;

do {
    System.out.println(++i);
} while (i < 100)
```

Continuará...