

Introducción a Java

Nivelación



Introducción a Java

La programación es el proceso mediante el cual podemos **diseñar y desarrollar un conjunto de instrucciones** que permiten realizar diferentes tareas o resolver problemas de manera automática, a través de la ejecución de un programa en un ordenador.

En términos más técnicos, la programación consiste en escribir una serie de instrucciones, utilizando un lenguaje de programación, cuya finalidad es resolver un problema concreto. A esto se le conoce como **algoritmo**. Estas instrucciones deberán estar escritas en un lenguaje que el ordenador sea capaz de entender e interpretar. Para ello, se utilizan los **lenguajes de programación**.

Existen diferentes conceptos de lenguaje de programación, cada uno con su sintaxis y reglas, que han de seguirse para escribir un código ejecutable. Por ejemplo, el **lenguaje ensamblador** trabaja en un entorno muy cercano a las instrucciones que maneja la CPU del ordenador. Esto hace que su legibilidad para nosotros sea algo más compleja. Por otra parte, existen **lenguajes de programación interpretados** que son más comprensibles para los humanos, pero requieren de ciertos mecanismos para que el ordenador sea capaz de entenderlos.

Nosotros, en este fastbook, nos centraremos en el **lenguaje de programación Java** con la idea de que puedas dar tus primeros pasos en este 'idioma' una vez finalices su estudio. Exploraremos sus fundamentos, estructuras, variables, bucles..., en definitiva, todos los imprescindibles para iniciar ese viaje en el que te convertirás en un desarrollador profesional.

Autor: Antonio Marchante

 **Introducción a Java**

 **Configuración del entorno de desarrollo**

 **Fundamentos de Java**

 **Bucles**

 **Funciones**

 **Qué hemos aprendido**

Introducción a Java



Java es un **lenguaje de programación de alto nivel**, orientado a objetos, que fue desarrollado por Sun Microsystems en la década de 1990. Es uno de los lenguajes de programación más populares y utilizados en todo el mundo.

Una de las características más destacadas de Java es su **portabilidad**. Esto significa que los programas escritos en Java pueden ejecutarse en diferentes sistemas operativos, como Windows, macOS y Linux, siempre y cuando se tenga instalada una máquina virtual de Java (JVM). La máquina virtual interpreta el código Java y lo ejecuta en el sistema operativo, haciendo que el código no sea dependiente de este.

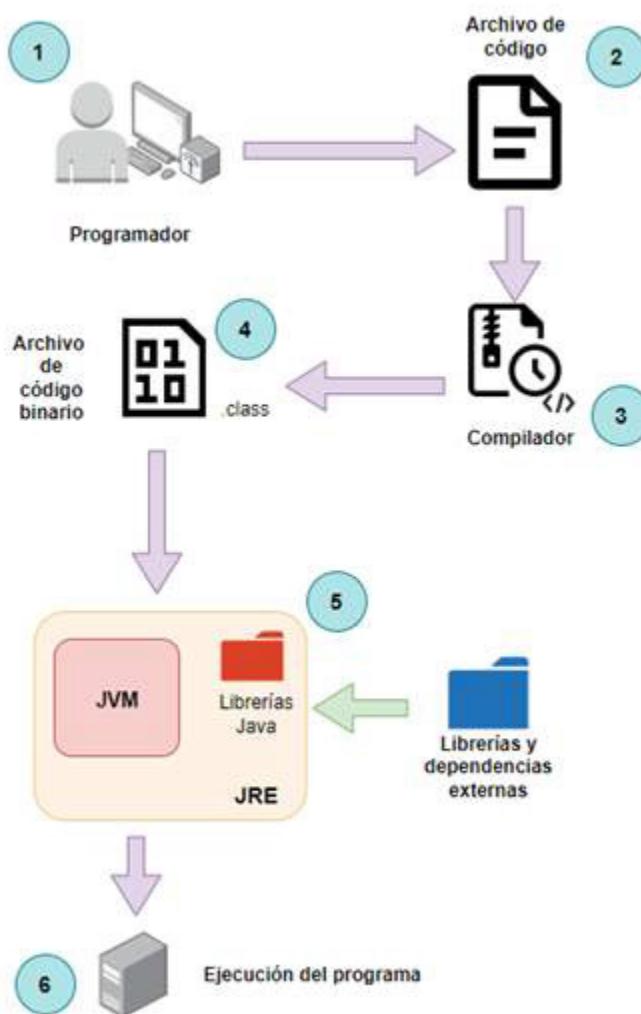
Java se utiliza en una amplia variedad de aplicaciones, desde el desarrollo de aplicaciones de escritorio y móviles hasta la creación de aplicaciones web. También se utiliza en la programación de sistemas embebidos y en la implementación de soluciones Big Data e inteligencia artificial.

Se caracteriza por ser un lenguaje seguro y robusto.
Incorpora características que permiten prevenir errores de programación comunes y ofrece mecanismos para el manejo de excepciones y la gestión de memoria.

Además, Java cuenta con una gran cantidad de bibliotecas y frameworks que facilitan el desarrollo de aplicaciones, como **JavaFX** para crear interfaces gráficas, **Spring** para el desarrollo de aplicaciones e **Hibernate** para la comunicación con Bases de Datos.

Proceso de compilación e interpretación

El flujo de la ejecución de un programa Java desde que lo codifica el programador es el siguiente:



Los pasos a seguir serían los descritos a continuación:

- 1 El programador escribe el código fuente en el IDE.
- 2 El archivo con el código se guarda con la extensión `.java`.
- 3 El código se compila en formato *bytecode*.
- 4 Se genera el archivo `.class`, que contendrá un lenguaje de bajo nivel.
- 5 La JVM (Java Virtual Machine), que está contenida en el JRE, junto con las librerías propias de Java, cargará los archivos `.class`.
- 6 Se ejecuta el programa en el equipo.

Características y ventajas de Java

Vamos a revisar ahora cuáles son las principales características y ventajas de Java con respecto a otros lenguajes.

- Portabilidad.** Java es un lenguaje de programación que funciona en diferentes sistemas operativos gracias a la JVM.
- Orientado a objetos.** Es un paradigma que facilita la modularidad y reutilización de código, lo que hace que los desarrollos sean más eficientes y mantenibles.
- Seguridad.** La JVM proporciona un entorno seguro que protege al sistema operativo de posibles problemas. Gracias, por ejemplo, al manejo de excepciones, la verificación de tipos y la gestión de la memoria.

- Multitarea.** Java ofrece la posibilidad de trabajar con varios hilos de ejecución a la vez. Esto permite que los programas Java sean más eficientes, ya que pueden realizar tareas en paralelo y acortar así los tiempos de ejecución y respuesta.
- Librerías propias.** Al contar con tantos años desde su lanzamiento, Java ha evolucionado y cuenta con gran cantidad de librerías (API) que facilitan la realización de tareas comunes como manipular archivos, comunicación por internet, etc.
- Comunidad.** Es uno de los lenguajes más usados en todo el mundo. Esto hace que tenga una gran cantidad de documentación y comunidad que pueden ayudarnos a solucionar problemas concretos durante la elaboración de nuestro código.
- Escalabilidad.** Gracias a las JVM, podremos escalar aplicaciones fácilmente utilizando características como la gestión automática de la memoria y la administración de recursos. Esto hará que, con una simple configuración, nuestra aplicación pueda crecer siempre que el equipo que la aloja lo permita.

Configuración del entorno de desarrollo



Qualentum Lab

Antes de entrar en detalle con la sintaxis del lenguaje, primero tenemos que preparar lo que podríamos denominar (permítome este símil) nuestra 'mesa de trabajo' e instalarnos las herramientas que necesitamos utilizar. ¡Vamos allá!

Instalación de JDK

Considero que es importante que previo a proceder con la instalación entendamos las **diferencias entre [JDK](#) y [JRE](#)**.

JDK

JDK (Java Development Kit) es un conjunto completo de herramientas para desarrolladores que permite crear, compilar y depurar aplicaciones. El JDK incluye:

- **Compilador Java (`javac`)**: convierte el código fuente Java en bytecode que puede ser ejecutado por la máquina virtual Java (JVM).
- **Herramientas de depuración y diagnóstico**: ayudan a identificar errores y problemas en el código.
- **Javadoc**: genera documentación a partir de comentarios en el código fuente.
- **Bibliotecas de clases y APIs**: contiene las bibliotecas y clases necesarias para el desarrollo de aplicaciones Java.
- **Herramientas para la gestión de JAR y JMOD**: para empaquetar y distribuir aplicaciones y bibliotecas.

JRE

En cuanto a **JRE (Java Runtime Environment)**, denominamos así al conjunto de herramientas y bibliotecas que se necesitan para ejecutar las aplicaciones Java en una máquina. Incluye la máquina virtual Java (JVM) y las bibliotecas de clases necesarias para ejecutar el bytecode Java. El JRE permite a los usuarios ejecutar aplicaciones Java sin necesidad de tener el JDK instalado. Sin embargo, no proporciona las herramientas de desarrollo que ofrece el JDK.

En resumen, el JDK se enfoca en el desarrollo de aplicaciones Java, proporcionando herramientas y bibliotecas para compilar, depurar y generar código, mientras que el JRE se enfoca en la ejecución de aplicaciones Java, proporcionando la JVM y las bibliotecas necesarias para ejecutar el código compilado.

Para la parte práctica de este lab y gran parte de las asignaturas del curso, utilizaremos el JDK que se descarga directamente desde nuestro **IDE (IntelliJ)**. Pero es posible que, en algunos casos, debamos utilizar una versión concreta de Java para un proyecto concreto (normalmente, por temas de compatibilidad). Para ello, deberemos dirigirnos a la página de **Oracle**. Dentro de ella, elegiremos la versión y la *release* (actualizaciones) que queramos descargar:

The screenshot shows the Oracle Java SE Development Kit 17.0.7 download page. At the top, it says "Java SE Development Kit 17.0.7" and "This software is licensed under the Oracle No-Fee Terms and Conditions License." Below that is a table with three columns: "Product / File Description", "File Size", and "Download". The first row shows "Linux Arm 64 Compressed Archive" with a file size of "172.12 MB" and a download link: "https://download.oracle.com/java/17/archive/jdk-17.0.7_linux-aarch64_bin.tar.gz (sha256)".

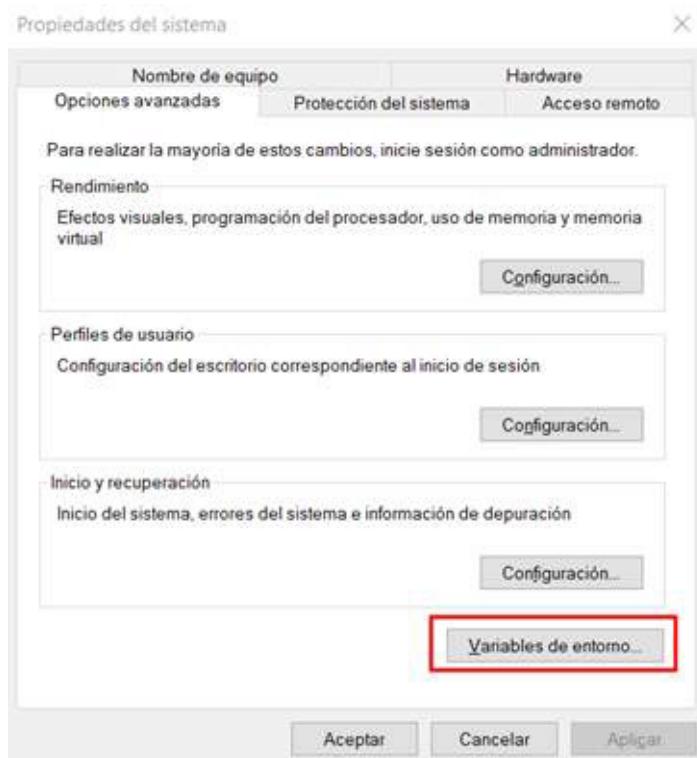
Fuente: Oracle.

Debemos tener en cuenta que lo subrayado en azul en la imagen hace referencia a la versión de Java, y lo subrayado en rojo a la *release*. Por norma general, una versión superior de Java es capaz de ejecutar sin problemas una versión anterior (es decir, la versión 17 sería capaz de ejecutar sin incidencias la versión 8).

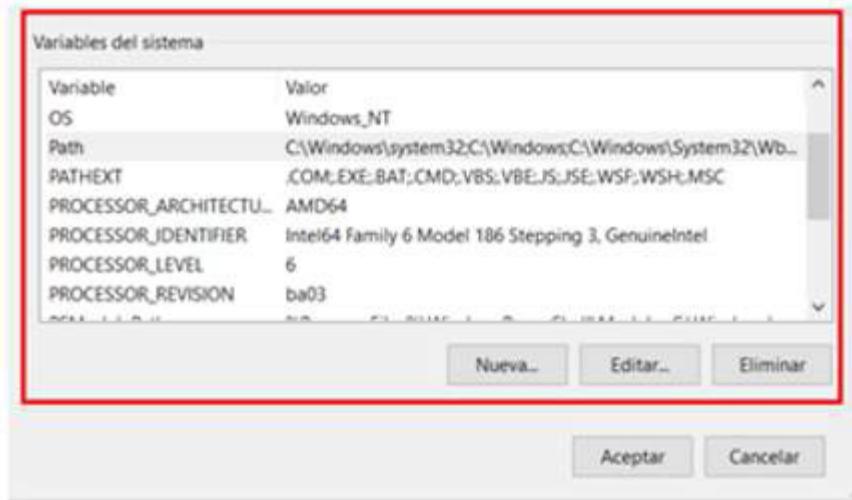
Configuración de variables de entorno

Para poder usar las funciones de Java desde la línea de comandos será necesario crear una variable de entorno a Windows (o el sistema operativo que utilicemos) para que este conozca la ubicación de la instalación de Java en el momento en que vayamos a referenciarla.

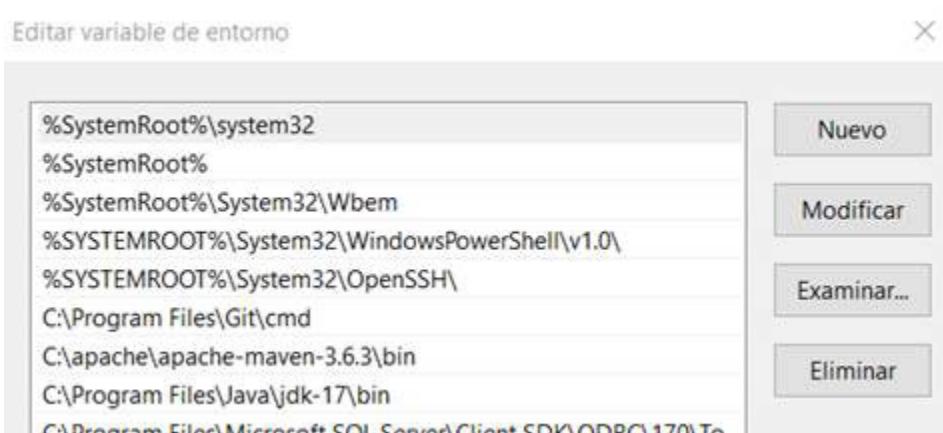
Con esta finalidad, debemos dirigirnos al buscador de Windows, escribir 'variables de entorno' y se nos abrirá la ventana que muestra la imagen.



Dentro de las variables del sistema, pulsaremos sobre 'Path' y clicaremos en 'Editar'.



Aquí deberemos añadir la ruta donde tengamos instalado Java, tal y como nos muestra la siguiente imagen.



Cómo compilar y ejecutar un programa Java desde la línea de comandos

Para compilar y ejecutar un programa desde la línea de comandos, debemos seguir el proceso que describimos a continuación.

Si tenemos el archivo "Main.java" con el siguiente código...

```
class Main {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

... Para poder ejecutarlo deberemos escribir este comando:

javac "Main.java"

Esto nos generará el archivo .class



Ahora bien, para poder ejecutarlo, simplemente tendremos que escribir lo siguiente:

java HelloWorld

Esto ejecutará el archivo `.class` en la JVM y mostrará con consola el resultado de dicha ejecución (fíjate en la imagen).



A screenshot of a terminal window. The command `>java Main.java` is entered at the prompt. The output shows the text "Hello, World!".

Instalación del IDE

Un **IDE (Integrated Development Environment)** es un programa que proporciona un conjunto de herramientas que facilitan la tarea del programador. Veamos ahora cuáles y para qué:



Editor de código

Un editor de texto en el que podremos escribir nuestros programas con facilidades como el resaltado de sintaxis, autocompletado, *indentación* automática y verificación de errores.



Compilación y ejecución

Nos permite compilar y ejecutar el código desde el propio IDE, haciéndonos ahorrar tiempo y evitándonos hacerlo desde la línea de comandos.



Depuración del código

Podremos depurar el código, crear puntos de ruptura y seguir la ejecución de un programa a fin de poder detectar posibles errores.

En nuestra formación en Java, utilizaremos el IDE IntelliJ de **JetBrains**, una empresa que tiene IDE's diferentes en función del lenguaje de programación que se vaya a usar. En nuestro caso, la herramienta que se facilita para el desarrollo de aplicaciones Java es **IntelliJ**, y podremos descargarla desde el siguiente [enlace](#).

Existen dos versiones de este IDE. Una versión Ultimate y una Community. En nuestro caso, **vamos a descargar la Community** ya que, pese a ser gratuita, cuenta con grandes funcionalidades y herramientas.

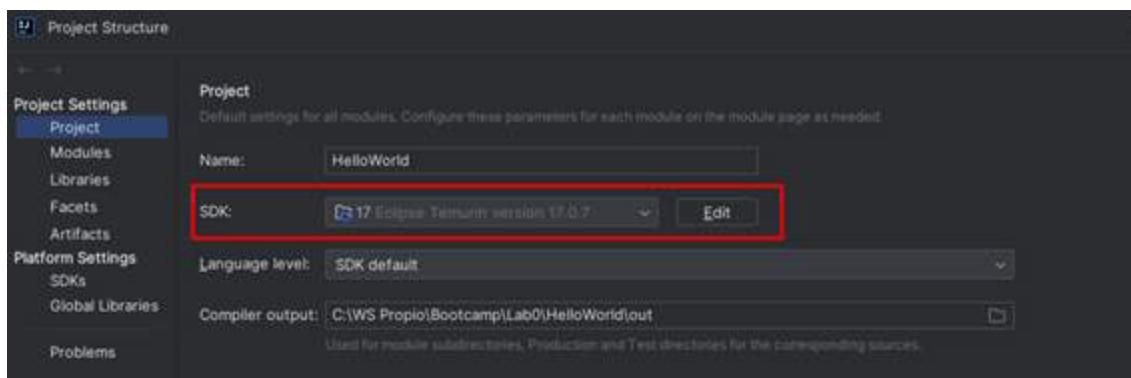


Fuente: JetBrains

- **Cómo ejecutar un programa Java desde el IDE**

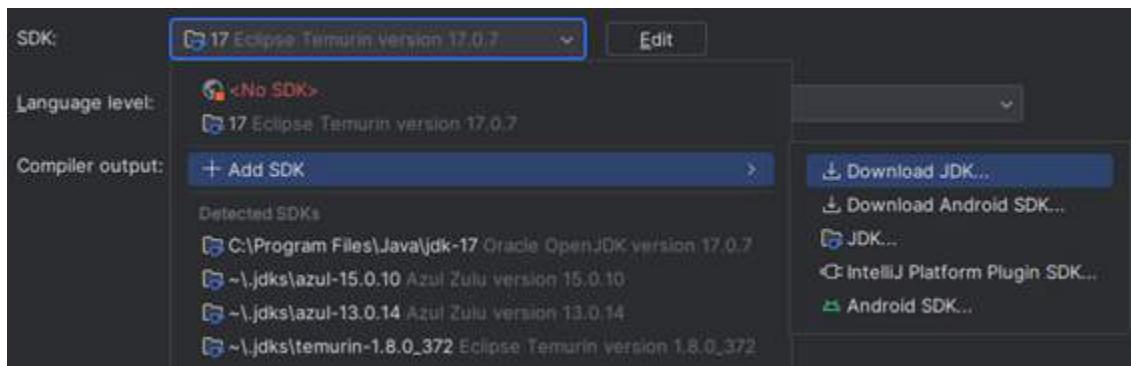
Después de instalar IntelliJ y crear un proyecto nuevo (esto nos creará un archivo Java con el famoso "*Hello World*"), deberemos seleccionar el JDK que utilizaremos.

Para ello, nos dirigiremos a *File > Project Structure*, y nos aparecerá la siguiente ventana:

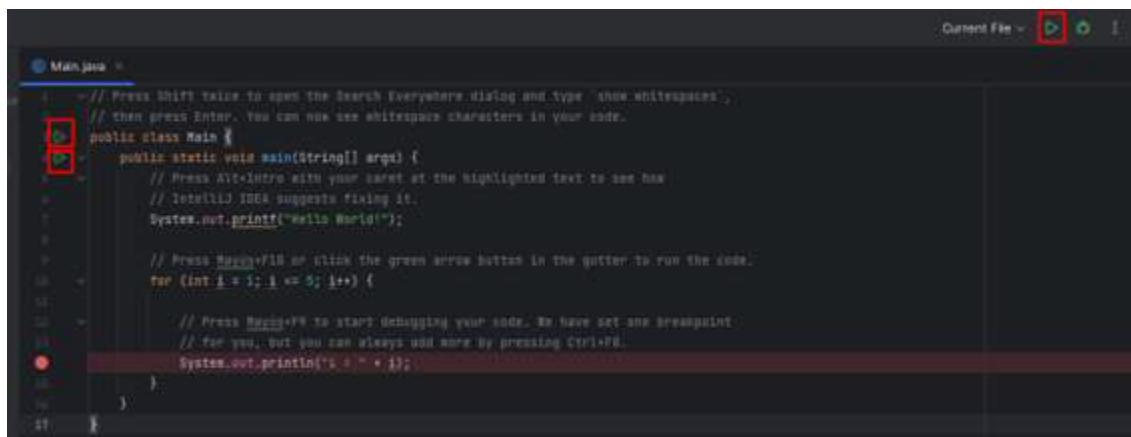


En la imagen anterior, en la ventana, podemos observar la versión de JDK que estamos utilizando.

Cabe destacar también que IntelliJ facilita la descarga de nuevos JDK. Si pulsamos sobre el desplegable podremos elegir entre los JDK instalados en el equipo o descargar uno nuevo si lo necesitamos.



Una vez definido nuestro JDK, simplemente deberemos pulsar sobre cualquiera de los botones *Play* que veremos tanto en el IDE como en el editor de texto.



Al hacerlo ¡atención!, se lanzará la aplicación y, por último, nos mostrará su ejecución por el terminal.

```
C:\Users\amarchante\.jdks\temurin-17.0.7\bin\java.exe "-javaagent:  
Hello World!i = 1  
i = 2  
i = 3  
i = 4  
i = 5  
  
Process finished with exit code 0
```

Fundamentos de Java



Ahora vamos a estudiar cuáles son los principios del lenguaje que nos toca tratar y que, estoy seguro, te resultará apasionante. Si es tu primera vez programando, quizás te asuste en una primera toma de contacto, o te parezca complicado... Pero créeme, no es muy diferente a estudiar un idioma o, tal vez, el mejor ejemplo son las matemáticas: requiere práctica, hacer y rehacer muchos ejercicios, consultar la documentación y... seguir practicando.

Variables, constantes y tipos de datos

Una variable o constante es un dato que se almacenará en memoria y se puede consultar a lo largo de la ejecución del programa. Para poder definir una variable o constante, deberemos especificar el tipo de dato que contendrá, y un nombre para poder referenciarla más adelante en el código.

La diferencia entre variable y constante es que la variable puede ver alterado su valor a lo largo del programa, mientras que la constante tendrá un valor fijo.

Para poder definir una constante solo deberemos añadir la palabra reservada **final**.

Aquí comparto un ejemplo:

```
Integer age = 18;  
final String bloodType = "AB+";
```

¡Importante! Como podemos ver, para que el programa en Java compile correctamente, debemos acabar cada línea con un ; (punto y coma). Esto delimitará cada instrucción.

Los tipos de datos que se pueden usar en Java tendrán que ver con el tipo de valor que contendrán. A su vez, dentro de un mismo tipo de dato, existen diferencias en función a la **precisión de los datos** que debemos almacenar. Esto es algo importante ya que, si intentamos almacenar un valor más grande de lo que es capaz de albergar, se producirá un error.

Pese a esto, se ha de tener en cuenta que un uso de tipo de dato que tenga una precisión mayor requerirá un mayor uso de la memoria del sistema donde se vaya a ejecutar el programa. Por este motivo, debemos ser previsores en cuanto al tamaño de datos que necesitaremos almacenar en cada variable y elegir, con criterio, el tamaño que más se adecúe a cada caso.

i Curiosidad: El problema del año 2038 es un problema derivado de este desbordamiento de los datos. En el sistema UNIX, se almacena la fecha contando los números de segundos desde la noche del 1 de enero de 1970 (un estándar dentro de la programación como POSIX). Por este problema, se prevé un nuevo efecto 2000, por el cual llegado el día 19 de enero de 2038, se producirá un desbordamiento en este valor (de 32 bits) y hará fallar todos los sistemas UNIX que no hayan realizado el cambio a un tipo de dato que contemple una precisión mayor.

1

Números enteros

Son un tipo de dato que almacena un número entero. En función de la longitud de este, deberemos usar una u otra de las siguientes:

Tipo de dato	Rango numérico
byte	-128 a 127
short	-32.768 a 32.767
int	-2^{31} a $2^{31}-1$
long	-2^{63} a $2^{63}-1$

Por ejemplo:

```
int age = 24;
```

2

Números decimales

Los números decimales, al igual que los enteros, pueden representarse en diferentes tipos de datos. Los más comunes son los siguientes:

Tipo de dato	Rango numérico	Rango numérico
float	1.4×10^{-45} a 3.4028235×10^{38}	aprox. 7 números decimales
double	4.9×10^{-324} a $1.7976931348623157 \times 10^{308}$	aprox. 15 números decimales

Aquí un ejemplo:

```
double temperature = 25.6;
```

3

Alfanumérico

String se utiliza para almacenar cadenas de caracteres que pueden contener números, letras, símbolos, etc.

Por ejemplo:

```
String name = "Juan";
```

La forma de concatenar dos o más Strings puede hacerse a través del operador `+`, del siguiente modo:

```
String name = "John" + " Doe";
String name2 = "Jane";
name2 += "Doe";
```

La forma de comparar Strings, al contrario que con datos de tipo numérico, no se hace por medio del operador `==`, si no por medio de una función propia de los Strings, la llamada `.equals`. Vamos a verlo en un ejemplo.

```
"John".equals("Jon") --> false
"John".equals("John") --> true
```

4

Tipo lógico

Boolean es un tipo de dato que solo puede tomar dos valores: `true` o `false` (verdadero o falso). Este tipo de campo puede utilizarse para determinar una condición, ya que toma un valor lógico binario (1 o 0), y determina si una condición o campo es verdadero o falso.

Por ejemplo:

```
boolean isStudent = true;
```

Estructuras de datos

1

Arrays

Los arrays o **vectores** son colecciones de datos de un mismo tipo. Dentro del array, cada elemento se identificará con una posición dada por un índice numérico entero. Este índice empezará en 0.

Su definición es la siguiente:

```
int [] grades = new int[5];
```

Como podemos observar, primero definimos el tipo de dato, seguido de los corchetes **[]** que identifican al array. Esta variable deberemos inicializarla definiendo de nuevo el tipo de dato y el tamaño total del array. En nuestro caso, 5.

Para poder asignar un valor a una posición concreta del array, deberemos indicar el índice de la siguiente manera:

```
grades [2] = 7;
```

Y para poder acceder a una dirección concreta, también deberemos indicar su índice:

```
int value = grades [1];
```

Y una cosa más, también será posible inicializar el array a la hora de declararlo. Lo haremos de la siguiente forma:

```
int [] ordinals = [1, 2, 3, 4];
```

2

Arrays bidimensionales

Es posible definir arrays bidimensionales (conocidos también como *matrices*). Estos son útiles cuando queremos organizar elementos en cuadrículas o tablas. Para esto deberemos definir el tamaño de ambas dimensiones, diferenciándose entre filas y columnas.

Este es un ejemplo:

```
int [][] matrix = new int [4] [3];
```

Este array bidimensional será una tabla con 4 filas y 3 columnas semejante a la siguiente:

Para acceder al valor de una posición concreta del array bidimensional no solo deberemos indicar su índice de fila, sino también el de columna.

```
matrix [2] [1] = 4;  
int value2 = matrix [1] [0];
```

Al igual que en el caso de los arrays unidimensionales, también será posible inicializarlos en el momento de declararlos:

```
int[][] matrix = {  
    {1, 2, 3},  
    {4, 5, 6},  
    {7, 8, 9}};
```

Lo que nos generará la siguiente estructura:

1	2	3
4	5	6
7	8	9

Operadores

1

Operadores aritméticos

Operación	Sintaxis	Ejemplo	Resultado	Explicación
Suma	+	$5 + 5$	10	Se realiza la suma entre los dos números.
Resta	-	$5 - 3$	2	Se realiza la resta entre los dos números.
Producto	*	$5 * 5$	25	Se realiza la multiplicación entre los dos números.
División	/	$12 / 3$	4	Se realiza la división entre los dos números.
Módulo (resto de la división)	%	$5 \% 2$	1	Será el resto de la división entre los dos números.

2

Operadores de asignación

Operación	Sintaxis	Ejemplo	Resultado	Explicación
Asignación simple	=	Int number =5;	5	Se asignará 5 a la variable <i>number</i> .
Asignación con suma	+=	number += 3;	8	Se sumará 3 a la variable <i>number</i> .
Asignación con resta	-=	number -= 2;	3	Se restará 2 a la variable <i>number</i> .
Asignación con multiplicación	*=	number *= 2;	10	Se multiplicará 2 por la variable <i>number</i> .
Asignación con división	/=	number /= 2;	2	Se dividirá la variable <i>number</i> entre 2.
Asignación con módulo	%=	Number %= 2;	1	Será el resto de la división entre la variable <i>number</i> entre 2.

3

Operadores de comparación

El resultado de estas operaciones siempre será de tipo boolean.

Operación	Sintaxis	Ejemplo	Resultado	Explicación
Igual a	<code>==</code>	<code>5 == 3</code>	false	Realiza una comparación para determinar si dos valores son iguales.
No igual a	<code>!=</code>	<code>5 != 3</code>	true	Realiza una comparación para determinar si dos valores son diferentes.
Mayor que	<code>></code>	<code>4 > 3</code>	true	Compara si el primer valor es mayor al segundo.
Menor que	<code><</code>	<code>5 < 2</code>	false	Compara si el primer valor es menor al segundo.
Mayor o igual que	<code>>=</code>	<code>5 >= 2</code>	true	Compara si el primer valor es mayor o igual al segundo.
Menor o igual que	<code><=</code>	<code>4 <= 3</code>	false	Compara si el primer valor es igual o menor al segundo.

Recuerda: el resultado de estas operaciones siempre será de tipo boolean.

Operación	Sintaxis	Ejemplo	Resultado	Explicación
AND	&&	$(5 > 3) \&\& (4 < 6)$	true	Es el resultado lógico de comparar dos valores booleanos. Por un lado, 5 es mayor que 3 (true). Por otro, 4 es menor que 6 (true).
OR		$(4 < 3) (4 > 3)$	true	Será el resultado de una puerta lógica OR. Siempre y cuando uno de los dos valores sea true (o ambos), el resultado será true. Si ambos son false, será false.
NOT	!	$!(4 > 3)$	false	Negará el valor booleano de la operación. Es decir, convertirá un valor true en false y viceversa.

5

Comparación entre booleans

La comparación entre booleans se realiza de acuerdo con las reglas de las comparaciones lógicas. Es decir, la misma que se da en las puertas lógicas en electrónica.

Valor 1	Operador	Valor 2	Resultado
true	&&	true	true
true	&&	false	false
false	&&	false	false
true		true	true
true		false	true
false		false	false
-	!	false	true
-	!	true	false

6

Operadores de incremento y decremento

Operación	Sintaxis	Ejemplo	Resultado	Explicación
Incremento	++	5++	6	Sumará uno al número indicado
Decremento	--	5--	4	Restará uno al número indicado

Estructuras de control (if-else, switch)

1

If

La estructura de *if* será un sub-bloque de código que albergará un código que solo se ejecutará en caso de que la condición definida en el *if* sea verdadera.

Su estructura es la siguiente:

```
if ( expression ) {
    // Code to execute in case that expression it's true
}
```

Por ejemplo:

```
if ( 4 > 3) {  
    System.out.println("4 it's greater than 3");  
}
```

2

If - Else

En caso de que queramos controlar si debemos ejecutar un código u otro en base a una condición, podremos utilizar la sentencia **else**.

Su estructura es la siguiente:

```
if ( expression ) {  
    // Code to execute in case that expression it's true  
}  
else {  
    // Code to execute in case that expression it's false  
}
```

Aquí lo vemos con un ejemplo:

```
if ( 4 > 3) {  
    System.out.println("4 it's greater than 3");  
}  
else {  
    System.out.println("4 isn't greater than 3");  
}
```

3

If - Else If - Else

También tendremos la opción de añadir más de una condición a controlar por medio de la sentencia `else if`. Esto hará que la sentencia `else` solo se ejecute si ninguna de las dos anteriores da como resultado `true`.

Su estructura es la siguiente:

```
if ( expression ) {
    // Code to execute in case that expression it's true
}
else if (expression2) {
    // Code to execute in case that expression2 it's true
}
else {
    // Code to execute in case that all expressions are false
}
```

De nuevo, lo vemos con un ejemplo:

```
if ( 4 > 3) {
    System.out.println("4 it's greater than 3");
}
else if (4 < 3) {
    System.out.println("4 isn't greater than 3");
}
else {
    System.out.println("I don't even know maths");
}
```

También existirá la posibilidad de que **un propio if contenga otra estructura if**:

```
if ( 4 > 3) {  
    System.out.println("4 it's greater than 3");  
  
    if ( 5 > 3) {  
        System.out.println("5 it's, also, greater than 3");  
    }  
}
```

4

Switch

Switch es una estructura de control que permite elegir entre diferentes bloques de código en función a una expresión a evaluar. Es una alternativa a la utilización de varias sentencias *if* para comprobar un valor común.

Su estructura es la siguiente:

```
switch (expresion) {  
    case value1:  
        // Code to execute for value1  
        break;  
    case value2:  
        // Code to execute for value2  
        break;  
    default:  
        // Code to execute if any of the above conditions it's true  
}
```

En esta estructura podemos destacar las siguientes palabras:

- switch** que será la que inicialice el bloque de código.
- case** que definirá el valor de la **expression** (en caso de que **expression == value1**, ejecutará el código de dicho **case**).
- break**, la que se utilizará para salir del *switch* y continuar con la ejecución del programa.
- default**, ya que, en el caso de que ningún valor definido coincida con el de **expression**, se ejecutará el código contenido en **default**.

Lo vemos en este ejemplo:

```
int dayOfWeek = 3;

switch (dayOfWeek) {
    case 1:
        System.out.println("Monday");
        break;
    case 2:
        System.out.println("Tuesday");
        break;
    case 3:
        System.out.println("Wednesday");
        break;
    case 4:
        System.out.println("Thursday");
        break;
    case 5:
        System.out.println("Friday");
        break;
    default:
        System.out.println("Weekend");
}
```

Bucles



Qualentum Lab

Un bucle es una estructura dentro del código, que hará que se repita un determinado bloque de código hasta que se cumpla cierta condición que rompa el bucle, y continúe con la ejecución del resto del código.

Bucle **for**

La estructura **for** es un bucle que permite repetir un bloque de código un número específico de veces. Este control del número de veces vendrá dado por un **contador** que se definirá a la hora de declarar el bucle.

Es muy útil para recorrer listas o colecciones de objetos.

Su sintaxis es la siguiente:

```
for (initialization; expression; increase) {  
    // Code to execute  
}
```

Veámoslo con un ejemplo. Si quisieramos inicializar un array con los números ordinales, podríamos utilizar el siguiente código:

```
int[] array = new int[5];
for (int i = 0; i < 5; i++) {
    array[i] = i;
System.out.println("Value in position " + i + " of the array is:
+ array[i]);
}
```

Bucle while

El bucle *while* es una estructura iterativa que repite un bloque de código siempre y cuando la condición sea verdadera. En el momento en que no lo sea, saldrá del bucle.

Su sintaxis es la siguiente:

```
while (expression) {
    // Code to execute while expression it's true
}
```

¡CUIDADO! Es importante que a lo largo de la ejecución del bucle While la condición pase a ser falsa. De lo contrario, el bucle se ejecutará indefinidamente y el programa no terminará.

Fíjate en el ejemplo:

```
int counter = 1;
while (counter <= 5) {
    System.out.println("Counter: " + counter);
    counter++;
}
```

Bucle do-while

El bucle do-while es una estructura similar al while, con la diferencia de que el código del bucle se ejecutará una vez antes de evaluar la expresión.

Esto es útil cuando necesitamos ejecutar un bloque de código y, en base al resultado de este, tengamos que decidir si seguirá ejecutándose o, por el contrario, continuará con la ejecución del programa.

Su sintaxis es la siguiente:

```
do {
    // Code to execute at least one time
} while (expression);
```

Aquí un ejemplo:

```
int counter = 0;
do {
    System.out.println("Counter: " + counter);
    counter++;
} while (counter < 5 && counter > 0);
```

En este caso, de utilizar la estructura while en lugar de la do-while, nunca ejecutaría el código.

Bucles anidados

En ciertas ocasiones necesitaremos utilizar una anidación de bucles para poder recorrer ciertas estructuras de datos.

Por ejemplo, si precisamos recorrer todas las posiciones de un array bidimensional, necesitaremos utilizar **la anidación de dos bucles for** de la siguiente forma:

```
int[][] matrix = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}};

int rowSize = 3;
int columnSize = 3;

for (int row = 0; row < rowSize; row++) {
    for (int column = 0; column < columnSize; column++) {
        System.out.print(matrix[row][column] + " ");
    }
    System.out.println(); // Line break after each row
}
```

Funciones



Qualentum Lab

Una función o método es un bloque de código que debería tener una tarea específica.

Estas funciones se pueden consultar desde otras partes del código, lo cual los hace reutilizables a lo largo de un programa y mantiene más legible el código de este.

Declaración y llamada de funciones

La sintaxis básica de una función será la siguiente:

```
returnType methodName(parameters) {  
    // Code of the method  
}
```

Y aquí te comparto un ejemplo de declaración de función:

```
void sayHello() {  
    System.out.println("Hello!");  
}
```

Por cierto, para poder invocar una función desde otra parte del código, simplemente tendremos que escribir su nombre de la siguiente forma:

```
sayHello();
```

Parámetros de entrada y retornos

Las funciones pueden tener parámetros de entrada y parámetros de salida. Esto es debido a que existirán funciones que necesitarán datos previos para poder realizar una cierta operación, y funciones que deberán devolver el resultado de una operación para poder ser usado en otra parte de la aplicación.

Si una función no devuelve ningún valor, se definirá el tipo *void*, como en el ejemplo de ***sayHello()***.

En caso de que se necesiten parámetros para usarse en la función, estos se indicarán tanto en la propia función como a la hora de invocarla desde el código.

```
void sayHelloToUser(String username) {  
    System.out.println("Hello, " + username);  
}
```

Para poder llamar a esta función desde el código, lo haremos de la siguiente forma:

```
sayHelloToUser("John");
```

Por otro lado, podremos necesitar que una función devuelva un valor para usarlo más adelante en el código. En este caso, deberemos indicar el tipo de dato que retornará la función en lugar de *void*, fíjate en es ejemplo!

```
int addNumbers(int firstNumber, int secondNumber) {  
    return firstNumber + secondNumber;  
}
```

Por último, para utilizar esta función desde el código, y poder usar su resultado, lo haremos de la siguiente forma:

```
int addition = addNumbers(4, 2);  
System.out.println("The result of the addition is: " + addition);
```

Qué hemos aprendido



Qualentum Lab

A lo largo de este fastbook, hemos establecido las bases de tu aprendizaje sobre Java.

Hemos visto en qué consiste este lenguaje, cuáles son sus ventajas y cuáles son sus cimientos. También ya sabes cómo puedes instalarlo y empezar a codificar el propio código en el entorno correcto. Varios puntos que hemos tratado y que debes repasar son el de los tipos de objetos que se utilizan durante la ejecución de un programa básico y el de las estructuras condicionales e iterativas para la realización de algoritmos.

Ya dispones de todos los *ingredientes* y *las recetas básicas* para empezar a practicar hasta que manejes este lenguaje con soltura.

¡Enhорabuena! Fastbook superado



Qualentum.com