¿Qué son los Booleanos?

Booleano (boolean) es un tipo de dato en Java que almacena sólo dos posibles valores lógicos: TRUE o FALSE (cierto o falso)

Podemos declarar variables o constantes igual que si fueran Integer, String, etc.

```
Boolean nombreVariable;
```

Podemos darle un valor directo a la variable (true/false).

```
Boolean nombreVariable = true;
```

O podemos escribir una expresión que devuelva un valor booleano.

```
Boolean nombreVariable = 6 > 3;
```

¿Qué expresiones devuelven un valor Booleano?

Son expresiones normalmente comparativas (igual, distinto, mayor qué, menor qué, etc.)

Podemos construir estas expresiones utilizando variables o valores fijos.

```
nombreVariable = numero > 3;
```

También podemos utilizar operadores lógicos para:

- Que la condición sea falsa (negación)
- Que dos condiciones o más tengan que ser ciertas a la vez (and)
- Que de dos condiciones o más, al menos una sea cierta (or)

Estos son los principales operadores:

OPERADOR	DESCRIPCIÓN
==	Es igual
!=	Es distinto
<, <=, >, >=	Menor, menor o igual, mayor, mayor o igual
&&	Operador and (y)
||	Operador or (o)
!	Operador not (no)

Operadores lógicos principales en Java

Abel Morillo Sánchez
Belén Tudó Ramírez

¿Cómo comparamos cadenas (String)?

Para saber si dos cadenas son iguales debemos usar "equals". Por ejemplo, para ver si la cadena1 es igual a la cadena2, sería así:

```
Boolean nombreVariable = cadena1.equals(cadena2);
```

¿Cómo podemos incluir una condición en nuestro programa para hacer una cosa u otra?

Lo podemos hacer con una estructura IF. Lo escribimos de este modo:

```
// Instrucciones 0
if (expresionCondicional1) {
    // Instrucciones A
}
else if (expresionCondicional2) {
    // Instrucciones B
}
else if (expresionCondicional3) {
    // Instrucciones C
}
else {
    // Instrucciones D
}
// Instrucciones Final
```

1. Se ejecutaría en primer lugar las "Instrucciones 0" (lo que esté antes de IF)
2. Luego se evalúa la expresión condicional 1. Si es TRUE, se ejecuta InstruccionesA. Y luego se ejecutaría Instruccciones Final.
3. Si expresión condicional 1 es FALSE, se evalúa expresión condicional 2. Si es TRUE, se ejecuta InstruccionesB. Y luego se ejecutaría Instruccciones Final.
4. Si expresión condicional 2 es FALSE, se evalúa expresión condicional 3. Si es TRUE, se ejecuta InstruccionesC. Y luego se ejecutaría Instruccciones Final.
5. Si expresión condicional 3 es FALSE, se ejecuta InstruccionesD. Y luego se ejecutaría Instruccciones Final.

No es obligatorio incluir el "else" final. Ni tampoco los "else if" adicionales.

Hay múltiples explicaciones y ejemplos en la red. Por ejemplo, en inglés está bastante bien descrito aquí https://www.w3schools.com/java/java_conditions.asp

Los condicionales también se pueden escribir en una sola línea, siempre que se trate de devolver o actualizar algún resultado en una variable. Es algo que no se usa habitualmente, pero puede que algún día lo necesitemos o que lo veamos escrito. Veamos un ejemplo:

El siguiente código:

```
Integer numero = 7;

String mensaje;
if (numero < 10) {
    mensaje = "Número menor a 10";
}
else {
    mensaje = "Numero mayor o igual a 10";
}
```

Es equivalente a este otro:

```
Integer numero = 7;

String mensaje = (numero < 10) ? "Número menor a 10" : "Numero mayor o igual a 10";
```

La interrogación se usa para preguntar si la condición es cierta. Lo que se pone a continuación es el valor devuelto en caso de que así sea. Los dos puntos ":" serían como "else", y lo que se pone a continuación sería el valor devuelto en caso de que la condición sea falsa.

¿Cómo podemos repetir instrucciones mientras una condición sea cierta?

Lo podemos hacer con una estructura WHILE. Lo escribimos de este modo:

```
// Instrucciones 0

while (expresionCondicional) {
    // Instrucciones A
}

// Instrucciones Final
```

1. Se ejecuta en primer lugar Instrucciones 0 (lo que está antes del while)
2. A continuación, se evalúa "expresionCondicional" Si es TRUE, se ejecuta "Instrucciones A"
3. A continuación, se repite el paso 2. Es decir, se vuelve a evaluar "expresionCondicional" Y si de nuevo es TRUE, se vuelve a ejecutar "Instrucciones A". Esto se repite de forma continua.
4. Cuando expresionCondicional es FALSE, el bucle se para. A continuación, se ejecutaría "Instrucciones Final"

Otra variación es DO-WHILE. Se escribe así:

```
// Instrucciones 0

do {
    // Instrucciones A
}
while (expresionCondicional);

// Instrucciones Final
```

1. Se ejecuta en primer lugar Instrucciones 0 (lo que está antes del do-while)
2. A continuación, se ejecuta "Instrucciones A" (da igual si la condición es TRUE O FALSE)
3. Luego se evalúa "expresionCondicional". Si es TRUE, se vuelve a ejecutar "Instrucciones A". Esto se repite continuamente.
4. Cuando expresionCondicional es FALSE, el bucle se para. A continuación, se ejecutaría "Instrucciones Final"

Operaciones habituales con bucles

Como hemos dicho, utilizamos un bucle para repetir una o varias instrucciones hasta que una condición deje de ser cierta. Dos ejemplos típicos para los que se usa con para hacer lo mismo un número de veces determinado y para acumular una operación.

PARA REPETIR N VECES LO MISMO

Para repetir un número de veces determinado las mismas instrucciones, usamos un bucle. Para controlar el número de veces podemos crear una variable a modo de contador que vamos incrementando. Veamos un ejemplo:

Supongamos que quiero escribir 7 veces la palabra "Hola"

```
Integer contador = 1; // creamos una variable contador en 1

while (contador <= 7) { // la condición será hasta que el contador sea 7
    System.out.println("Hola"); // imprimimos "hola"
    contador = contador + 1; // incrementamos el contador
}
```

SUMAR UNA LISTA DE VALORES

Para ello tendremos que crearnos una variable donde ir acumulando la suma. Imaginemos que queremos sumar 7 veces el número 10 y conocer el resultado. Se haría así:

```
Integer contador = 1; // creamos una variable contador en 1
Integer acumulado = 0; // creamos una variable donde iremos sumando

while (contador <= 7) { // la condición será hasta que el contador sea 7
    acumulado = acumulado + 10; // vamos sumando 10 al acumulado
    contador = contador + 1; // incrementamos el contador
}

System.out.println("Resultado = " + acumulado); // Imprimimos el resultado
```

¿Cómo podemos trabajar con cadenas?

Como sabemos, las cadenas son objetos de tipo String. Esta clase tiene un conjunto de métodos con diferentes utilidades. Los principales que debemos conocer son todos estos:

```
String cadena = "Abel Morillo Sanchez ";
String sinEspaciosFinalInicial = cadena.trim();
String enMayusculas = cadena.toUpperCase();
String enMinusculas = cadena.toLowerCase();
String repetidaTresVeces = cadena.repeat(3);
String desdeLaEmeEnAdelante = cadena.substring(5);
String desdeLaBeHastaLaErre = cadena.substring(1,7);
String cambiandoElesPorZetas = cadena.replaceAll("l", "z");

Integer longitud = cadena.length();
Integer posicionDeLaPrimeraEle = cadena.indexOf("l");
Integer posicionDeLaUltimaEle = cadena.lastIndexOf("l");
Integer posicionDeLaPrimeraEleDesdeLaI = cadena.indexOf("l", 8);

Boolean contieneMorillo = cadena.contains("Morillo");
Boolean sonIguales = cadena.equals("Ejemplo");
Boolean sonIgualesIgnorandoMinMay = cadena.equalsIgnoreCase("Ejemplo");
Boolean esCadenaVacia = cadena.isEmpty();
Boolean esCadenaVaciaSoloConEspacios = cadena.isBlank();
Boolean comienzaPorAb = cadena.startsWith("Ab");
Boolean terminaEnEz = cadena.endsWith("ez");

Integer comparando = cadena.compareTo("Ejemplo");
// si comparando es > 0 → cadena es mayor que “Ejemplo”
// si comparando es < 0 → cadena es menor que “Ejemplo”
// si comparando es = 0 → cadena es igual que “Ejemplo”
```

Además de estos métodos principales, podemos encontrarnos con otro muy utilizado:

charAt(posicion): devuelve el carácter de la cadena que está en la posición indicada. Si la posición fuera mayor o igual que la longitud de la cadena, generará un error.

```
String mensaje ="Hola Mundo";
System.out.println(mensaje.charAt(0)); //H
System.out.println(mensaje.charAt(5)); //M
```

El caso concreto de **compareTo**, cuando hablamos de que una cadena es mayor que otra, lo que queremos decir es que es alfabéticamente mayor. Es decir, si las ordeno alfabéticamente, la que es mayor estará en segundo lugar, y la que es menor, en primer lugar. En otras palabras, la palabra menor es la que me encuentro antes en el diccionario.

Cuando no nos acordemos cómo funcionan, podemos consultar el javadoc que nos aparece en EclipseIDE o la documentación de JAVA:

https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/lang/String.html

¿Hay alguna alternativa al bucle WHILE / DO-WHILE?

Existe una alternativa a los bucles while y do-while. Es el bucle FOR. Normalmente se utiliza siempre que queremos programar este conjunto de instrucciones:

1. Inicializar variables
2. Repetir algo mientras se cumpla una condición
3. En cada iteración, realizar al final siempre una operación

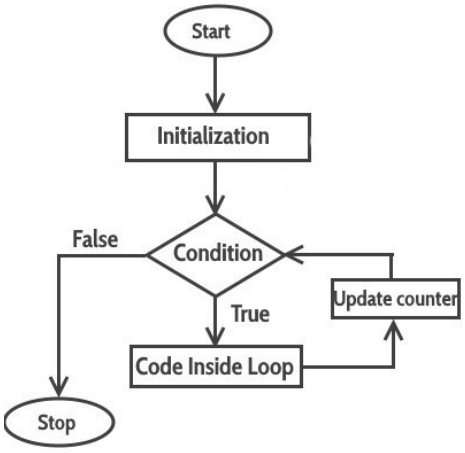
Para un bucle WHILE con esta estructura:

```
variables;
while(condicion){
    ... // sentencias
    instrucciones;
}
```

Podemos escribir más fácilmente un FOR así:

```
for(variables ; condicion ; instrucciones){
    ... // sentencias
}
```

Lo normal es usar los bucles FOR para implementar este tipo de algoritmos:



Lo que en un WHILE sería:

```
int i = 0;
while(i < 11) {
    System.out.println(i);
    i++;
}
```

En un FOR sería:

```
for(int i = 0; i < 11 ; i++){
    System.out.println(i);
}
```

En la inicialización del FOR sólo podemos poner una instrucción. Pero podemos meter la inicialización de más de una variable. Por ejemplo, lo que en un WHILE hacemos así:

```
int i =  0,
    j = 10;
while(i < 11){
    System.out.println(i + "," + j);
    i++;
    j--;
}
```

En un FOR lo podemos hacer así:

```
for(int i = 0, j = 10 ; i < 11 ; i++ , j--){
    System.out.println(i + "," + j);
}
```

Aunque normalmente en un FOR sólo se trabaja con una variable.

Tampoco es obligatorio rellenar las tres partes del FOR. Podemos escribir algo así:

```
for(; condicion ;){
    ... // sentencias
}
```

Pero no tiene mucho sentido porque es lo mismo que un WHILE así:

```
while(condicion){
    ... // sentencias
}
```

¿Cómo podemos obtener números aleatorios?

Podemos utilizar la clase Random. Tendremos primero que inicializar un objeto de esa clase así:

```
Random random = new Random();
```

Luego podemos pedirle aleatorios de cualquier tipo. Por ejemplo, para números enteros se haría de este modo:

```
Integer aleatorio = random.nextInt(6, 10);
```

Los dos números que indicamos son el intervalo entre los que tiene que estar el aleatorio generado. El primer número (el 6 en el ejemplo de arriba) sería el límite inferior. El segundo número (el 10 en el ejemplo de arriba) sería el límite superior **excluido**. Es decir, en el ejemplo de arriba estaríamos solicitando **aleatorios entre 6 y 9 ambos inclusive**.

¿Cómo podemos saber el resto de dividir un número entre otro? (Pares e impares)

Esta operación se llama módulo. Y se puede obtener con el operador %. Por ejemplo, para obtener el resto de dividir entre 2 un número, se haría así:

```
Integer numero = 21;
Integer resto = numero % 2;
```

Esto lo podemos utilizar para saber si un número es par o impar, por ejemplo. Cuando el resto de dividir un número entre 2 es cero significa que el número es par:

```
Integer numero = 21;
Integer resto = numero % 2;
if (resto == 0){
   System.out.println(“El número es par”);
}
else{
   System.out.println(“El número es impar”);
}
```

Del mismo modo lo podemos utilizar para saber si un número es múltiplo o divisible por otro.

¿Cómo podemos crear e invocar otros métodos para dividir nuestro código?

Dentro de la misma clase, puede que nos venga bien dividir el código en diferentes métodos en lugar de tenerlo todo junto dentro del método "main". Normalmente, esto se hace persiguiendo estas ventajas:

a) Claridad en la lectura del código, ya que todo aparece más ordenado
b) Reutilización de código: si tengo que hacer varias veces lo mismo no es necesario copiar el código, basta con llamar al método que lo hace
c) Estructurar el código siguiendo el patrón de diseño "divide y vencerás"

Los métodos pueden:

1. Devolver un único valor o no devolver ningún valor. Este valor puede ser cualquier cosa: un String, un Boolean, un Integer, un array, etc.
2. Recibir ninguno, uno o varios valores que se denominan parámetros. También pueden ser de distinto tipo.

La forma de escribir un método es así:

```
public static TIPO_DEV nombreMetodo(TIPO_PARAM1 nombreParam1, TIPO_PARAM2 nombreParam2, ...) {

        // AQUÍ IRÍA EL CÓDIGO

}
```

Donde:

- TIPO_DEV → es el tipo de dato que vamos a devolver. Si no devolvemos nada, escribimos "void"
- TIPO_PARAM1, TIPO_PARAM2… → son los tipos de parámetros que vamos a recibir
- nombreParam1, nombreParam2… → son los nombres de esos parámetros

Veamos ejemplos:

```java
public static void main(String[] args) {
    // Ejemplo de llamada a un método que no devuelve nada ni recibe nada
    imprimirNumero();
}

// Ejemplo de método que no devuelve nada (void) y que no recibe nada
public static void imprimirNumero() {
    System.out.println(7);
}
```


Abel Morillo Sánchez
Belén Tudó Ramírez

```java
public static void main(String[] args) {
    // Ejemplo de llamada a un método que no devuelve nada y recibe dos parámetros
    imprimirNumero("Hola", 7);
}

// Ejemplo de método que no devuelve nada (void) y que recibe dos parámetros
public static void imprimirNumero(String saludo, Integer numero) {
    System.out.println(saludo + " " + numero);
}
```

```java
public static void main(String[] args) {
    // Ejemplo de llamada a un método que devuelve un entero y recibe dos parámetros
    Integer cuadrado = imprimirNumero("Hola", 7);
}

// Ejemplo de método que devuelve un entero y que recibe dos parámetros
public static Integer imprimirNumero(String saludo, Integer numero) {
    System.out.println(saludo + " " + numero);
    Integer cuadrado = numero*numero;
    return cuadrado;
}
```

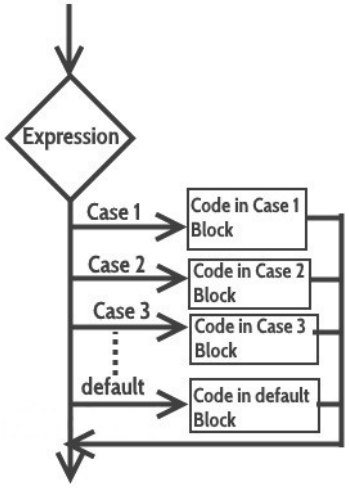
```java
public static void main(String[] args) {
    // Ejemplo de llamada a un método que devuelve un entero y no recibe nada
    Integer numero = imprimirNumero();
}

// Ejemplo de método que devuelve un entero y no recibe nada
public static Integer imprimirNumero() {
    Integer numero = 7;
    System.out.println(numero);
    return numero;
}
```

AVANZADO

¿Hay alguna alternativa a la estructura IF-ELSE?

Existe una alternativa a la estructura IF-ELSE. Es la estructura SWITCH. Se utiliza cuando que queremos preguntar por el valor de una variable y tenemos varios casos posibles.

La forma tradicional de usar la estructura SWITCH sería:




```
switch (expresión)
{
  case valor1:
    declaracion1;
    break;
  case value2:
    declaracion2;
    break;
  .
  .
  case valorN:
    declaracionN;
    break;
  default:
    declaracionDefault;
}
```

Como ejemplo, lo que en un IF escribiríamos así:

```
int x = 3;
if (x == 1) {
    System.out.println("x es igual que 1");
}
else if (x == 2) {
    System.out.println("x es igual que 2");
    }
    else if (x == 3) {
        System.out.println("x es igual que 3");
    }
    else {
        System.out.println("Ni idea de lo que es x");
    }
```

En un SWITCH lo escribimos así:

```
int x = 3;
switch (x){
    case 1:
        System.out.println("x es igual que 1");
        break;
    case 2:
        System.out.println("x es igual que 2");
        break;
    case 3:
        System.out.println("x es igual que 3");
        break;
    default:
        System.out.println("Ni idea de lo que es x");
}
```

Si no se indica BREAK después de las instrucciones de un CASE, se seguirán ejecutando todas las instrucciones del resto de CASE que haya debajo. Por eso es muy importante no olvidar el BREAK.

A veces, nos puede interesar no poner el BREAK porque para varios CASE siempre se hace lo mismo. Por ejemplo:

```
int x = someNumberBetweenOneAndTen;
switch(x){
    case 2:
    case 4:
    case 6:
    case 8:
    case 10: {
        System.out.println("x es es numero par"); break;
    }
}
```

Podemos ver ejemplos de switch en la web oficial de Java:
https://docs.oracle.com/javase/tutorial/java/nutsandbolts/switch.html

A partir de la versión 12 de java, se introduce una nueva forma de utilizar SWITCH que mejora la sintaxis y funcionalidad haciéndola más flexible. Se eliminan los break y utiliza sintaxis similar a las lambdas (->)

Switch como declaración:

```
int puntuacion = 85;
String calificacion;

// switch statement con lambdas-like para asignar una calificación
switch (puntuacion / 10) {
    case 10, 9 -> calificacion = "A";
    case 8 -> calificacion = "B";
    case 7 -> calificacion = "C";
    case 6 -> calificacion = "D";
    case 5, 4, 3, 2, 1, 0 -> calificacion = "F";
    default -> {
        System.out.println("Puntuación inválida: " + puntuacion);
        calificacion = "Sin calificación";
    }
}

System.out.println("La calificación es: " + calificacion);
```

Este switch no devuelve ningún valor, simplemente ejecuta una serie de instrucciones dependiendo del valor de puntuacion/10.

Switch como expresión:

```java
String mes = "Abril";

// switch como expresión utilizando yield
String estacion = switch (mes) {
    case "Diciembre", "Enero", "Febrero" -> "Invierno";
    case "Marzo", "Abril", "Mayo" -> {
        System.out.println("Estamos en primavera");
        yield "Primavera";
    }
    case "Junio", "Julio", "Agosto" -> "Verano";
    case "Septiembre", "Octubre", "Noviembre" -> "Otoño";
    default -> {
        System.out.println("Mes no válido: " + mes);
        yield "Desconocido";
    }
```

En este caso, el switch devuelve un valor que se asigna a la variable estacion. Según el mes que se introduce, el programa determina la estación del año.

Yield se usa dentro de un switch como expresión para devolver un valor desde los bloques case que pueden contener varias instrucciones.