Tema 3: EL TIPO DE DATOS BOOLEANO. INSTRUCCIONES CONDICIONALES Y DE REPETICIÓN.

3.1.- El tipo de datos booleano

3.1.1.- Dominio de valores

Es el conjunto de valores {verdadero, falso}

3.1.2.- Representación interna

Se puede representar con un bit que codifique uno de los valores como 0 y el otro como 1

3.1.3.- Representación externa y definición en Java

En Java es el tipo `boolean.` Los posibles valores que toma son: `true` y `false.`

```
boolean esPar; /* Es una variable que guarda si un número es par o no */
```

3.1.4.- Operaciones

Lógicas

Los operadores Y lógico (&&), O lógico (||) y NO lógico (!) trabajan con operandos lógicos o booleanos y devuelven resultados lógicos o booleanos igualmente

En la siguiente tabla se muestran cuáles son los valores de A && B, A || B y !A dependiendo de cuáles sean los valores de A y de B.

A	**B**	**A && B**	**A || B**	**!A**
falso	falso	falso	falso	verdadero
falso	verdadero	falso	verdadero	verdadero
verdadero	falso	falso	verdadero	falso
verdadero	verdadero	verdadero	verdadero	falso

Esta tabla se interpreta de esta manera. Por ejemplo, si `A` es falso y `B` es verdadero (2ª fila), entonces (`A&&B`) es falso, (`A||B`) es verdadero y (`!A`) es falso. Si no le encontráis sentido a eso, imaginaros que A es "hace frío" y B es "llueve". Si A es falso entonces "no hace frío", y si B es verdadero entonces "llueve". En ese caso, (`A&&B`)= "hace frío y llueve" es falso (puesto que no hace frío), (`A||B`)= "hace frío o llueve" es verdadero (ya que llueve) y (`!A`)= "no hace frío" es verdadero.

Como se puede ver, una condición formada por varias condiciones compuestas mediante un Y lógico es cierta si y sólo si todas las condiciones son ciertas.

Como se puede ver, una condición formada por varias condiciones compuestas mediante un O lógico es cierta si y sólo si existe al menos una condición que es cierta.

De comparación

Existen otros operadores que devuelven valores booleanos y que pueden ser aplicados a otro tipo de operandos: enteros, caracteres, reales, booleanos, etc.

Son los de comparación:

<	MENOR QUE
<=	MENOR O IGUAL QUE
>	MAYOR QUE
>=	MAYOR O IGUAL QUE
==	IGUAL A
!=	DISTINTO DE

Revisión de la prioridad entre operadores

En la siguiente tabla aparece la prioridad de los operadores aritméticos y voléanos. Son más prioritarios los que están más arriba, y operadores de igual prioridad se evalúan de izquierda a derecha.

( )
!
* / %
+ -
< > <= >= == !=
&&
||
=

Ejercicio: Evaluar la expresión (`a%2==0 && a>=0 && a<100 || a%2==1`) para distintos valores de `a` (a=50, a=51, a=109 y a=110)

¿Puedes explicar cuándo esa expresión se evalúa a cierto? Solución en este pie de página[2].

Ejercicio: Evaluar la expresión (`a!=0 && b%a==0`) para distintos valores (b=14, a=7)(b=15, b=4)(b=8, a=0)

¿Puedes explicar cuándo esa expresión se evalúa a cierto? Solución en este pie de página[3].

NOTA: En Java los operadores && y || se evalúan de manera que cortocircuitan, esto es, cuando hay varios operandos unidos por &&, en cuanto se encuentre uno a la izquierda que devuelva FALSO se deja de evaluar el resto puesto que el resultado ya será VERDADERO. De la misma manera, cuando hay varios operandos unidos por ||, en cuanto se encuentre uno (empezando por la izquierda) que devuelva VERDADERO se deja de evaluar el resto puesto que el resultado será VERDADERO.

[2] La expresión da cierto si y sólo si el número `a` es impar o par entre 0 y 99

[3] La expresión sirve para ver si b es un número múltiplo de a, y controla que no se divida por 0, para que no se produzca un error

3.2.- Instrucción condicional simple (si-entonces-si no)

Sintaxis: **if** (<expr_booleana>) <bloque_instr 1>
[**else** <bloque_instr 2>]

Nota: cuando en la sintaxis algo aparece entre corchetes [ ], significa que es opcional.

Semántica: Se evalúa la expresión booleana y si el resultado es VERDADERO entonces se ejecuta el primer bloque de instrucciones y si es FALSO y se ha definido una parte **else** entonces se ejecuta el segundo bloque de instrucciones.

Ejemplo: Parte de un programa que escriba por pantalla el contenido de la variable entera `a` seguido de "es un número positivo" o "no es un número positivo" dependiendo de si lo es o no.

```
       if (a>=0)
              System.out.println (a+" es un número positivo");
       else   System.out.println (a+" no es un número positivo");
```

If anidados

Es posible que dentro de un bloque de instrucciones aparezcan más instrucciones **if** (**if** anidados). Hay que indicar que la parte **else** se asocia siempre con el último **if** abierto y no terminado.

Ejemplo:

```
       if (a>=0)
              if (a==0)
                     System.out.println (a+" es cero");
              else /* Se asocia con el último if NO TERMINADO */
                     System.out.println (a+" es positivo");
       else System.out.println (a+" es negativo");
```

Si se quisiera que el **else** correspondiera al otro **if** entonces habría que “terminar” el segundo

```
       if (a>=0)
              { if (a==0)
                     System.out.println (a+" es cero"); }
       else /* Se asocia con el último if NO TERMINADO */
              System.out.println (a+" es negativo");
/* Nótese que en este caso si a es positivo NO SE ESCRIBE NADA por pantalla */
```

3.3.- Instrucción condicional múltiple o de casos (switch)

Sintaxis: **switch (** <expresión> **) {**
{ {**case** <constante> **:** } <bloque_instr> }
[**default :** <bloque_instr >]
}

Semántica:

- Primero se evalúa la expresión del **switch**.
- Después se busca por orden en las distintas sentencias **case** si se encuentra una constante igual al valor de la expresión y si es así se ejecuta el bloque e instrucciones correspondiente a dicha sentencia **case**. Si el valor de la expresión del **switch** no se encuentra en ninguna sentencia **case** y se ha definido una sentencia **default**, entonces se ejecuta ese bloque.
- Si en el bloque de instrucciones a ejecutar se encuentra la instrucción **break** entonces se termina de ejecutar el **switch** y si no, entonces se siguen ejecutando por orden los bloques de instrucciones de las siguientes sentencias **case** (hasta que no haya más o se ejecute una sentencia **break**).

Ejemplo: parte de un programa que deja en la variable `nd` el número de días correspondientes al número de mes almacenado en la variable `mes` y año almacenado en la variable `a`.

```
switch (mes)
 { case 1: case 3: case 5: case 7: case 8: case 10: case 12: nd=31;
break;
  case 4: case 6: case 9: case 11: nd=30; break;
  case 2: if (a%400==0 || a%4==0 && a%100!=0) nd=29; else nd=28;
break;
  default: System.out.println("ERROR en número de mes. "); }
```

Nota: todo lo que se puede programar con una sentencia **switch** se puede programar con sentencias **if** anidadas.

```
if (mes==1 || mes==3 || mes==5 || mes==7 || mes==8 || mes==10 || mes==12)
       nd=31;
else if (mes==4 || mes==6 || mes==9 || mes==11)
              nd=30;
     else  if (mes==2)
              if (a%400==0 || a%4==0 && a%100!=0) nd=29; else nd=28;
           else System.out.println("ERROR en número de mes. ");
```

Más información en:
http://java.sun.com/docs/books/tutorial/java/nutsandbolts/switch.html