

Algunas veces vamos a necesitar que un bloque de código se ejecute dependiendo de una condición, vamos a ver primero el condicional **If**:

```
Condicionales.java x
1 package apuntes;
2 import stdlib.StdIn;
3
4 public class Condicionales {
5
6     public static void main (String[] args) {
7
8         System.out.print("¿Quieres que te salude? -> ");
9         boolean answer = StdIn.readBoolean();
10
11         if (answer == true) {
12
13             System.out.println("Hola gente.");
14
15         }
16
17     }
18
19 }
```

>>>

```
Problems @ Javadoc Declaration Console x
<terminated> Condicionales [Java Application] C:\Program File
¿Quieres que te salude? -> true
Hola gente.
```

```
Problems @ Javadoc Declaration Console x
<terminated> Condicionales [Java Application] C:\Program File
¿Quieres que te salude? -> 0
```

Como veís, si la condición del if es afirmativa, las línea 13 se ejecuta, pero si *false* o 0, no.

Ahora bien, **¿Qué puede ser una condición?**

Una condición puede ser una variable de tipo *boolean* o una comparación, he aquí las comparaciones existentes:

- == -> algo *igual* a algo
- != -> algo *distinto* de algo
- < -> algo *menor* que algo
- > -> algo *mayor* que algo
- <= -> algo *menor o igual* que algo
- >= -> algo *mayor o igual* que algo

¿Y si queremos realizar varias comparaciones?

Podemos hacer que se tengan que cumplir muchas condiciones, o que con que se cumpla una ya baste:

- && -> Condicion1 && Condicion2 [es semejante a poner: Condicion1 y Condicion2]
- || -> Condicion1 || Condicion2 [es semejante a poner: Condicion1 o Condicion2]

Veamos más en profundidad los *ifs*.

IFS

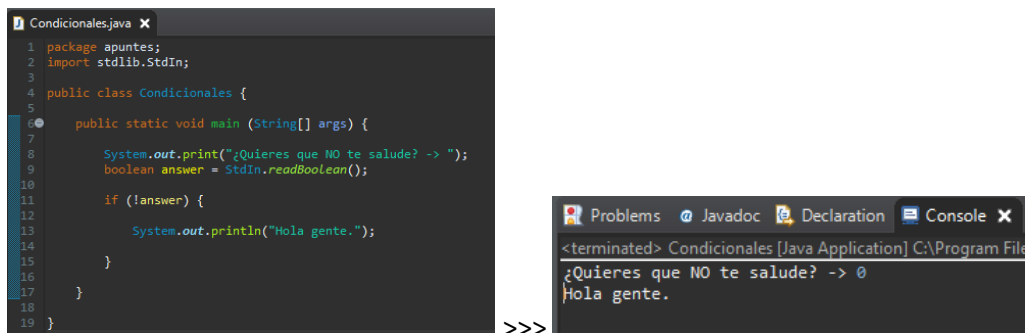
Gramaticalmente se puede decir:

```
if (true) { //Hago esto }
```

Una comparación devuelve un boolean, que es *true/false*, y una variable de tipo *boolean* también se puede usar directamente, por ejemplo, en la primera imagen, no hace falta hacer esa comparación, se podría poner:

```
if (answer) {
```

Solamente, ya que *answer* contiene *true/false*, de tal forma que se respeta el *if (true)*, también podemos realizar la negación de una condición poniendo "!" delante, cuando ponemos esto, los *true* se convierte en *false* y viceversa:



The screenshot shows a Java IDE with a file named 'Condicionales.java'. The code is as follows:

```
1 package apuntes;
2 import stdlib.StdIn;
3
4 public class Condicionales {
5
6     public static void main (String[] args) {
7
8         System.out.print("¿Quieres que NO te salude? -> ");
9         boolean answer = StdIn.readBoolean();
10
11         if (!answer) {
12             System.out.println("Hola gente.");
13         }
14     }
15 }
16
17
18
19 }
```

Below the code editor, the console output is visible:

```
<terminated> Condicionales [Java Application] C:\Program File
¿Quieres que NO te salude? -> 0
Hola gente.
```

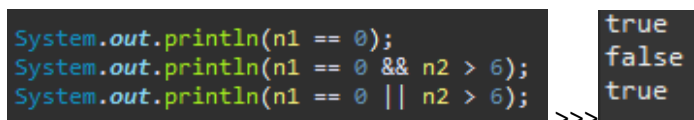
Fijaros en que he puesto, si quieres que **NO** te salude, y luego he negado la respuesta, por lo tanto, si yo introduzco *false*, y luego lo niego en la condición del *if*, se va a meter dentro, ya que la negación de *false* es *true*, es un poco lioso de entender, pero si experimentáis un poco con esto por vuestra cuenta creo que lo pillareis más rápido, aun así, voy a dejar unos cuantos ejercicios para resolver al final.

Ahora vamos a ver unos cuantos ejemplos de condiciones.

```
boolean verdad = true;
boolean mentira = false;

int n1 = 0, n2 = 5;
double d1 = 0.0, d2 = 0.001;
```

Vamos a hacerlo en base a estas variables, se me pasó mencionar que podemos crear varias variables en una sola sentencia, poniendo una coma entre medias, Ej: *int n1, n2 = 5;*



The screenshot shows the following code:

```
System.out.println(n1 == 0);
System.out.println(n1 == 0 && n2 > 6);
System.out.println(n1 == 0 || n2 > 6);
```

The console output is:

```
true
false
true
```

Como veis es bastante intuitivo, en un *if*, si la condición resulta *true*, se mete, sino no, es bastante fácil, las funciones se pueden meter en una condición, si recordamos la función *sumar* del PDF anterior, podríamos hacer algo como: *if (sumar (5,7) > 6) { //hacer algo }* En este caso es bastante fácil, pero hay funciones mucho más complejas como imaginareis, también si una función devuelve un booleano, se puede poner ella sola, porque si nos fijamos **una función no es más que una variable, que se calcula con un procedimiento dentro de la misma en base a unos parámetros.**

Otros ejemplos:

```
System.out.println(d1 == 0); true
System.out.println(d2 == 0); false
System.out.println(d1 < d2); true
>>>
```

Nada que comentar.

```
System.out.println(verdad); true
System.out.println(mentira || n1 > -2); true
System.out.println((d1 == 0 && d2 > 1) || verdad); true
>>>
```

En el tercer *System*, tened cuidado con los paréntesis, si vais a poner todo *&&* o todo *||*, no pasa nada por no ponerlos, pero si los mezcláis, os recomiendo poner paréntesis.

```
/*
 * sumar (int a,b) : int
 * POST: Suma dos numeros
 */
public static double sumar (double a, double b) {
    return a+b;
}
```

```
System.out.println(sumar(n2+d2,n1) == 5.001); >>> true
```

Como veis en los parámetros se pueden pasar sumas, restar o lo que sea, mientras en valor final de lo que se pasa como parámetro corresponda con el tipo, en este caso, *double*, y como veis también la función no es más que una variable.

Y el último ejemplo para ifs:

```
/*
 * and (boolean cond1,cond2) : boolean
 * POST: Decide ambos son true
 */
public static boolean and (boolean cond1, boolean cond2) {
    return cond1 && cond2;
}
```

```
System.out.println(and(verdad,mentira) == false); >>> true
```

Como le pasamos *true* y *false*, la función devolverá *false*, y como *false* es igual a *false*, nos escribirá un *true*.

Hay más curiosidades, y mecánicas, pero no puedo acordarme de todas, además la mejor forma de aprender a usar los condicionales es practicando.

Dicho esto, empecemos con la sentencia *else*.

ELSE

Imaginemos que queremos que, si una condición se cumple, se realice un bloque de código, y sino (*else*) se realice otro bloque, esto es lo que se consigue con el *else*.

Recordemos que tenemos estas variables:

```
boolean verdad = true;
boolean mentira = false;

int n1 = 0, n2 = 5;
double d1 = 0.0, d2 = 0.001;
```

```
if (verdad) {
    System.out.println("Hola");
} else {
    System.out.println("Mundo");
}
```

>>> "Hola"

Esto es lo mismo que poner:

```
if (verdad) {
    System.out.println("Hola");
}
if (!verdad){
    System.out.println("Mundo");
}
```

>>> "Hola"

Es decir, si se cumple la condición, escribe "Hola", y si no, escribe "Mundo".

```
if (mentira) {
    System.out.println("Hola");
} else {
    System.out.println("Mundo");
}
```

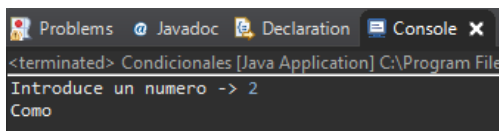
>> "Mundo"

Vamos a hablar de lo que es **anidar ifs**, y por qué es importante, para ello nos vamos a basar en la siguiente estructura, olvidándonos de todo lo anterior:

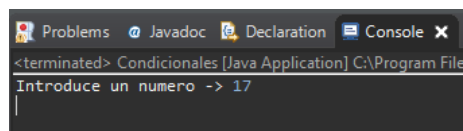
```
Condicionales.java X
1 package apuntes;
2 import stdlib.StdIn;
3
4 public class Condicionales {
5
6     public static void main (String[] args) {
7
8         System.out.print("Introduce un numero -> ");
9         int x = StdIn.readInt();
10
```

Vamos a basarnos en lo que introduzcamos por teclado ->

```
if (x == 0) {  
    System.out.println("Hola");  
}  
if (x == 1) {  
    System.out.println("Mundo");  
}  
if (x == 2) {  
    System.out.println("Como");  
}  
if (x == 3) {  
    System.out.println("Estamos?");  
}
```



Problems Javadoc Declaration Console X
<terminated> Condicionales [Java Application] C:\Program File
Introduce un numero -> 2
Como

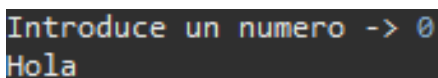


Problems Javadoc Declaration Console X
<terminated> Condicionales [Java Application] C:\Program File
Introduce un numero -> 17
|

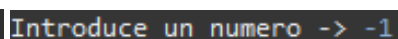
Como veis realizo cada comparación, y si una es *true* se mete, el problema es que con esta forma va a realizar **TODAS** las comparaciones, aquí es donde entra el anidado de ifs.

```
if (x == 0) {  
    System.out.println("Hola");  
} else {  
    if (x == 1) {  
        System.out.println("Mundo");  
    } else {  
        if (x == 2) {  
            System.out.println("Como");  
        } else {  
            if (x == 3) {  
                System.out.println("Estamos?");  
            }  
        }  
    }  
}
```

Nota de referencia: Ésta imagen



Introduce un numero -> 0
Hola



Introduce un numero -> -1

Como veis la estructura es mucho más liosa, pero mucho más óptima a su vez, y la eficiencia la tienen muy en cuenta en la UPM, por lo que yo trataría de ponerlo de esta forma, ya que suelen quitar 2 puntos sobre 10 por eficiencia, y por pregunta...

Ahora veamos una última cosita respecto a los *ifs* y *elses*.

Dada esta estructura:

```
if (x == 0) {  
    System.out.println("Hola");  
} else {  
    if (x == 1) {  
        System.out.println("Mundo");  
    } else {  
        if (x == 2) {  
            System.out.println("Como");  
        } else {  
            if (x == 3) {  
                System.out.println("Estamos?");  
            } else {  
                System.out.println("Todo lo demas fallo.");  
            }  
        }  
    }  
}
```

>>>

```
Introduce un numero -> 17  
Todo lo demas fallo.
```

Como veis, al meter un número que no cumple ninguna condición anterior, se va al último *else*, ya que haría algo como: si se cumple esto, hace esto, sino, si se cumple la segunda condición hace esto, sino ... sino se ha cumplido nada se va al último *else*, que básicamente su condición es que no se cumpla ninguna de las anteriores, al igual que cuando tienes solo un *if* y un *else*, la condición del *else* es que no se cumpla la condición del *if*. Es un lío, pero si se piensa detenidamente tiene sentido.

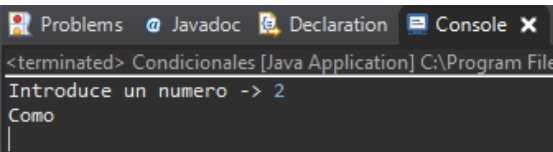
Y bueno, la importancia de la anidación de *ifs* es que como veis, si se mete en la primera condición, ya no realiza las siguientes, se sale directamente al final, y esto puede que para cuatro condiciones no parezca mucho, pero si son programas más complejos, o que se van a repetir a la larga como en la página web de una empresa, pues es algo a considerar, ya que ahorrarse 4ms una vez no es mucho, pero ahorrárselos 1 millón de veces es algo más de una hora, y esto en el caso de 4ms, imaginaos si ahorramos 1 segundo entero, pues son 227 días de procesos inútiles, así que recordad que la eficiencia es súper importante, y aún más en proyectos grandes.

Pasemos ahora a hablar de los *switchs*...

SWITCH

Yo lo definiría como una forma ordenada de anidar *ifs*, tienen la siguiente estructura:

```
switch (x) {  
    case 0:  
        System.out.println("Hola");  
        break;  
    case 1:  
        System.out.println("Mundo");  
        break;  
    case 2:  
        System.out.println("Como");  
        break;  
    case 3:  
        System.out.println("Estamos?");  
}
```

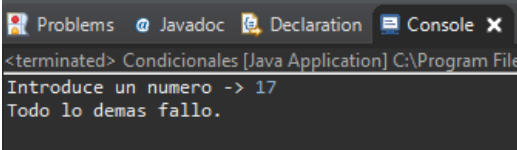


>>>

Yo introduzco la variable *x*, y en base al valor de esa variable *< switch (x) >*, vamos a realizar una cosa u otra, si *x == 0* -> *S.o.p("Hola"); break;* si *x == 2* -> *S.o.p("Como"); break;*

Notese que todas llevan un *break* al final de cada caso, y antes del siguiente, salvo el último, el case 3, esto lo explicaré más adelante, ahora bien, ésta estructura se parece a la primera imagen de *ifs* anidados, la cual marqué anteriormente, y me refiero a que no tiene un bloque que se llama si todo falla, es decir, si yo introduzco, 4 o 400, no va a ocurrir ninguno de estos casos, para ello tenemos el caso *default*, el cual es llamado cuando no se ha entrado en ningún otro caso:

```
switch (x) {  
    case 0:  
        System.out.println("Hola");  
        break;  
    case 1:  
        System.out.println("Mundo");  
        break;  
    case 2:  
        System.out.println("Como");  
        break;  
    case 3:  
        System.out.println("Estamos?");  
        break;  
    default:  
        System.out.println("Todo lo demas fallo.");  
}
```



>>>

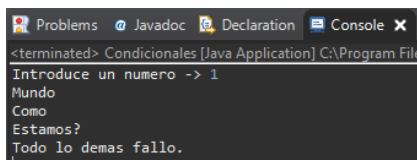
Creo que se entiende cómo funciona el *switch*, pero pasemos a ver qué ocurre si quitamos esos *breaks*.

```

switch (x) {
case 0:
    System.out.println("Hola");
case 1:
    System.out.println("Mundo");
case 2:
    System.out.println("Como");
case 3:
    System.out.println("Estamos?");
default:
    System.out.println("Todo lo demas fallo.");
}

```

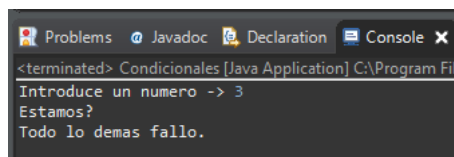
>>>



```

Problems Javadoc Declaration Console X
<terminated> Condicionales [Java Application] C:\Program File
Introduce un numero -> 1
Mundo
Como
Estamos?
Todo lo demas fallo.

```



```

Problems Javadoc Declaration Console X
<terminated> Condicionales [Java Application] C:\Program File
Introduce un numero -> 3
Estamos?
Todo lo demas fallo.

```

Los *breaks* indican que se salga de una estructura, si está contenido en un *if*, se saldrá de este, si lo está dentro de un bucle *while*, *do while* o *for*, se saldrá de los mismos (ya veremos estos bucles), y si está dentro de un *switch*, al leerlo se saldrá del mismo, lo que ocurre aquí, es que cuando yo introduzco “1”, se va al case 1, y realiza todos los bloques que hay por debajo, lo mismo para cuando introduzco 3, esto se debe a que para que nos entendamos, lo que hay dentro de un *switch* es un único bloque de código, y tú, con los *cases*, le dices desde donde empezar a leer ese bloque de código, por eso introducimos los *breaks* antes de un *case*, y por eso no hace falta ponerlo en el último caso.

Aunque estos *breaks* se puedan usar en todo tipo de bucles, la UPM considera mala práctica usarlos en algo que no sean los condicionales ***switch***, por lo que no los vamos a usar en ningún otro sitio.

Otra cosa importante referente a los ***returns***, los *returns* son un *break* absoluto dentro de una función, si está dentro de 30 bucles da exactamente igual, toda la función se detendrá y devolverá el valor que sea, la UPM **PROHIBE** poner *returns* dentro de bucles *while*, *do while* y *for*, pero si podemos ponerlos dentro de *ifs*, siempre y cuando estos *ifs* no estén contenidos dentro de un bucle, yo recomiendo que, si tenemos una función, en la primera línea de ésta poner:

```
TIPO_DE_DATO_A_DEVOLVER resultado;
```

```
//Resto de la función
```

```
return resultado;
```

Es decir, devolver el resultado siempre al final, y así nunca nos equivocaremos, por poner un *return* dentro de un bucle **restan 2 puntos**.

Dicho esto he de decir que recomiendo no usar la letra “ñ” ni tildes en vuestros programas, ya que es sintaxis “exclusiva” del castellano, por lo que algunos programas pueden liarse a la hora de usarlos, tenéis en la carpeta de ejercicios de drive unas clases para implementar funciones solamente utilizando *ifs*, junto con las soluciones, pero recomiendo no mirarlas.