

PROGRAMACIÓN

UD 2: Estructuras básicas de control.

Índice

- 1 Condiciones simples
- 2 El tipo lógico
- 3 Operadores de comparación
- 4 Comparación de cadenas (Strings)
- 5 Operadores lógicos
- 6 Precedencia de operadores
- 7 Instrucción if..else
- 8 Instrucción switch
- 9 Bucles
 - 1 Bucle for
 - 2 Bucle while
 - 3 Bucle *do .. while*

1. Condiciones simples.

- En la resolución de un problema, es necesario a menudo **tomar decisiones** en función de las características del mismo, esto es, **se necesita poder alterar la secuencia de cálculos que se efectúan en función de los datos de entrada**, o de resultados obtenidos con antelación.
- Las instrucciones condicionales *permiten construir programas que tomen decisiones* acerca de los cálculos a efectuar en función de las características del problema.

1. Condiciones simples.

- Controlar el flujo es *determinar el orden en el que se ejecutarán las instrucciones en nuestros programas*.
- Si NO existiesen las sentencias de control, entonces los programas se ejecutarían de forma secuencial, empezarían por la primera instrucción e irían una a una hasta llegar a la última.
- Pero, obviamente este panorama sería muy malo para el programador. Por un lado, en sus programas no existiría la posibilidad de elegir uno de entre varios caminos en función de ciertas condiciones (sentencias de selección). Y por el otro, no podrían ejecutar algo repetidas veces, sin tener que escribir el código para cada una (sentencias repetitivas).

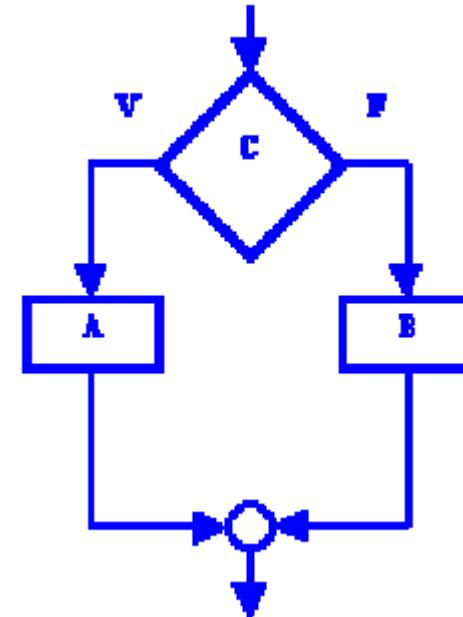
1. Condiciones simples.

- Para estos dos problemas tenemos dos soluciones: las ***sentencias de control selectivas*** y ***las repetitivas***.
- Estos dos conjuntos de sentencias se les llama **estructuradas o estructuras de selección y repetitivas**, porque a diferencia de las simples pueden contener en su cuerpo otras sentencias.
- Las **sentencias selectivas** se llaman así porque **permiten elegir uno de entre varios caminos por donde seguirá la ejecución del programa**.
- Esta **selección** viene determinada por la **evaluación de una expresión lógica**.
- Este tipo de sentencias se dividen en **tres**: (las vemos en esta unidad)
 - La sentencia ***if***.
 - La sentencia ***if-else***.
 - La sentencia ***switch***.

1. Condiciones simples.

• Selección (pseudocódigo)

- Inicio
- SI condición
 - Sentencias 1
- SINO
 - Sentencias2
- FIN SI
- Fin



- Se evalúa la condición: si es cierta se ejecutan las *Sentencias 1* y si es falsa las *Sentencias 2*.
- Se ejecuta unas o las otras sentencias pero NUNCA los dos caminos.

1. Condiciones simples.

- **Actividad 1:** Hacer un pseudocódigo para guiar a una persona a salir de esta clase hasta la calle.
 - Considerar:
 - Que pueden haber puertas cerradas.
 - Doblar a la izquierda o derecha solamente, no es una orden precisa. Indicar el giro en grados.
 - Este ejercicio requiere ordenes secuenciales y selectivas.
 - Discutirlo.

1. Condiciones simples.

- **Ejemplo:** Se desea determinar si cierta variable temperatura tiene un valor mayor o menor que 100 (escribiendo un mensaje según lo sea o no). Esto podría expresarse mediante las instrucciones:

Si (temperatura>100)

System.out.println("La temperatura es mayor que 100!");

Sino

System.out.println("La temperatura es menor o igual que 100");

- Una vez escrito el mensaje correspondiente, la ejecución del programa continuará en la instrucción siguiente (que en el ejemplo aparece indicada mediante puntos suspensivos).

2. El tipo lógico.

La condición anterior:

`temperatura>100`

adquiere al ser evaluada (del mismo modo que cualquier otra condición) uno de los dos posibles valores lógicos:

- *cierto*
- *falso*

que se denotarán en los programas como: *true o false*.

Se denomina **expresión lógica**, a cualquier expresión, secuencia de operandos y operadores, que al ser evaluada tiene uno de los valores: *true o false*. Naturalmente, cualquier condición es una expresión lógica.

2. El tipo lógico.

Toda expresión lógica tiene además como tipo de datos el tipo lógico que se denotará en los programas como: ***boolean***.

```
boolean alta;  
.....  
alta = temperatura > 100;  
Si (alta)  
    System.out.println("La temperatura es mayor que 100!");  
Sino  
    System.out.println("La temperatura es menor o igual que 100");
```

- Los dos únicos valores constantes que puede tener cualquier expresión de tipo ***boolean*** son los ya mencionados: ***true o false***.

3. Operadores de comparación.

- Los **operadores de comparación**, también denominados *operadores relacionales*, se utilizan para *comparar entre sí valores de elementos que tienen el mismo tipo simple*.
- Son tipos simples los tipos numéricos ya vistos, junto con el tipo *char* y el *boolean*.
- Los operadores y su significado se listan a continuación:

OPERADOR	NOMBRE OPERACIÓN	EJEMPLO
<	menor	temperatura < 100
>	mayor	minuto > 10
<=	menor o igual	hora <= 24
>=	mayor o igual	dia >= 1
==	igual	mes == 12
!=	distinto	temperatura != 100

3. Operadores de comparación.

Matemática	Java
=	==
<	<
>	>
\leq	\leq
\geq	\geq
\neq	$!=$

4. Comparación de cadenas (*Strings*).

- La comparación entre dos cadenas (**Strings**) se realiza en Java de un modo distinto a como se efectúa la comparación entre valores pertenecientes a tipos elementales (tipos de datos simples).
- Un **String** es en Java un objeto, por lo que las operaciones de comparación se efectúan mediante la notación de punto, que se estudiará más adelante.
- Pero os adelanto que se tendrá que utilizar el método **equals()** de la clase *String*, el cual recibe un *parámetro de tipo Object que puede ser un String, que será la cadena a comparar*.

```
public boolean equals(Object anObject)
```

4. Comparación de cadenas (*Strings*).

- Ejemplo:

```
Scanner teclado = new Scanner(System.in);
String asig = teclado.nextLine();
Si (asig.equals("PRO"))
    System.out.println("Sí, es de primero");
Sino Si (asig.equals("PMM"))
    System.out.println("No, es de segundo");
Sino Si (asig.equals("PES"))
    System.out.println("No es de DAM y es de segundo de
DAW");
Sino Si ...
...
Sino Si (asig.equals("DI"))
    System.out.println("No, es de segundo");
Sino
    System.out.println("No es de este Ciclo Formativo");
```

5. Operadores lógicos.

- Es posible construir una nueva expresión lógica yuxtaponiendo otras expresiones lógicas, ligadas entre sí mediante las denominadas **conectivas lógicas**.

Si ((temperatura>=15) && (temperatura<=20))
.....



5. Operadores lógicos.

- Además de la conectiva del ejemplo anterior para expresar la **conjunción o y-lógico** `&&`, existen también conectivas para expresar la **disyunción u o-lógico** `||`, la **o-exclusiva** `^` y la **negación** `!`
- Ejemplo1: temperatura es mayor o igual que 15.

Si `((temperatura>15) || (temperatura=15))`

- Ejemp `// similar a (temperatura >= 15)` ntre o y 100.
(no sale en el examen AUR)

```
if ((temperatura>=0) ^ (temperatura<=100)) ....  
//similar a (temperatura>100) || (temperatura<0)  
.....
```

5. Operadores lógicos.

COMO CURIOSIDAD

- La disyunción exclusiva $p \oplus q$ puede ser expresada en términos de *conjunción lógica* (\wedge), *disyunción lógica* (\vee), y *negación* (\neg) de la siguiente manera:
$$p \oplus q = (p \wedge \neg q) \vee (\neg p \wedge q)$$
- La conjunción exclusiva $p \oplus q$ puede ser expresada de la siguiente manera:
$$p \oplus q = \neg(p \wedge q) \wedge (p \vee q)$$
- Esta represen~~tación~~ $p \oplus q = \neg(p \wedge q) \wedge (p \vee q)$ puede resultar útil en la construcción de un circuito o una red, ya que sólo tiene un operador \neg y un número reducido de operadores \wedge y \vee . La prueba de esta identidad es la siguiente:

$$\begin{aligned} p \oplus q &= (p \wedge \neg q) \vee (\neg p \wedge q) \\ &= ((p \wedge \neg q) \vee \neg p) \wedge ((p \wedge \neg q) \vee q) \\ &= ((p \vee \neg p) \wedge (\neg q \vee \neg p)) \wedge ((p \vee q) \wedge (\neg q \vee q)) \\ &= (\neg p \vee \neg q) \wedge (p \vee q) \\ &= \neg(p \wedge q) \wedge (p \vee q) \end{aligned}$$

$$p \oplus q$$

$$p \oplus q = \neg((p \wedge q) \vee (\neg p \wedge \neg q))$$

5. Operadores lógicos.

- Java proporciona dos operadores diferentes para cada una de las conectivas.
 - Los denominados **cortocircuitados** tienen la particularidad de que en las subexpresiones donde aparecen **se evalúa, siguiendo el recorrido de izquierda a derecha**, la mínima parte de la subexpresión necesaria para conocer el valor lógico. **Si no se cumple la condición de un término no se evalúa el resto de la operación.**
 - Por ejemplo: $(a == b \&& c != d \&& h >= k)$ tiene tres evaluaciones: la primera comprueba si la variable a es igual a b. Si no se cumple esta condición, el resultado de la expresión es falso, ya que las tablas de verdad de la AND dicen que si un operando de los dos es falso, el resultado es falso, por lo tanto no se evalúan las otras dos condiciones posteriores.
 - Sin embargo, cuando se utiliza un operador **no cortocircuitado** **se produce siempre la evaluación de toda la subexpresión** donde éste aparece. En Java, los operadores no cortocircuitados para la *conjunción* y la *disyunción* son respectivamente: **&** y **|**. NO LOS VAMOS A USAR.

5. Operadores lógicos.

- A continuación, se muestran las **tablas de verdad** de los operadores lógicos vistos:

x	y	x && y x & y	x y x y	x ^ y	!x
false	false	false	false	false	true
false	true	false	true	true	true
true	false	false	true	true	false
true	true	true	true	false	false

5. Operadores lógicos.

A continuación, se muestran las tablas de verdad de los operadores lógicos vistos: (continuación)

- TRUCO: Puedes ver el valor de true como si fuera un 1, y el valor de false como si fuera un 0. Luego el operador lógico && (AND) como si fuera el producto *, y el operador lógico || (OR) como si fuera la suma +.

- Ejemplo:

- true && true -> $1 * 1 = 1 > 0$ TRUE
- true && false -> $1 * 0 = 0 > 0$ FALSE
- true || false -> $1 + 0 = 1 > 0$ TRUE
- true || true -> $1 + 1 = 2 > 0$ TRUE

Siempre que sea > 0 será TRUE. Si es ≤ 0 será FALSE.

6. Precedencia de operadores.

M	grupo 0:	()
a	grupo 1:	++ -- +(unario) -(unario) !
y	grupo 2:	* / % ←
o	grupo 3:	+ -
r	grupo 5:	> >= < <=
p	grupo 6:	== !=
r	grupo 7:	&
e	grupo 8:	^
c	grupo 9:	
e	grupo 10:	&&
d	grupo 11:	
e	grupo 12:	?:
n	grupo 13:	(operador ternario)
ci		= op= (op es uno de: +,-,*,/,%,&, ,^)
a		

Dentro de una misma expresión, si nos encontramos varios operadores de un mismo grupo, tendrán preferencia mayor de derecha a izquierda.

Recuérdese que las expresiones se evalúan, por defecto, de izquierda a derecha.

7. Instrucción “if...else...”.

- **1^a forma:** La forma más simple de una **instrucción condicional** en Java es la siguiente:

```
if (B){  
    S  
}
```

- donde S es una instrucción (por ej., una asignación seguida de “;”) o bloque cualquiera, y B una condición (expresión lógica).
- Su ejecución se efectúa evaluando la condición B, de forma que si es cierta se ejecuta S. Al acabar la ejecución de la instrucción (o bloque), ésta continúa en la instrucción siguiente a la condicional.

7. Instrucción “if...else...”.

- La idea básica de esta sentencia es la de *encontrarnos ante una bifurcación de un camino*, en la que seguiremos por uno u otro camino dependiendo de la respuesta a una pregunta que se halla escrita en la bifurcación.
- Una variante de esto es que en lugar de dos posibilidades, se nos presenten más caminos por los que poder seguir.

7. Instrucción “if...else...”.

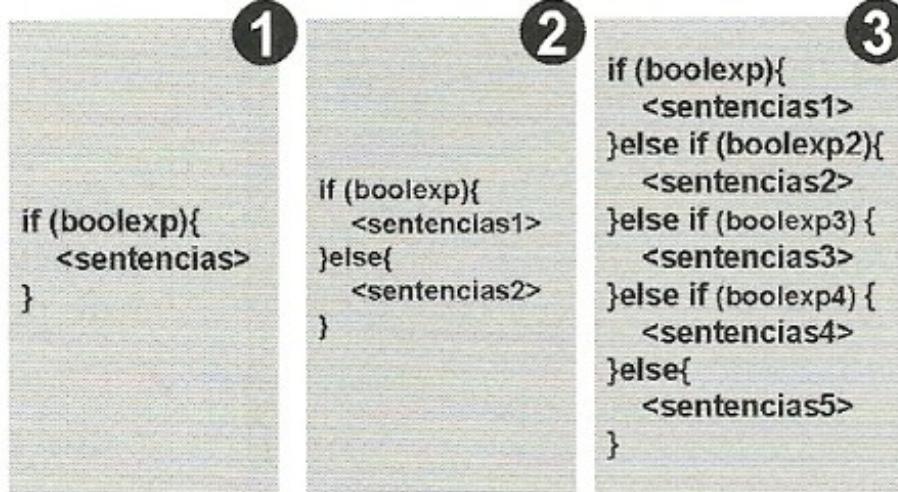


Figura 3.1. Estructura IF

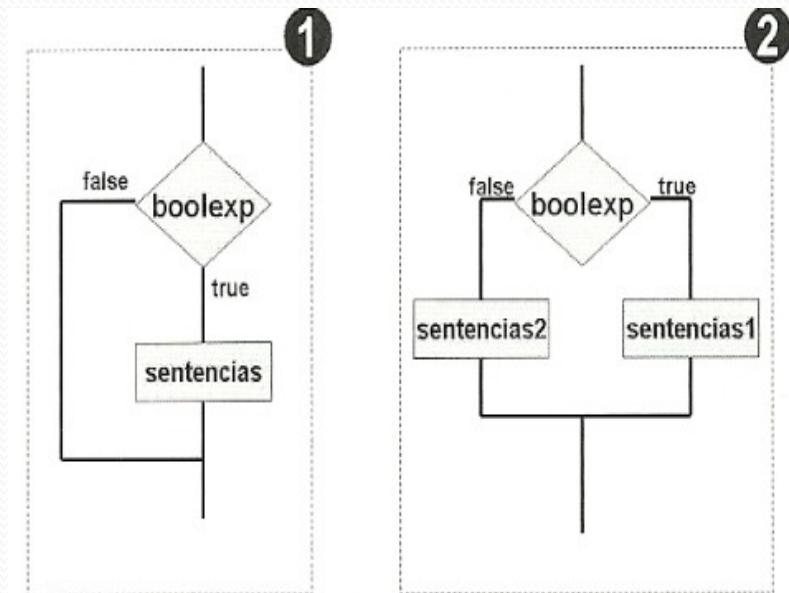


Figura 3.2. Flujo de ejecución en estructuras IF

7. Instrucción “*if...else...*”.

- Esta sentencia evalúa una condición:
 - si la condición es cierta
 - ejecuta un código,
 - si no es cierta
 - ejecuta el código a continuación del ELSE (si éste existe) o la instrucción inmediatamente después del bloque de instrucciones del IF.

7. Instrucción “if...else...”.

- **2^a forma:** La otra forma de instrucción condicional es:

```
if (B){  
    S1  
}else{  
    S2  
}
```

- donde S₁ y S₂ son instrucciones simples (o bloques) cualesquiera. Su ejecución se efectúa evaluando la condición *B*, de forma que si esta es cierta se ejecuta S₁, por lo contrario, si no es cierta, se ejecuta S₂. Al acabar la instrucción condicional la ejecución continúa en la instrucción siguiente.

7. Instrucción “if...else...”.

- Ejemplo:

Pseudocódigo	Java
IF N > 2	if (n>2)
Mostrar “Es mayor que 2”	system.out.println (“Es mayor que 2”);
SINO	else
Mostrar “NO Es mayor que 2”	system.out. println (“No es mayor que 2”);
FINSI	

7. Instrucción “if...else...”.

- **CONSEJO:** Java te aconseja que uses las llaves { } para englobar en bloques las instrucciones que se ejecutarán si se cumple la condición del *if* o del *else*.

Si NO usas las llaves, si se evalúa a cierto la condición del *if* o del *else*, se ejecutará solamente la primera instrucción que siga al if o al else.

Ejemplo:

```
if(temperatura>10)
    temperatura=20;
System.out.println(temperatura);
```

La instrucción *temperatura=20*, sólo se ejecutará si se cumple la condición *temperatura>10*, y el *System* siempre se ejecutará, independientemente que se cumpla o no la condición *temperatura>10*, incluso de que esté tabulado.

7. Instrucción “*if...else...*”.

- Una sentencia *if* evaluará primero su condición, y
 - Si el *resultado es true* ejecutara la sentencia o bloque de sentencias que se encuentre justo después de la condición.
 - Si la condición se evalúa como *false* y si existe una cláusula *else*, entonces se ejecutara la sentencia o bloque de sentencias que siguen al *else*. Esta parte es opcional. Si NO existe una cláusula *else* ejecutará la siguiente instrucción después del bloque de instrucciones que engloba el *if*.
- La única restricción que se impone a la expresión condicional es que sea una expresión válida de tipo booleano que se evalúe como *true* o como *false*.

7. Instrucción “*if...else...*”.

- Ejemplo 1: Código que elige el mayor de dos enteros: (NO es ejecutable, faltaría declaración de la clase y el *main*).

```
int a=3, b=5, m;  
if (a<b)  
    m=b;  
else  
    m=a;
```

7. Instrucción “if...else...”.

- Ejemplo 2: Código con una instrucción condicional sin parte *else*, donde se imprime si un numero es par o impar: (NO es ejecutable, faltaría declaración de la clase y el main).

```
int a=3;  
String mensaje="El numero " + a + " es ";  
if (a%2 != 0)  
    mensaje+="im";  
mensaje+="par. ";  
System.out.println(mensaje);
```

7. Instrucción “*if...else...*”.

- Ejemplo 2: Código con una instrucción condicional sin parte *else*, donde se imprime si un numero es par o impar: (NO es ejecutable, faltaría declaración de la clase y el main). (continuación)
- **Producido el resultado :**
 - El numero 3 es impar. (Por pantalla)
 - El numero 4 es par. (Por pantalla)
 - Dependiendo de si a es inicializada a 3 o a 4.

7. Instrucción “*if...else...*”.

● Reglas para el *if*

1. La condición va entre paréntesis
2. La línea del *if* y *else* NO lleva punto y coma (;)
3. El IF y el ELSE se ejecuta para una sola sentencia, la que le sigue en el programa, la inmediatamente siguiente.
4. Si se necesita más de una sentencia se encierra entre llaves. Esto se llama bloque, comentado ya en la unidad didáctica 1.
5. NO es obligado poner un *else* si éste no se utiliza.
6. Tabula SIEMPRE todas aquellas instrucciones que estén dentro de un *if* o un *else*.

7. Instrucción “*if...else...*”.

Estructura “*else if*”

- Es posible construir una serie de comprobaciones uniendo un *if* a la cláusula *else* de un *if* anterior.

7. Instrucción “if...else...”.

Estructura “else if”

- Ejemplo:

```
int a = 4;  
if (a > 5){  
    System.out.println("La variable es mayor a 5");  
}else if(a == 5){  
    System.out.println("La variable es igual a 5");  
}else{  
    System.out.println("La variable es menor que 5");  
}
```

7. Instrucción “if...else...”.

Estructura “else if”

- **IMPORTANTE:** En este tipo de secuencias de sentencias *if-else* anidadas, las cláusulas *else* siempre se emparejan con el *if* más cercano sin *else*. Para variar este comportamiento se deben utilizar llaves para crear bloques de sentencias.

7. Instrucción “if...else...”.

- “*if* anidados
- Ejemplo:

```
int matematicas = 4, lengua = 2;
if (matematicas >= 5) {
    if (lengua >= 5) {
        System.out.println("Enhorabuena");
    } else{
        System.out.println("No has aprobado todas las asignaturas");
    }
} else{
    System.out.println("No has aprobado todas las asignaturas");
}
```

8. Instrucción “switch”

- En la resolución de muchos problemas surge la necesidad de efectuar cálculos diferentes para distintos valores de una única variable o expresión simple.
- Por ejemplo, alguna variable de entrada tiene 10 posibles valores distintos, y para cada uno de los mismos es necesario un tratamiento especial, se podría utilizar una instrucción "if...else..." anidada; sin embargo ésta suele ser difícil de leer y mantener, por ello Java introduce una instrucción condicional adicional con el objetivo de facilitar el tratamiento de casos como el anterior.

8. Instrucción “switch”

- Esta sentencia examina los valores de una variable y en base a ello ejecuta sentencias independientes.

8. Instrucción “switch”

- Es la instrucción "switch", que tiene la siguiente forma general:

```
switch (expresion) {  
    case val1: [SC1] [break;]  
    case val2: [SC2] [break;]  
    .....  
    .....  
    case valn: [SCn] [break;]  
    [default: [SCn+1] ]  
}
```

- Aquellas componentes que aparecen entre corchetes indican opcionalidad (pueden o no aparecer).
- expresión** es una expresión de un tipo simple cualquiera. A partir de la API 7 se permite que sea también una cadena de texto *String*.
- La serie de valores **val1**, **val2**, ... **valn**, son todos valores compatibles con el de **expresión**. **SC1**, **SC2**, ... **SCn+1**, son instrucciones (o secuencias de instrucciones) cualesquiera.
- Se abre una llave para iniciar los posibles valores que pueda tomar dicha variable.

8. Instrucción “switch”

- La ejecución de una instrucción **switch** es como sigue:
 - Se evalúa en primer lugar la expresión, comparándose el valor resultante con el de cada uno de los valores asociados a las etiquetas **case**, si coincide con alguno entonces se ejecuta todo el código que sigue a la etiqueta **case** correspondiente, incluso el asociado a las etiquetas **case** posteriores, hasta que, o bien se finaliza toda la instrucción **switch**, o bien se encuentra una instrucción **break**, en cuyo caso toda la instrucción **switch** se acaba inmediatamente.
 - Si ninguno de los valores coincide con el de la expresión entonces se ejecuta, si existe (es opcional), la instrucción o bloque asociado a la etiqueta **default**. Una vez finalizada la ejecución del **switch**, el programa continúa a partir de la instrucción siguiente.

8. Instrucción “switch”

- Ejemplo:

```
public class EjemploSwitch {  
    public static void main (String[] args) {  
        int mes = 3;  
        switch (mes) {  
            case 1:  
                {System.out.println("Enero");  
                break;  
                }  
            case 2:  
                { System.out.println("Febrero");  
                break;  
                }  
            case 3:  
                { System.out.println("Marzo");  
                break;  
                }  
  
                // Resto de meses  
  
            default:  
                {System.out.println("Mes inexistente");  
                break;  
                }  
        }  
    }  
}
```

8. Instrucción “switch”

- NOTA: El programa se va ejecutando hasta acabar el **switch** o encontrar un **break**. Si entra en un caso y no encuentra un **break**, realizará las instrucciones de los **case** que tiene por debajo de él (incluso lo que haya en el **default**, si lo hay) hasta encontrar un **break** o acabar el **switch**.
- **Ten en cuenta:**
 - Dentro del CASE no se necesitan llaves.
 - El programa entra en el DEFAULT si no entró en otra opción.
 - El BREAK hace que el CASE se termine y con ello el SWITCH entero.

8. Instrucción “switch”

- Ejemplo:

```
1 ▼ public class PruebaSwitch{  
2 ▼   public static void main(String args[]){  
3     int paso=1;  
4     switch( paso )  
5     {  
6       case 1: System.out.println("Paso 1 (ponerse cómodo) sin finalizar. ");  
7       case 2: System.out.println("Paso 2 (regular espejos) sin finalizar. ");  
8       case 3: System.out.println("Paso 3 (abrochar cinturón) sin finalizar. ");  
9       case 4: System.out.println("Paso 4 (arrancar motor) sin finalizar. ");  
10      }  
11    }  
12 }
```

- En este caso, si `paso==1`, entrará en el `case 1`, y hará también todos los demás casos, ya que no hay un *break*.
- Si `paso==3`, haría su propio caso, `case 3`, y al no haber un *break*, realizaría también los casos por debajo de él, hasta encontrar un *break*, en este caso también realizaría `case 4`.

9. Bucles.

Los bucles son bloques de código que se ejecutan repetidamente mientras se cumpla una condición booleana.

- **Tipos de bucles:**

- for
- while
- do-while

9.1. Bucle *for*.

Permite implementar la repetición de un cierto conjunto de instrucciones un número de veces determinado o indeterminado pero con una finalización a través de una condición.

- Se utiliza una **variable de control** del bucle, llamada índice, que va recorriendo un conjunto prefijado de valores en un orden determinado. Para cada valor del índice en dicho conjunto, se ejecuta una vez el mismo conjunto de instrucciones incluidas en el bloque.
- Cada vez que se pasa por el bucle, el índice tendrá un valor diferente.

Sintaxis:

```
for (expresión_inicialización; expresión_finalización, expresión de iteración)
{
    instrucciones
}
```

9.1. Bucle for.

```
public class BucleFor
{
    public static void main (String args[]) {
        for (int i = 0; i < 5; i++)
        {
            System.out.println("Esta es la repetición número: " + i );
        }
    }
}
```

Funcionamiento:

**La primera ejecución del bucle ejecuta la inicialización:
int i = 0. Sólo la primera vez. Después:**

- 1 Se evalúa la condición: $i < 5$**
- 2 Si la condición es true se ejecuta el código del bloque. Si es false termina el bucle y continúa en la siguiente instrucción.**
- 3 Despues de ejecutar el código del bloque se realiza el incremento; $i++$**
- 4 Se vuelve al paso 1.**

9. El bucle *while*

Permite implementar la repetición de un mismo conjunto de instrucciones mientras que se verifique una determinada condición al comienzo de cada ciclo.

```
while (condicion){  
    instrucciones  
}
```

Su funcionamiento es:

Al comienzo de cada iteración se evalúa la expresión lógica:

- a Si el resultado es VERDADERO, se ejecuta el conjunto de instrucciones y se vuelve a iterar, es decir, se repite el paso 1. Así hasta que no se cumpla la condición.
- b Si el resultado es FALSO, se detiene la ejecución del ciclo WHILE y el programa se sigue ejecutando por la instrucción siguiente al }.

9. El bucle *while*

```
1 import java.util.*;
2
3 public class BucheWhile
4 {
5     public static void main (String args[])
6     {
7         Scanner entradaConsola = new Scanner(System.in);
8         int numero = 1;
9
10        while (numero != 0) {
11            System.out.println("Introduce un numero entero:(0 para salir) ");
12            numero = entradaConsola.nextInt();
13        }
14        entradaConsola.close();
15    }
16
17 }
18 }
```

Si la condición planteada inicialmente no se cumple, las instrucciones que contiene el while no se ejecutan y pasa a ejecutarse el siguiente grupo de instrucciones. Por lo tanto, **puede llegar a no ejecutarse** el bloque de instrucciones. Así que hay que tener cuidado al usarlo.

9. El bucle do...while

Permite implementar la repetición de un mismo conjunto de instrucciones mientras que se verifique una determinada condición después de realizarse el ciclo.

```
do{  
    instrucciones  
} while (condicion)
```

Su funcionamiento es:

Se ejecutan las instrucciones del bloque.

Al final de cada iteración se evalúa la expresión lógica:

Si el resultado es VERDADERO, se ejecuta el conjunto de instrucciones y se vuelve a evaluar la expresión lógica.

Si el resultado es FALSO, se detiene la ejecución del ciclo y el programa se sigue ejecutando por la instrucción siguiente al }.

9. El bucle *do...while*

```
import java.util.*;  
  
public class BucheWhile  
{  
    public static void main (String args[])  
    {  
        Scanner entradaConsola = new Scanner(System.in);  
        int numero;  
  
        do {  
            System.out.println("Introduce un numero entero:(0 para salir) ");  
            numero = entradaConsola.nextInt();  
        } while (numero != 0);  
        entradaConsola.close();  
    }  
}
```

Primero se ejecutan las instrucciones y después de realiza la comprobación de la condición. Dicho de otro modo, **se pasa al menos una vez** por las instrucciones que contiene antes de salir del bucle.