

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Tema 2 – Estructuras de control

En este tema vamos a ver una serie de estructuras que permiten desviar el avance secuencial del programa dependiendo si se cumplen o no ciertas condiciones.



Para tomar decisiones es necesario saber crear condiciones cuyo resultado será Verdadero o Falso, y dependiendo de ese resultado se ejecutarán distintas partes del código. Las condiciones se crean con los operadores de relación y los operadores lógicos. Estos establecen una operación cuyo resultado al final es verdadero o falso. Entendamos esto con un ejemplo:

```
import java.util.*;

public class Positivo {
    public static void main(String[] args) {
        Scanner sc = new Scanner (System.in);
        int numero;

        System.out.print("Dime un número entero: ");
        numero = sc.nextInt();
        System.out.println();
        if (numero>0) {
            System.out.println("El número es positivo");
        }
    }
}
```

En el caso anterior hay una sentencia if que funciona así: en el caso de que la condición dentro del paréntesis sea verdadera, se ejecuta lo que hay dentro de las llaves. Si es falsa, no se ejecuta.

Entonces hay que aprender a entender y realizar estas condiciones, por lo que empezaremos a ver operaciones lógicas, que en lugar de ser como las aritméticas, cuyo resultado es numérico, en las lógicas el resultado siempre es true o false (verdadero o falso).

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Operadores de relación

Son operadores que permiten realizar comparaciones entre valores que pueden estar guardados en variables.

```

== Igualdad
< Menor que
<= Menor o igual que
> Mayor que
>= Mayor o igual que
!= Distinto de

```

El resultado de estas operaciones será *true* o *false*, es decir, un valor booleano.

Veamos el resultado de algunas relaciones:

```

int entero=10;
double real=3.5;;
char caracter='A';
System.out.println(entero+5!=15);
System.out.println(real>=Math.PI);
System.out.println(caracter>='a');

```

Las comparaciones numéricas no tienen mayor complicación. Con los caracteres hay que tener cuidado ya que lo que se compara es, internamente, su código Unicode. Así resultará que cualquier mayúscula siempre será menor que cualquier minúscula.

Observa algunos códigos ascii/unicode en la siguiente tabla.

32:	51:3	70:F	89:Y	108:l
33:!	52:4	71:G	90:Z	109:m
34:"	53:5	72:H	91:[110:n
35:#	54:6	73:I	92:\	111:o
36:\$	55:7	74:J	93:]	112:p
37:%	56:8	75:K	94:^^	113:q
38:&	57:9	76:L	95:	114:r
39:'	58::	77:M	96:~	115:s
40:(59:;	78:N	97:a	116:t
41:)	60:<	79:O	98:b	117:u
42:*	61:=	80:P	99:c	118:v
43:+	62:>	81:Q	100:d	119:w
44:,	63:?	82:R	101:e	120:x
45:-	64:@	83:S	102:f	121:y
46:.	65:A	84:T	103:g	122:z
47:/	66:B	85:U	104:h	123:{
48:0	67:C	86:V	105:i	124:
49:1	68:D	87:W	106:j	125:}
50:2	69:E	88:X	107:k	126:~


En cuanto a la comparación de cadenas no se puede hacer directamente con los operadores si no que ha de usarse una función contenida en cada cadena y se usa de la siguiente forma:

```

String nombre="Curro";
System.out.println(nombre.equals("CURRO"));

```

Nombre_de_la_cadena.equals(cadena a comparar) compara las cadenas y se convierte en *true* o *false* dependiendo de la igualdad de las mismas.

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Operadores lógicos

Son los que realizan la lógica clásica: y, or, not.

`&&` → El resultado es *true* sólo si los dos operandos son *true* (Y lógico).

`||` → El resultado es *true* si al menos uno de los dos operandos es *true* (O lógica).

`!` → Es unario, quiere decir que sólo actúa sobre un operando. Lo que hace es invertirlo: si es *true* lo pasa a *false* y si es *false* lo pasa a *true*. (NOT lógica).

Actividad: Indica el resultado de las siguientes expresiones lógicas.

Si $A=4$ y $B=10$

$A!=B \ || \ A<0$

$A==4 \ \&\& \ B==10$

$A==4 \ \&\& \ B<10$


$!(A>B) \ || \ (B==11)$

A la hora de trabajar con distintos operadores hay que tener en cuenta la prioridad para usar o no paréntesis. En cualquier caso, desde la Web de Oracle se recomienda el uso de paréntesis aunque sea por claridad de código.

La prioridad es la siguiente:

<code>()</code>	Paréntesis
<code>! ~ %</code>	Not lógico y de bit
<code>* /</code>	Multiplicativos
<code>+ -</code>	Aditivos
<code><< >></code>	Desplazamiento de bits
<code>< > <= >=</code>	Relacionales
<code>== !=</code>	Igualdad
<code>&</code>	AND entre bits
<code>^</code>	XOR entre bits
<code> </code>	OR entre bits
<code>&&</code>	AND lógico
<code> </code>	OR lógico
<code>=</code>	Asignación

*Nota: Las operaciones lógicas "entre bits" tienen un resultado numérico y no booleano. Es decir, mientras el `&&` hace una operación entre valores lógicos y resulta en un *true* o *false* (Por ejemplo $a>2 \ \&\& \ b!=0$), `&` realiza una operación entre bits de dos números. Por ejemplo $0xFA \ \& \ 0x07$, que daría $0x02$)*

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Actividades:

1. Da el resultado de las siguientes expresiones lógicas sabiendo que las variables implicadas valen:
 $a = 10$; $b = 12$; $c = 13$; $d = 10$;

- a) $(a > b \parallel a < c) \&\& (a == c \parallel a > = b)$
- b) $(a > = b \parallel a < d) \&\& (a > = d \&\& c > d)$
- c) $! (a == c) \&\& (c > b)$

2. Contesta a las preguntas que se plantean sobre las siguientes expresiones lógicas.

a) $! ('a' == 'z')$

¿Resultado?

b) $(mes == 10 \&\& dia > = 24) \parallel (mes == 11 \&\& dia < = 22)$


¿Para que valores es true?

c) $(x * 2 > y - 3 \parallel x > y - 1) \&\& y < 5$

¿Orden de evaluación?

d) $!p \parallel q \&\& r$

¿Que tipo de dato contienen p, q y r? ¿Orden de evaluación?

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Estructuras Selectivas

Son sentencias que permiten a partir de una decisión ejecutar unas líneas de código u otras, como el primer ejemplo del tema.

Estructura if-else

Estructura del if simple:

```
if (expresión_lógica){
    Acción1;
    Acción2;
    .....
    AcciónN
}
```

Estructura if-else (alternativa):

```
if (expresión_lógica){
    Acción1;
    Acción2;
    .....
    AcciónN;
} else {
    AcciónN+1;
    AcciónN+2;
    .....
    AcciónK;
}
```

En Java se admite que en la estructura if (y en otras) si solo se va a ejecutar un comando no son necesarias las llaves. Por ejemplo se podría poner:

```
if (expresión_lógica)
    Acción1;
o
if (expresión_lógica)
    Acción1;
else
    Acción2;
```

Sin embargo no es una práctica que se recomienda al programar en Java. Por un lado es norma de estilo recomendado por Oracle, y por otro el uso de llaves disminuye la aparición de errores. Por ejemplo:

```
int a=0;
if (a>=0)
    System.out.println("Dentro del if");
    System.out.println("¿Dentro del if?");
```

Ejecuta el código anterior y luego mete los *System.out.println* entre llaves.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Se pueden meter unas sentencias if dentro de otras. Por ejemplo el siguiente programa, que amplía la funcionalidad del primer ejemplo, es válido:

```
import java.util.*;

public class Positivo {
    public static void main (String[] args) {
        Scanner sc = new Scanner(System.in);
        int numero;
        System.out.println("Introduce un número");
        numero=sc.nextInt();

        if (numero>0) {
            System.out.println("Numero positivo");
        } else {
            if (numero==0) {
                System.out.println("Numero igual a 0");
            } else {
                System.out.println("Numero negativo");
            } // Fin del if (numero==0)
        } // Fin del if (numero>0)
        System.out.println("Esta línea siempre se ejecuta");
    }
}
```

Cuando se anidan sentencias, lo importante es tener en cuenta que siempre la última en abrirse es la primera en cerrarse.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Sentencia Switch

Permite tomar una decisión entre varias dependiendo del valor de una variable o expresión. El valor del que depende la decisión puede ser *byte*, *short*, *char* o *int*. Desde Java 7 también puede usarse una String.

```
switch(variable) {
    case valor1:    AccionesValor1;
                    break;
    case valor2:    AccionesValor2;
                    break;
    ...
    case valorN:    AccionesValorN;
                    break;
    default:        AccionesEnOtrosCasos; // apartado opcional
                    break; // Opcional
}
```

Veamos un ejemplo:

```
Scanner sc = new Scanner(System.in);
int numeroMes;
String cadenaMes;
System.out.println("Introduce un número de mes (1-12)");
numeroMes=sc.nextInt();
switch (numeroMes) {
    case 1: cadenaMes = "Enero";
    break;
    case 2: cadenaMes = "Febrero";
    break;
    case 3: cadenaMes = "Marzo";
    break;
    case 4: cadenaMes = "Abril";
    break;
    case 5: cadenaMes = "Mayo";
    break;
    case 6: cadenaMes = "Junio";
    break;
    case 7: cadenaMes = "Julio";
    break;
    case 8: cadenaMes = "Agosto";
    break;
    case 9: cadenaMes = "Septiembre";
    break;
    case 10: cadenaMes = "Octubre";
    break;
    case 11: cadenaMes = "Noviembre";
    break;
    case 12: cadenaMes = "Diciembre";
    break;
    default: cadenaMes = "Mes inválido";
    break;
}
System.out.println(cadenaMes);
```

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Hay que tener en cuenta los siguientes puntos:

- Una vez que se llega al *switch*, se evalúa la expresión y se salta al caso que corresponda.
- Lo que se puede hacer con un *switch* se puede hacer con múltiples *if*. Queda a decisión del programador cual es la estructura más clara u óptima para cada caso.
- El comando *break* implica que no sigue ejecutándose el *switch*, por lo que si se quita se ejecutarían todas las opciones posteriores a la del salto. Prueba a quitar todos los *breaks* salvo Diciembre en el programa anterior y pruébalo.
- El apartado default es opcional, se suele poner siempre por claridad aunque esté vacío ya que en un futuro puede interesar completarlo.
- También es opcional el último *break*.
- No se puede realizar switch con cualquier variable, sólo *byte*, *short*, *char* o *int*.
- Desde Java 7 se puede usar también *String*.


Veamos un ejemplo clásico de uso del *switch*: la realización de un menú de opciones. Planteo aquí solamente el esqueleto (Puedes ver una variante con códigos ANSI en el apéndice 2).

```
Scanner sc = new Scanner(System.in);
int opcion;

System.out.println("Menú de opciones");
System.out.println("-----");
System.out.println("1.- Suma");
System.out.println("2.- Resta");
System.out.println("3.- Multiplicación");
System.out.println("4.- División\n\n");
System.out.println("Teclee opción (1-4)");
opcion = sc.nextInt();


switch (opcion) {
    case 1: // Subprograma de suma
        break;
    case 2: // Subprograma de resta
        break;
    case 3: // Subprograma de multiplicación
        break;
    case 4: // Subprograma de división
        break;
    default:
        System.out.println("Opción no válida.\nEl programa finalizará.");
        break;
}

System.out.println("Hasta otra!");
```


	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Actividades:

1. *Realiza un programa que calcule el seguro de un coche sabiendo que el precio base es de 300€ y que se incrementa 100€ para menores de 21 y hay un bonus de -30€ si el conductor tiene más de 10 años de carné. Se le pedirá al usuario su edad y los años de carné y el programa le informará del precio final del seguro.*
(Opcional) *Haz el programa más eficiente de forma que compruebe que si el conductor es menor de 28 años ya no le pregunte por los años de carné.*
2. *Realiza un programa que pida dos números al usuario y los divida, pero si el divisor es cero que informe del problema al usuario y no realice la división.*
3. *Empieza con el boletín de tema. Con lo visto podrías hacer los ejercicios 2, 8 y 10.*

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Estructuras Iterativas (Bucles)

Es un proceso en el que se ejecutan una serie de operaciones un número determinado de veces. Las operaciones son siempre las mismas pero cambia el valor de los datos en cada repetición.

Para realizar un bucle o lazo, se necesita una condición de salida del mismo. Dicha condición debe estar asociada a alguna variable que cambie dentro de la estructura repetitiva.

Ejemplo: *Una persona se quiere comprar un nuevo ordenador. Tiene elegido el modelo pero no tiene intención de pagar más de 900€. Actualmente el precio es más caro, por lo que esta persona va cada 3 días a la tienda y si no ha bajado de precio el ordenador, no lo compra.*

Un bucle tiene 3 partes principales:

- *Inicialización (antes del bucle en sí mismo)* : Inicializa las variables que se usarán dentro del bucle
- *Cuerpo del bucle y modificación*: Sentencias internas al bucle y que van a cambiar los valores de una o más variables.
- *Condición de salida del bucle*: Condición que valora una o más variables y toma la decisión de salir del bucle.

Dentro de los bucles es muy habitual usar dos tipos de variables de control del bucle:

Contador: Variable que se usa dentro de un bucle para contar el número de veces que sucede algo. Suelen tener la forma:

$N=N+K$ donde K es un número fijo (habitualmente 1)

Un contador es necesario porque necesitamos contar la cantidad de veces que sucede algo, por ejemplo, la cantidad de veces que se ha ejecutado el bucle. Es necesario saberlo para saber también cuando hay que parar. Si yo quiero que algo se repita 100 veces, necesito contar en una variable de 1 a 100.

Acumulador: Es una variable que guarda el resultado temporal de una operación que acaba cuando acaba el bucle. Dentro del bucle se usa así:

$N=N+S$

Donde S es una cantidad variable resultado de alguna expresión dentro de un bucle.

El acumulador es necesario porque en algunas ocasiones tengo que repetir una operación de forma sucesiva. Para llegar al resultado final. Por ejemplo si tengo que sumar $1+2+3+4+5+6+...$ lo que hago en cada vuelta es una suma parcial y guardo en el acumulador dicho resultado para seguir en la siguiente vuelta. Así en la primera vuelta sumo $1+2$, en la segunda sumo $3+3$ (el primer 3 es el $1+2$ de la primera vuelta), en la tercera sumo $6+4$ (el 6 viene de $1+2+3$), en la cuarta $10+5$ (el $10=1+2+3+4$) y así sucesivamente. Ese resultado parcial lo guardo en el acumulador.

Puedes usar un acumulador para una suma, para una multiplicación o incluso para concatenar una cadena.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Tanto el contador como el acumulador son necesarios inicializarlos **antes** del bucle.

Por ejemplo si tenemos una variable *cont* inicializada a 0 y de forma repetitiva ejecutamos la sentencia:

```
cont=cont+1;
```

en cada iteración *cont* se va incrementando de 1 en 1. Toma por tanto los valores: 0,1,2,....

En el caso del acumulador, en cada iteración se suman números distintos. Por ejemplo si tenemos sendas variables *cont* y *acu* inicializadas a 0 y repetimos continuamente las siguientes líneas.

```
cont=cont+1;
acu=acu+con;
```

Actividad: ¿Qué sucede? Indica el valor de ambas en cada repetición.

Operadores de asignación

En Java y en general en lenguajes derivados del C existe una forma compacta de escribir ciertas operaciones en las que la variable sobre la que se realiza la asignación forma parte también de la operación como ocurre en el caso de contadores y acumuladores. De esta forma las líneas de código anteriores en Java se pueden (y deben) escribir de la siguiente forma:

```
cont++; //añade 1 a cont
acu+=con; //añade con a acu
```

Veamos los operadores:

```
+=   añade y asigna números, concatena y asigna Strings
-=   resta y asigna
*=   multiplica y asigna
/=   divide y asigna
%=   resto y asigna
++   incremento unidad
--   resta unidad
|=   Or a nivel de bit y asigna
^=   Xor a nivel de bit y asigna
&=   And a nivel de bit y asigna
>>= Desplazamiento de bits a la derecha y asignación
<<= Desplazamiento de bits a la izquierda y asignación
```

Actividad: Indica como se realiza:

- el conteo de números impares (indica dos formas).*
- múltiplos de cinco (indica dos formas).*
- decremento de uno en uno.*
- multiplicar de 4 en 4 (4*8*12*16*...)*

Pero todo esto no tiene sentido conocerlo sin saber como crear los bucles. Vemos estas estructuras:

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Estructura while.

Estructura que evalúa una condición y mientras dicha condición sea verdadera repite una serie de sentencias. El formato es:

```
while (condición) {
    Acciones;
}
```

Notas:

- Es necesario inicializar variables implicadas **antes** de entrar en el while.
- Dentro de las acciones de while debe existir alguna cambio que implique alguna variable de la condición (si no es así tenemos un bucle infinito).
- Las acciones del while pueden no ejecutarse si la condición no se cumple la primera vez.

Ejemplo:

```
int cont = 1;
while (cont < 11) {
    System.out.println("Cont = " + cont);
    cont++; //Es lo mismo que cont=cont+1;
}
```

Estructura do-while

Estructura que realiza una serie de acciones y las vuelve a realizar mientras se cumpla cierta condición.

```
do {
    Acciones;
} while(condición);
```

Notas:

- Es necesario inicializar variables implicadas **antes** de entrar en el while.
- Dentro de las acciones de while debe existir alguna cambio que implique alguna variable de la condición (si no es así tenemos un bucle infinito).
- Las acciones se ejecutan siempre al menos una vez. Esta es la diferencia con el while.

Ejemplo:

```
int cont = 1;
do {
    System.out.println("Cont = " + cont);
    cont++; //Es lo mismo que cont=cont+1;
}while (cont < 11);
```

Para ver la diferencia entre *while* y *do-while* prueba a inicializar cont a 20 y prueba ambos casos.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Nota: Dentro de un while pueden utilizarse las siguientes dos instrucciones especiales:

break; Indica que se ha de abortar la ejecución del bucle y continuarse ejecutando fuera del bucle.

continue; Indica que se ha de abortar la ejecución de las <Acciones> y reevaluarse la <condición> del bucle, volviéndose a ejecutar las <instrucciones> si es cierta o pasándose a ejecutar la instrucción siguiente al while si es falsa.

Pero no las usaremos por el momento hasta que tengamos un control más fuerte de los bucles.

Ejemplo: Repetición de la ejecución de un programa con menú (fragmento):

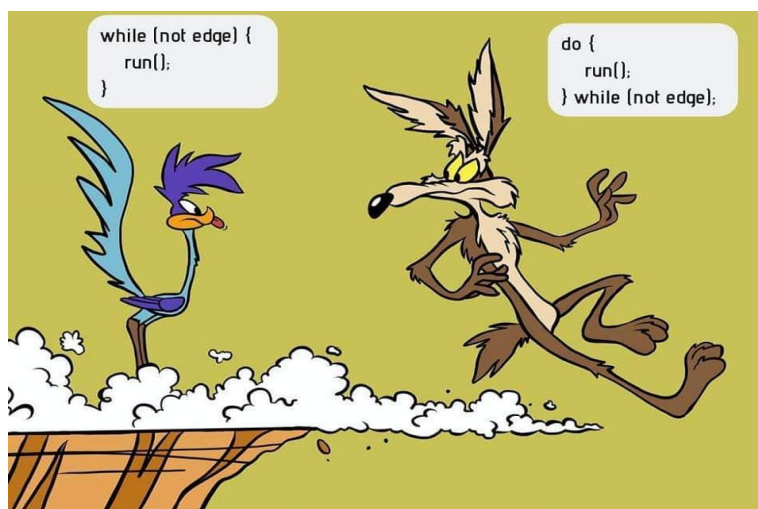
```
do {
    // println's con opciones de menú
    System.out.println("1.- Opción 1");
    System.out.println("2.- Opción 2");
    System.out.println("3.- Salir");
    System.out.println("Seleccione opción:");
    opcion = sc.nextInt();

    switch (opcion) {
        case 1: // Opción 1
            break;

        case 2: // Opción 2
            break;

        case 3:
            System.out.println("Hasta Pronto!");
            break;

        default:
            System.out.println("Opción no válida.");
            break;
    }
} while (opcion != 3);
```



COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Estructura for.

Este es un bucle idéntico al *while* pero escrito de forma compacta, lo que lo hace muy cómodo para un montón de tareas ya que en la primera línea vemos tanto la inicialización como la condición de permanencia como la modificación de la/s variable/s implicadas en la condición.

```
for (<inicialización>; <condición>; <modificación>) {
    instruccion;
}
```

En los tres bucles que hemos visto, si el cuerpo del bucle es de una única línea no son necesarias las llaves.

Ejemplo (completamente equivalente al ejemplo del while):

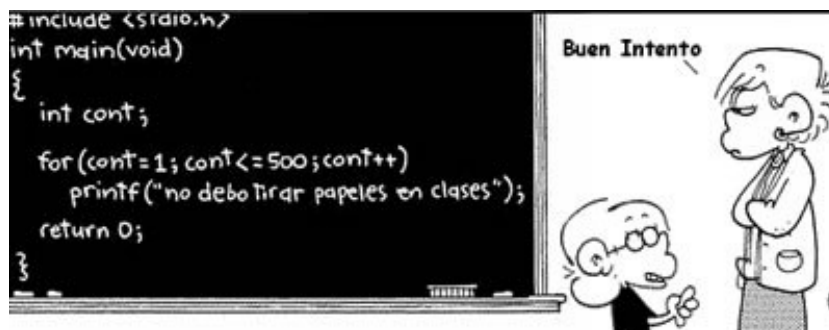
```
for (int cont=1; cont<11; cont++){
    System.out.println("Cont = "+cont);
}
```

Este ejemplo es totalmente equivalente al while anterior. Ojo, no es equivalente a un do-while porque el for, al igual que el while, puede que no ejecute nunca las instrucciones que contiene en el cuerpo.

Los bucles se pueden anidar. Si tenemos un bucle dentro de otro, hasta que acaba el interno no avanza el externo. Veamos un ejemplo:

```
System.out.println(" i j");
System.out.println("-----");
for (int i=10; i<13; i++) {
    for (int j=50; j<53; j++) {
        System.out.println(" "+i+" "+j);
    }
}
```

Por supuesto también se pueden anidar cualquier estructura de las vistas dentro de otras. Simplemente hay que tener cuidado con el uso de las llaves.



COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Conversión de tipos

En ocasiones es necesario realizar conversión de tipos de datos para guardar un dato en una variable distinta a la original. Por ejemplo puede darse el caso de tener un número como *String* y no podemos operar con él. En ese caso tenemos que convertirlo a entero o real para operar.

Actividad: Haz un programa en el que inicializas un String con el dato "231" y muestre ese dato más 5 ¿Cuál es el resultado?

Hagámoslo ahora con la conversión indicada:

```
String dato="231";
int num = Integer.parseInt(dato);
System.out.println(num+5);
```

Si lo que deseas es convertir unstring a nº real debes usar:

```
Double.parseDouble(str)
```

siendo *str* una variable o constante tipo *String*.

Para la conversión inversa (numérico a cadena) llega con sumar la cadena vacía al número o variable numérica.

```
dato="" + num;
```

Para conversiones entre tipos numéricos no se usan comandos si no una técnica denominada **casting** que consiste en indicar a que tipo de dato se convierte en la asignación. Veamos un ejemplo:

```
double n1=12.65;
int n2;
n2 = n1;
System.out.println(n1+"\n"+n2);
```

si pruebas lo anterior da un error. Sin embargo cambiando la tercera línea con un casting a entero se soluciona:

```
n2=(int)n1;
```

Entrada de datos (nuevamente)

Teniendo esto en cuenta, otra posible solución al problema del `nextLine` tras un `nextInt`, otra solución es la siguiente:

Leer todo con `nextLine()` para recoger el retorno de carro y luego hacer la conversión con `parseInt()`.

```
System.out.println("Introduce un numero");
num = Integer.parseInt(sc.nextLine()); // lee cadena y lo convierte a integer
System.out.println("Introduce un nombre");
nombre = sc.nextLine();
System.out.println("Nombre: " + nombre + "\nNúmero: " + num);
```

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Salida de datos con formato (printf)

En ocasiones se necesita dar un formato especial a los datos de salida. Por ejemplo puede interesar mostrar solo dos decimales de cierto número o que un dato ocupe un número determinado de casillas.

En este caso no nos valen las sentencias print/println si no que se usa la función *System.out.printf*.

Veámoslo con ejemplos:

```
System.out.println("Valores de Pi");
System.out.println("-----\n");
System.out.println("Pi sin formato: "+ Math.PI);
System.out.printf("El numero Pi es %.2f y su doble es %.4f\n", Math.PI, 2 * Math.PI);
```

Con %f indicamos que se va a introducir en esa posición una variable real (float point). Al meterle un número con un punto entre % y f, se le indican los decimales. Así, al poner %.2f se le está diciendo que en esa posición de la cadena irá un número real representado con 2 decimales.

```
int n1 = 95;
System.out.printf("%d\n%4d\n%10d\n", n1, n1, n1);
System.out.printf("%04d\n%010d\n", n1, n1);
System.out.printf("%x\n", n1);
```

En este caso con %d insertamos un valor entero en sistema de numeración decimal. Si ponemos %x lo hacemos en hexadecimal. Cuando metemos un número entre el % y d (o x) se reservan espacios para escribir el número y se justifica a la derecha (p. ej. %4d escribe el 95 justificado a la derecha en 4 espacios).

Si al número tras el % se le antepone un 0, rellena los espacios con ceros.

Si el número fuera negativo, justificaría a la izquierda. Pruébalo.

Prueba también a usar X en vez de x.

Finalmente si juntamos ambos formateos con un número real, podemos reservar espacios al mismo tiempo que indicamos decimales. Por ejemplo:

```
System.out.printf("El numero Pi es\n%15.2f\n", Math.PI);
```

Algunos caracteres de formato que podemos encontrar son:

```
%f: Para valores reales.
%d: Para valores enteros en sistema de numeración base 10 (decimal)
%c: para char
%s: para String
%o: Muestra nº entero en base octal
%x: Muestra nº entero en base hexadecimal
```

Se pueden ver todos en:

<http://www.java2s.com/Code/JavaAPI/java.lang/System.out.printf.htm>

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Apéndice I: Operador condicional ternario ?:

Es una forma compacta de escribir una sentencia if.

Condición ? ValorA si true : valorB si false;

Es decir se evalúa la condición y el operador resulta ValorA si la condición es verdadera y ValorB si es falsa. Veamos un par de ejmplos:

```
System.out.println(numero >= 0 ? "Positivo ó 0" : "Negativo");
```

```
respuesta = valor == 5 ? 1 : -1;
```

```
System.out.println(n + " dia" + (n > 1 ? "s" : ""));
```

De todas formas es un operador que puede resultar complejo de leer por lo que debe ser usado en casos muy particulares.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Apéndice II: Menú con ANSI

Como curiosidad introduzco en este texto la posibilidad de usar códigos ANSI que soportan algunos terminales como los de Linux para realizar acciones clásicas para las que Java no tiene directamente un comando: Borrar la pantalla, cambio de tipo de letra, cambios de color, reposicionamiento del cursor, etc...


Este es el menú que se vio en el apartado del switch pero usando algunos de estos códigos como ejemplo:

```
import java.util.Scanner;

public class Menu {
    public static void main (String[] args) {
        Scanner sc = new Scanner(System.in);
        int opcion;
        // Los códigos ANSI no son soportados por todos los terminales
        // ANSI: Borra pantalla y va al inicio
        System.out.print("\u001b[2J\u001b[H");

        // ANSI: Negrita (o cambio de color)
        System.out.print("\u001b[1m");
        System.out.println("Menú de opciones");
        System.out.println("-----");

        // ANSI: Restaura a Normal
        System.out.print("\u001b[0m");
        System.out.println("1.- Suma");
        System.out.println("2.- Resta");
        System.out.println("3.- Multiplicación");
        System.out.println("4.- División\n\n");
        System.out.println("Teclee opción (1-4)");
        opcion=sc.nextInt();
        switch (opcion) {
            case 1: // Subprograma de suma
                break;
            case 2: // Subprograma de resta
                break;
            case 3: // Subprograma de multiplicación
                break;
            case 4: //Subprograma de división
                break;
            default: System.out.println("Opción no válida.\nEl programa finalizará.");
                break;
        }
        // ANSI: Video Inverso (Se invierte color fondo y letra)
        System.out.print("\u001b[7mHasta otra!\u001b[0m");
    }
}
```

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Cualquier código ANSI empieza por el código ASCII Escape (27 decimal, 1B hexadecimal) y a continuación el código que realiza cierta operación. Cuando decimos que se escriba un código ANSI, realmente lo que hacemos es mandar un comando a la consola para que cambie su comportamiento.

Por ser códigos extraños, para hacer más legible el programa es conveniente asignarlo a constantes de la siguiente forma:

```
public static final String ANSI_CLS = "\u001b[2J";
public static final String ANSI_HOME = "\u001b[H";
...
```

de forma que luego se pueden usar en una salida por pantalla así:

```
System.out.print(ANSI_CLS+ANSI_HOME);
```

lo que produce un código mucho más claro.

Si quieres informarte más sobre códigos ANSI busca en internet.

OJO: El terminal de **Windows NO** acepta códigos ANSI por defecto (se espera que lo haga en versiones nuevas)). Para que los acepte se debe ejecutar el siguiente comando de consola como administrador:

```
reg add HKEY_CURRENT_USER\Console /v VirtualTerminalLevel /t REG_DWORD /d 0x00000001 /f
```

Fuentes:

<http://download.oracle.com/javase/tutorial/java/TOC.html>

<http://www.rgagnon.com/javadetails/java-0047.html>

<http://www.cafeaulait.org/course/week2/index.html>

<https://www.codeproject.com/Tips/5255355/How-to-Put-Color-on-Windows-Console>