



UNIVERSIDAD DE ALMERÍA

Grado en Ingeniería Informática

Introducción a la Programación

2021-2022



Tema 1. Fundamentos de Programación

Estructuras de control selectivas o condicionales

Por defecto, las instrucciones de un programa se ejecutan **secuencialmente**

Sin embargo, al describir la resolución de un problema, es normal que tengamos que tener en cuenta condiciones que influyen sobre la secuencia de pasos que hay que dar para resolver el problema.

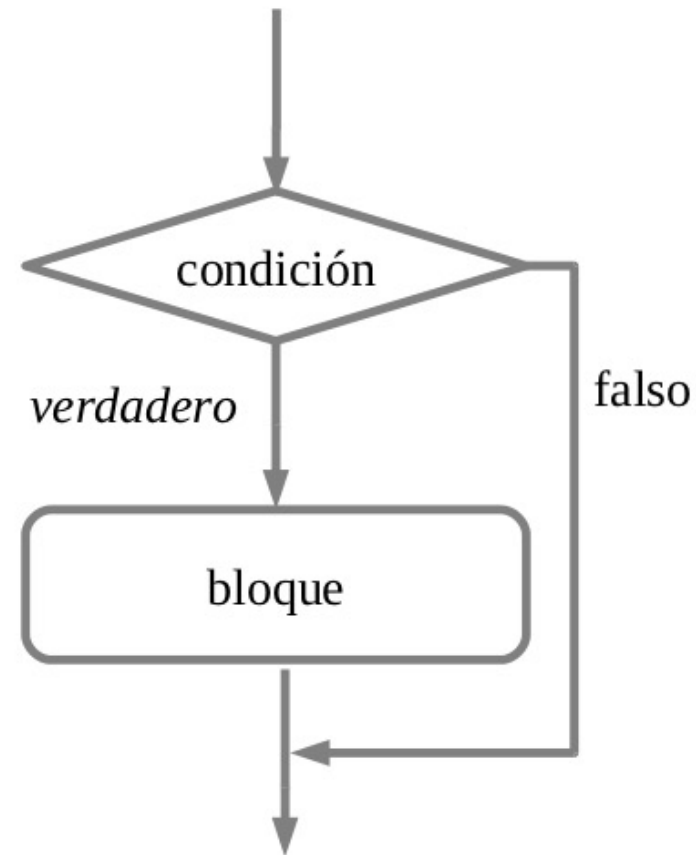
Las estructuras de control condicionales o selectivas nos permiten decidir qué ejecutar y qué no en un programa.

Estructuras de control selectivas

Sintaxis Selectiva Simple, cláusula `if`

```
if (condición)  
    sentencia;
```

```
if (condición) {  
    bloque  
}
```



donde ***bloque*** representa un *bloque de instrucciones*.

Bloque de instrucciones:

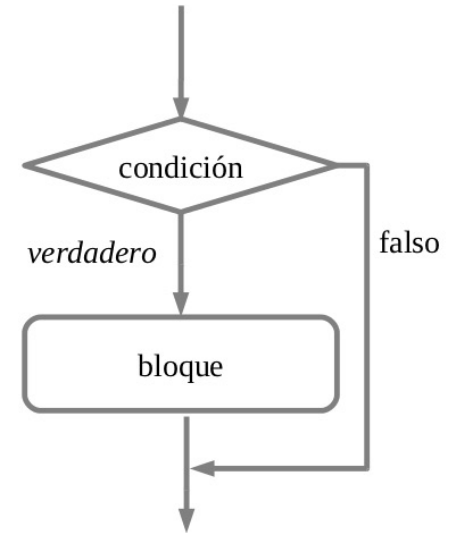
Secuencia de instrucciones encerradas entre dos llaves {...}

Estructuras de control selectivas

Ejemplos de uso selectiva o condicional simple:

```
if (nota >= 5)  
    System.out.println("APROBADO");
```

```
if (nota >= 5) {  
    System.out.println("APROBADO");  
    System.out.println("Recupera todo");  
}
```



Estructuras de control selectivas

Consideraciones acerca del uso de la sentencia **if**:

- **No olvidar los paréntesis** al poner la condición del **if**, es un error sintáctico (los paréntesis son necesarios)
- **No hay que confundir el operador de comparación == con el operador de asignación =**
- Los operadores relacionales: ==, !=, <= y >=, han de escribirse **sin espacios**.
- ==> y =< **no son operadores válidos en Java**.
- El fragmento de código afectado por la condición del **if** debe tabularse o indentarse para que visualmente se interprete correctamente el ámbito de la sentencia **if**:

```
if (condición) {  
    // Aquí se incluye el código  
    // que ha de ejecutarse  
    // cuando se cumple la condición del if  
}
```

Estructuras de control selectivas

➤ Aunque el uso de llaves no sea obligatorio cuando el **if** sólo afecta a una sentencia, es recomendable ponerlas siempre para delimitar explícitamente el ámbito de la sentencia **if**.

➤ **Error común:**

```
if (condición); ←  
    sentencia;
```

// es interpretado como

```
if (condición)
```

```
    ;           // Sentencia vacía
```

```
sentencia;
```

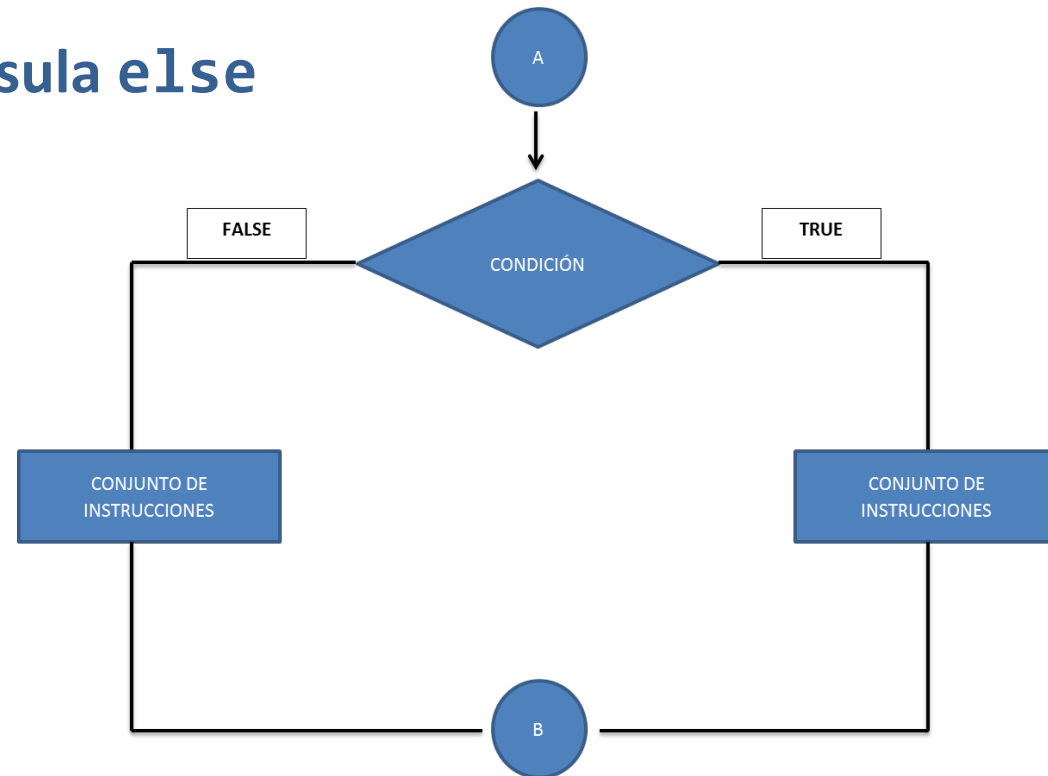
!!!La **sentencia** siempre se ejecutaría!!!

Estructuras de control selectivas

Sintaxis Selectiva Doble, cláusula `else`

```
if (condición)  
    sentencia1;  
else  
    sentencia2;
```

```
if (condición) {  
    bloque1;  
} else {  
    bloque2;  
}
```

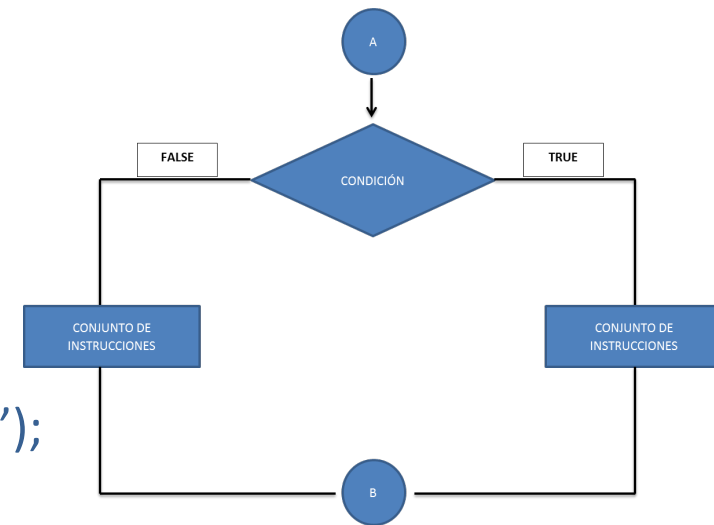


Estructuras de control selectivas

Ejemplos de uso selectiva o condicional doble

```
if (nota >= 5)
    System.out.println("APROBADO");
else
    System.out.println("SUSPENSO");
```

```
if (nota >= 5) {
    System.out.println("APROBADO");
    System.out.println("Recupera todo");
} else {
    System.out.println("SUSPENSO");
    System.out.println("No recupera");
}
```



Estructuras de control selectivas

Ejemplos de uso selectiva con operadores relacionales

```
if (nota < 0) || (nota > 10)
    System.out.println("NOTA NO VALIDA");
else
    System.out.println("VALIDA");
```

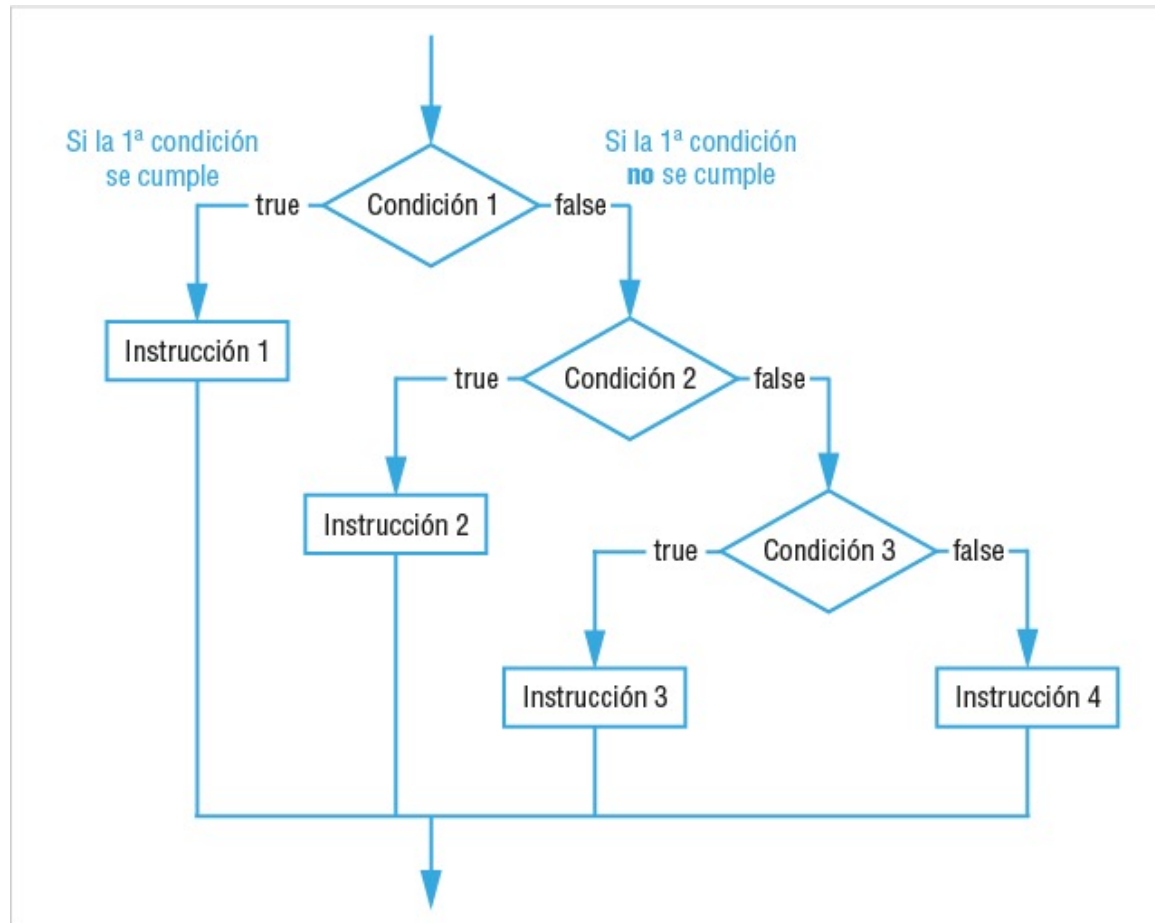
```
if (nota >= 0) && (nota <= 10) {
    System.out.println("VALIDA");
} else {
    System.out.println("NOTA NO VALIDA");
}
```

Estructuras de control selectivas

Sintaxis Selectiva Múltiple, cláusula `if ... else if ...`

```
if (condición1)
    sentencia1;
else if (condición2)
    sentencia2;
.....
else
    sentenciaN;
```

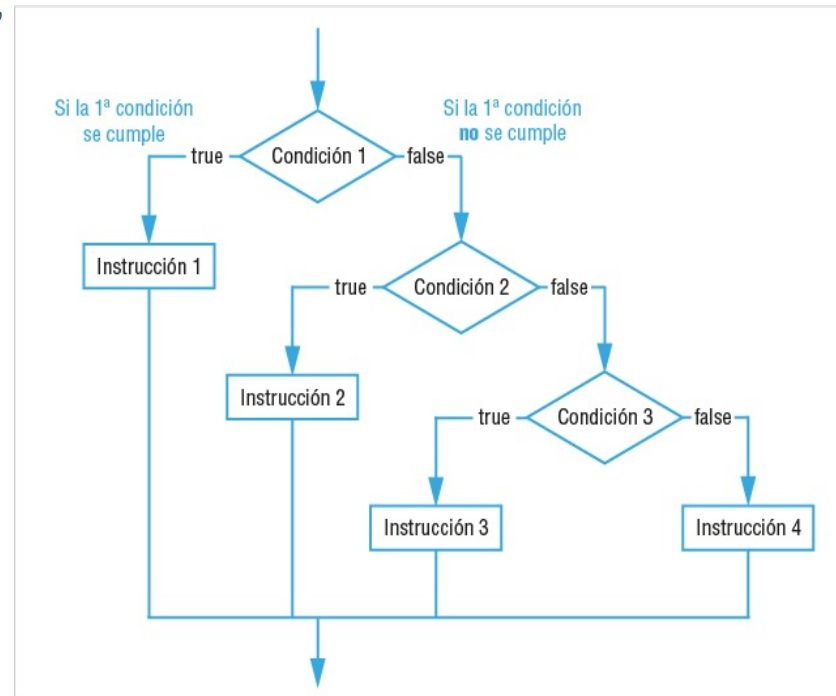
Idem con *bloques*



Estructuras de control selectivas

Ejemplo de uso selectiva múltiple

```
if (nota >= 9)
    resultado = "Sobresaliente";
else if (nota >= 7)
    resultado = "Notable";
else if (nota >= 5)
    resultado = "Aprobado";
else
    resultado = "Suspenso";
```



Se puede expresar con cuatro selectivas simples

Estructuras de control selectivas

Sintaxis de Anidamiento de Selectivas Doble

```
if (condición1) {  
    sentencia1;  
} else {  
    if (condición2) {  
        sentencia2;  
    } else {  
        sentencia3;  
    }  
}
```

Estructuras de control selectivas

Ejemplo de uso selectiva anidada

```
if (a != 0) {  
    x = -b / a;  
    resultado = "La solución es " + x;  
} else {  
    if (b != 0) {  
        resultado = "No tiene solución.";  
    } else {  
        resultado = "Solución indeterminada.";  
    }  
}
```

Se puede expresar con **tres selectivas simples**

Estructuras de control selectivas

Sintaxis del operador condicional: ? : (Operador ternario)

Java proporciona una forma de abreviar una sentencia **if**:
el **operador condicional** (operador ternario) **? :**

variable = condición ? expresión1 : expresión2;

equivale a

if (condición)

variable = expresión1;

else

variable = expresión2;

Estructuras de control selectivas

Ejemplo de uso del operador condicional ? :

$\text{max} = (x > y) ? x : y;$

$\text{min} = (x < y) ? x : y;$

```
if(x>y)
    max=x;
else
    max=y;
```

```
if(x<y)
    min=x;
else
    min=y;
```

Estructuras de control selectivas

<i>absolute value</i>	<code>if (x < 0) x = -x;</code>
<i>put x and y into sorted order</i>	<pre> if (x > y) { int t = x; x = y; y = t; </pre>
<i>maximum of x and y</i>	<pre> if (x > y) max = x; else max = y; </pre>
<i>error check for division operation</i>	<pre> if (den == 0) System.out.println("Division by zero"); else System.out.println("Quotient = " + num/den); </pre>
<i>error check for quadratic formula</i>	<pre> double discriminant = b*b - 4.0*c; if (discriminant < 0.0) { System.out.println("No real roots"); } else { System.out.println((-b + Math.sqrt(discriminant))/2.0); System.out.println((-b - Math.sqrt(discriminant))/2.0); } </pre>

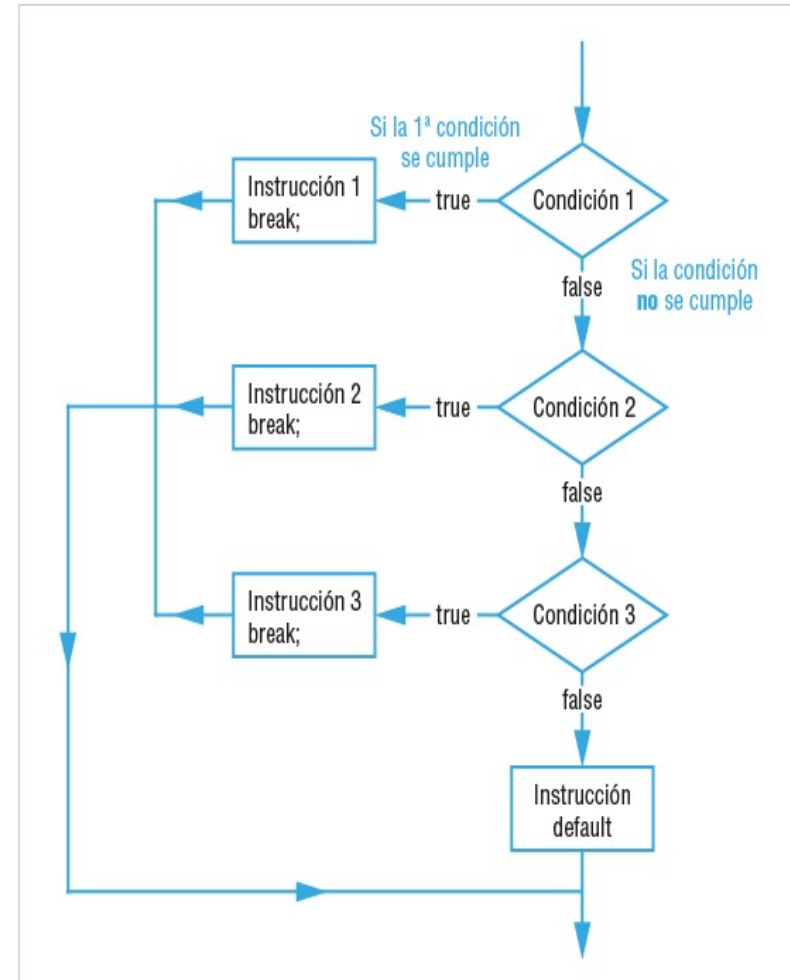
Sedgewick

Typical examples of using if statements

Estructuras de control selectivas

Sintaxis Selectiva Múltiple con la sentencia `switch`

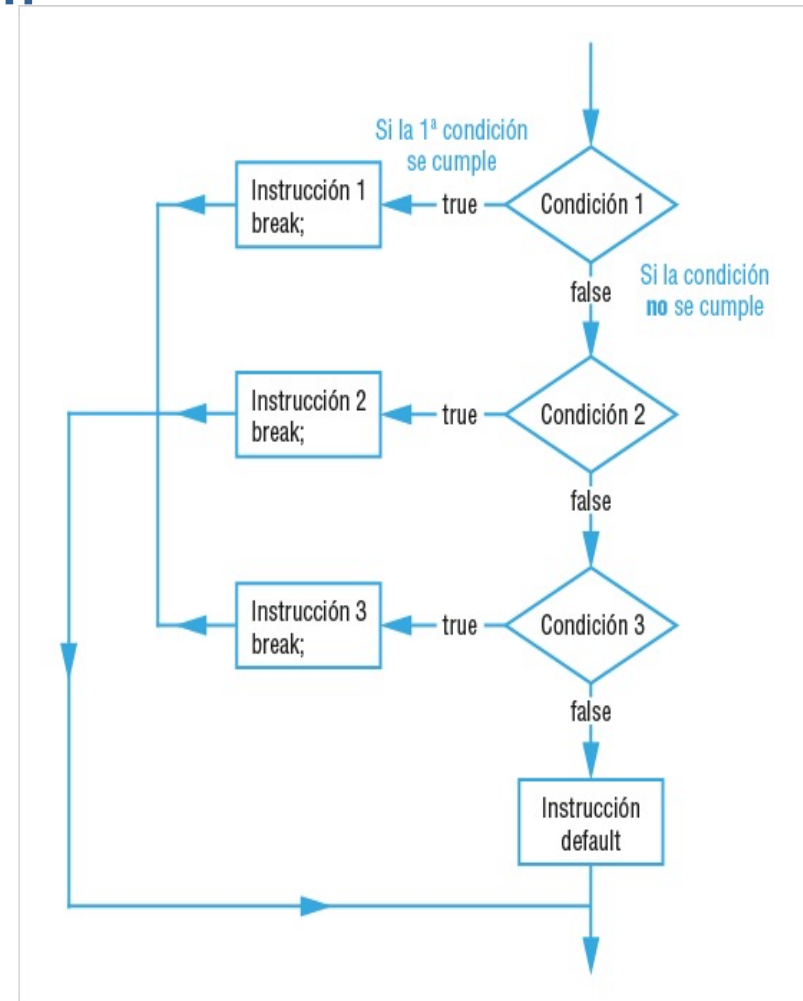
```
switch (expresión) {  
    case expr_cte1:  
        bloque1;  
        break;  
    case expr_cte2:  
        bloque2;  
        break;  
    ...  
    case expr_cteN:  
        bloqueN;  
        break;  
    default:  
        bloque_por_defecto;  
}
```



Estructuras de control selectivas

Ejemplo Selectiva Múltiple con switch

```
int diaSemana= 4;  
switch (diaSemana) {  
case 1:  
    System.out.println("LUNES");  
    break;  
case 2:  
    System.out.println("MARTES");  
    break;  
case 3:  
    System.out.println("MIERCOLES");  
    break;  
case 4:  
    System.out.println("JUEVES");  
    break;  
case 5:  
    System.out.println("VIERNES");  
    break;  
case 6:  
    System.out.println("SABADO");  
    break;  
case 7:  
    System.out.println("DOMINGO");  
}
```



Estructuras de control selectivas

Consideraciones al respecto

- ☐ Permite seleccionar entre varias alternativas posibles.
- ☐ Se selecciona a partir de la evaluación de una única expresión.
- ☐ La expresión del **switch** puede ser de tipo: **byte**, **short**, **char**, **int**, **String**, ...
- ☐ Los valores de cada caso del **switch** han de ser constantes.
- ☐ En Java, cada bloque de código de los que acompañan a un posible valor de la expresión entera ha de terminar con una sentencia **break**;
- ☐ La etiqueta **default** marca el bloque de código que se ejecuta por defecto (cuando al evaluar la expresión se obtiene un valor no especificado por los casos anteriores del **switch**). Es opcional.
- ☐ En Java, se pueden poner varias etiquetas seguidas acompañando a un único fragmento de código si el fragmento de código que ha de ejecutarse es el mismo para varios valores de la expresión entera que gobierna la ejecución del **switch**.

Estructuras de control selectivas

```
switch (nota) {  
    case 0:  
    case 1:  
    case 2:  
    case 3:  
    case 4:  
        resultado = "Suspenso";  
        break;  
    case 5:  
    case 6:  
        resultado = "Aprobado";  
        break;  
    case 7:  
    case 8:  
        resultado = "Notable";  
        break;  
    case 9:  
    case 10:  
        resultado = "Sobresaliente";  
        break;  
    default:  
        resultado = "Error";  
}
```

// CÓDIGO EQUIVALENTE

```
int nota=3;  
String resultado;  
if ( nota >= 9 )  
    resultado = "Sobresaliente";  
else if ( nota >= 7 )  
    resultado = "Notable";  
else if ( nota >= 5 )  
    resultado = "Aprobado";  
else if ( nota >= 0 )  
    resultado = "Suspenso";  
else  
    resultado = "Error";  
  
System.out.println("La calificacion es "+resultado);
```

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

Estructuras de control selectivas

```
int numeroDia=0;  
String dia = "jueves";
```

```
switch (dia) {  
    case "lunes":  
        numeroDia=1;  
        break;  
    case "martes":  
        numeroDia=2;  
        break;  
    case "miercoles":  
        numeroDia=3;  
        break;  
    case "jueves":  
        numeroDia=4;  
        break;  
    case "viernes":  
        numeroDia=5;  
        break;  
    case "sabado":  
        numeroDia=6;  
        break;  
    case "domingo":  
        numeroDia=7;  
        break;  
    default:  
        numeroDia=0;  
}  
System.out.print("El numero de dia es: " + numeroDia);
```

Tema 1. Fundamentos de Programación

➤ Estructuras de control repetitivas / iterativas / bucles

Permiten repetir una acción o conjunto de acciones un determinado número de veces. En ocasiones, ese número es conocido de antemano, pero en otras ocasiones, no.

En un bucle hay que tener en cuenta dos aspectos:

- **La condición de terminación** es una **expresión booleana** que indica cuándo se deja de ejecutar el bucle. Coloquialmente se dice que es la condición que me permite escapar o salir del bucle.
- **Cuerpo del bucle**, es la acción o conjunto de acciones que se repiten.

Estructuras de control repetitivas

- En un bucle se **repite** la ejecución de una sentencia o un bloque de sentencias siempre que la **condición de terminación sea verdadera** (`true`)
- Cada repetición de todas las instrucciones del cuerpo del bucle es una **iteración**.
- Al final de cada iteración se vuelve a evaluar la condición, y de ser **verdadera** (`true`), se ejecuta una nueva iteración.
- En el momento en el que la **condición sea falsa** (`false`), se dejan de ejecutar las sentencias y **se escapa o sale** del bucle.
- Tres tipos de bucles:
 - Bucle **while**
 - Bucle **for**
 - Bucle **do-while**

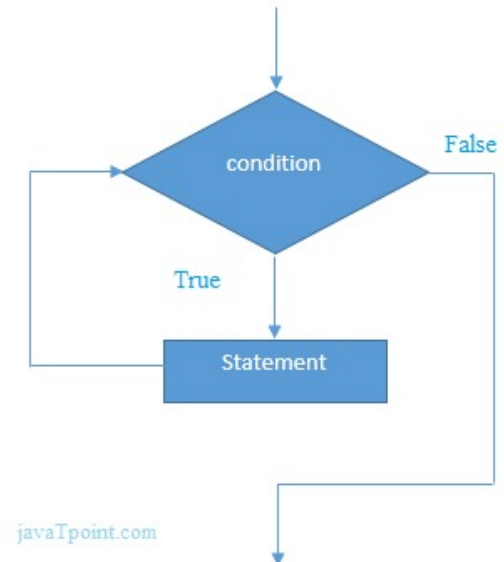
Estructuras de control repetitivas

El bucle **while**

- La condición de terminación se comprueba **antes** de comenzar la ejecución de las instrucciones. **Si es falsa** (`false`) NO hace nada.
- **Si es verdadera** (`true`) ejecuta todas las sentencias del cuerpo del bucle y cuando las termina vuelve a comprobar la condición.
- Se repiten los pasos hasta que la condición sea **falsa** (`false`).
- **Sintaxis:**

while (condición de continuación del bucle)
 sentencia; // Cuerpo del bucle

```
while (condición) {  
    bloque  
}
```



Estructuras de control repetitivas

El bucle **while**

- El comportamiento del bucle **while** es el siguiente:
 1. Se evalúa la **condición** (condición de terminación) asociada al bucle.
 2. Si la evaluación de la condición es `false`, termina la ejecución del bucle.
 3. Si, por el contrario, la evaluación de la condición es `true`, se ejecuta el bloque de instrucciones
 4. Tras ejecutar el bloque de instrucciones, se vuelve al punto 1.
- Un bucle **while** puede realizar cualquier número de iteraciones, desde cero, cuando la primera evaluación de la condición resulta `false`. En el caso de que la condición sea `true` y las variables que la controlan nunca tomen valores que la hagan `false`, tendremos lo que se conoce como **bucle infinito**.

Estructuras de control repetitivas

Ejemplo. Ejemplo de un fragmento de código donde se utiliza una estructura repetitiva **while**

```
.....  
int i = 1;  
while (i <= 5) {  
    System.out.println("Iteracion numero " + i);  
    i++;  
}  
.....
```

Estructuras de control repetitivas

Seguimiento Manual

```
int i = 1;
while (i <= 5) {
    System.out.println("Iteración número "+i);
    i++;
}
```

Iteración	i	Salida a pantalla
0	1	-----
1	2	Iteración número...1
2	3	Iteración número...2
⋮	⋮	⋮
5	6	Iteración número...5

Estructuras de control repetitivas

Ejemplo.

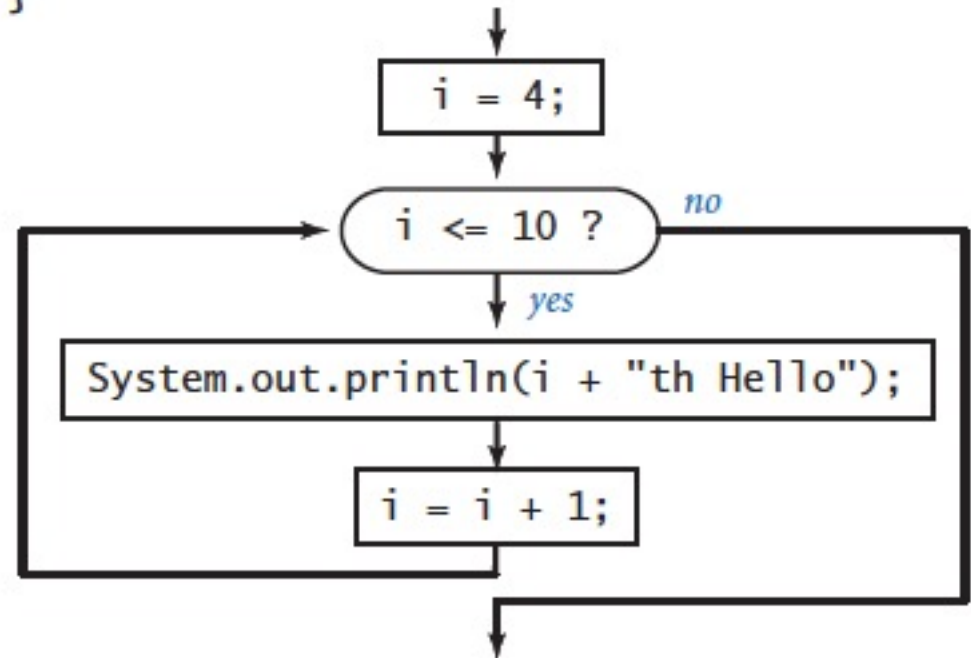
Estructura repetitiva **while**

-Este ciclo se repite 7 veces:
para $i = 4, 5, 6, 7, 8, 9$ y 10

-En la última iteración ($i=10$) se incrementa el valor de i y pasa a valer $i=11$.

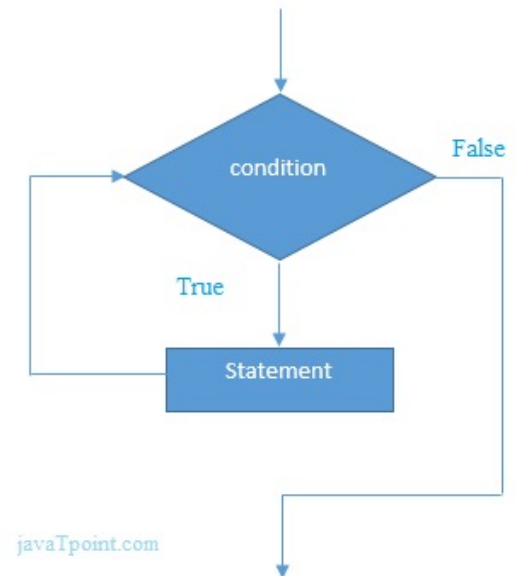
-> La expresión se evalúa a false

```
int i = 4;  
while (i <= 10)  
{  
    System.out.println(i + "th Hello");  
    i = i + 1;  
}
```



Estructuras de control repetitivas

- La ventaja que tiene esta estructura (**while**) es que, comprobada la condición por primera vez, si es `false`, **nunca se llega a ejecutar** el bucle.
- También cabe la posibilidad de que, si la condición no está bien construida, nunca se haga `false` con lo cual tendríamos los *temidos bucles infinitos*.



Estructuras de control repetitivas

Ejemplo. Bucle que no se ejecuta nunca

.....

```
int i = 7;
```

```
while (i <= 5) {
```

```
    System.out.println("Iteracion numero " + i);
```

```
    i++;
```

```
}
```

.....

Estructuras de control repetitivas

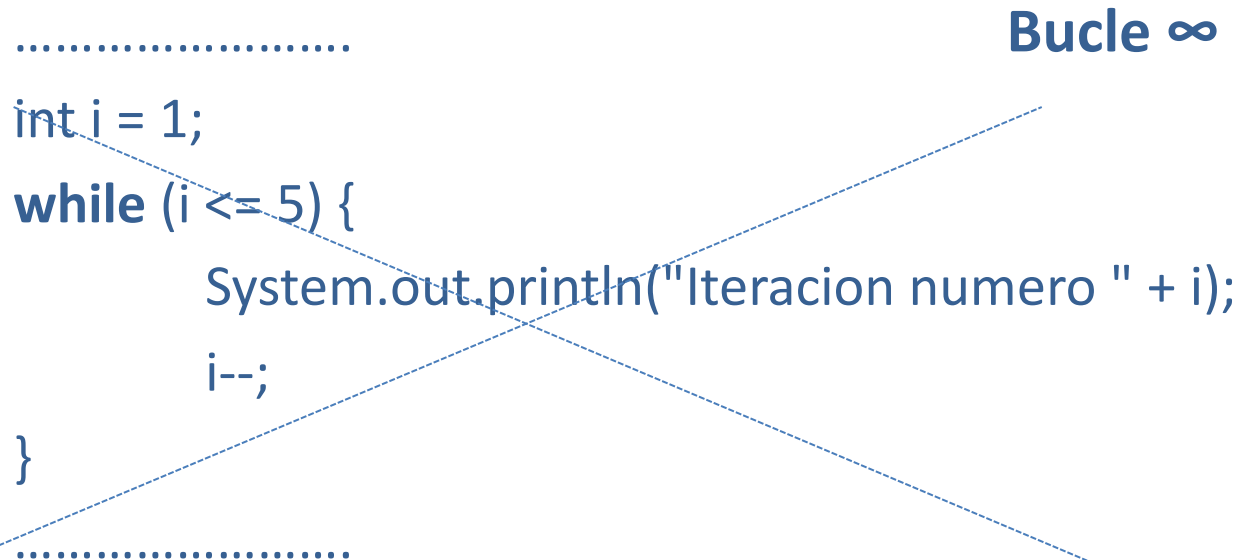
Ejemplo. Bucle infinito

.....

```
int i = 1;  
while (i <= 5) {  
    System.out.println("Iteracion numero " + i);  
    i--;  
}
```

.....

Bucle ∞



Estructuras de control repetitivas

En el bucle son característicos dos elementos: **contadores** y **sumadores**.
A continuación, se describen con un ejemplo.

```
.....  
int i = 1;  
suma = 0;  
dato = 2;  
while (i <= 5) {  
    suma = suma + dato;           // suma += dato;  
    dato = dato + 2;             // dato += 2;  
    i = i + 1;                   // i++;  
}
```

Contador: variable *i* que se incrementa/decrementa en una cantidad fija en cada iteración o paso.

Sumador: variable *suma* que se incrementa/decrementa en una cantidad variable en cada iteración o paso.

Estructuras de control repetitivas

Salidas anticipadas de bucles - Terminar bucle con anterioridad

- **break** finaliza completamente el bucle, sale del bucle
- **continue** detiene la iteración actual, y continua con la siguiente
- Cualquier programa puede escribirse sin **break** y **continue**, incluso hay autores que recomiendan no utilizarlos porque rompen la secuencia natural de ejecución de las instrucciones

```
i = 1;
while (i <= 10) {
    System.out.println("La i vale " + i);
    if (i == 2) {
        break;
    }
    i++;
}
```

```
i = 0;
while (i < 10) {
    i++;
    if (i % 2 == 0) {
        continue;
    }
    System.out.println("La i vale " + i);
}
```

Estructuras de control repetitivas

El bucle **for**

- Ayuda a simplificar la escritura de un bucle especialmente cuando conocemos de antemano el **número de iteraciones** del bucle.

- **Sintaxis**

```
for (expr1; expr2; expr3) {  
    bloque;    // Cuerpo del bucle  
}
```

- Donde

- **expr1**: Inicialización
- **expr2**: Condición de terminación (expresión booleana)
- **expr3**: Incremento

Estructuras de control repetitivas

El bucle **for**

- El funcionamiento del bucle **for** es el siguiente:
 1. Se ejecuta la **inicialización** (de uno o más variables) y se hace una sola vez, al principio.
 2. Después se evalúa la **condición de terminación**, si es `false`, salimos del bucle y continuamos con el resto del programa; en caso de que la evaluación sea `true`, se ejecuta todo el bloque de instrucciones.
 3. Cuando termina de ejecutarse el bloque de instrucciones se ejecuta el **incremento**.
 4. Se vuelve de nuevo a evaluar la ***condición de terminación*** del bucle (volver al punto 2).

Estructuras de control repetitivas

Ejemplo. Ejemplo de un fragmento de código donde se utiliza una estructura repetitiva o bucle **for**

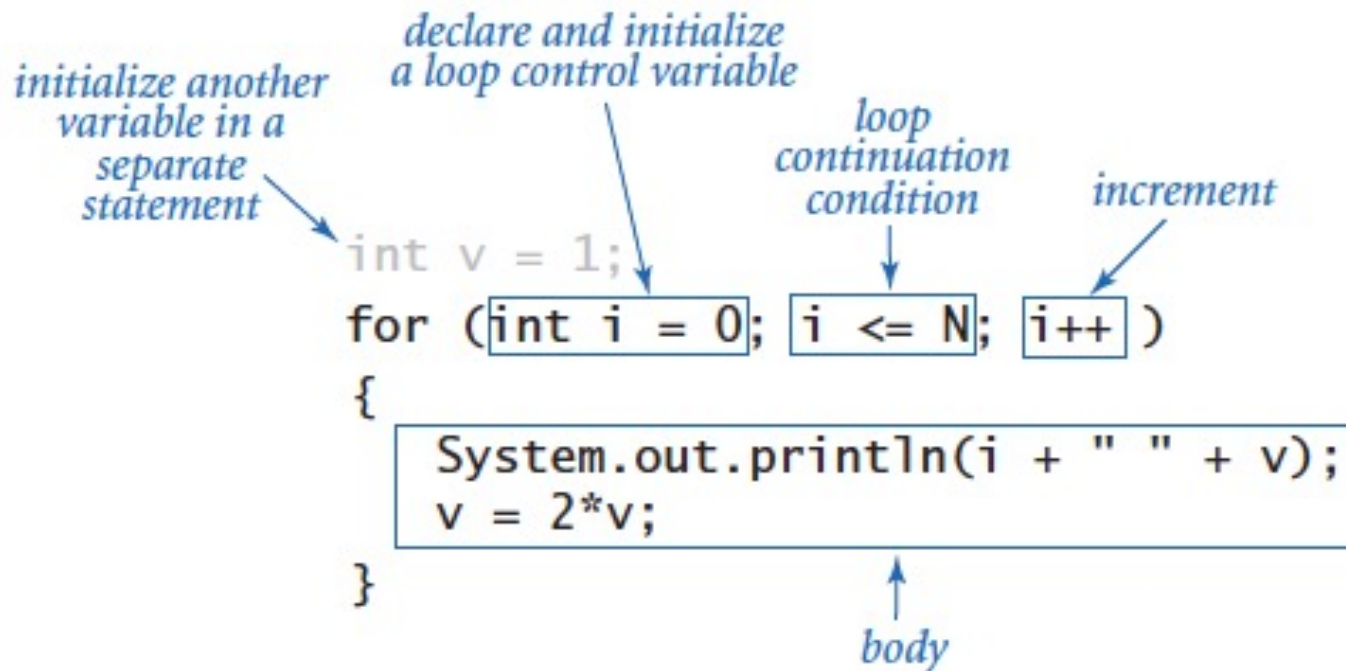
```
.....  
for (int i = 1; i <= 5; i++) {  
    System.out.println("Iteracion numero " + i);  
}  
.....
```

Salida

```
Iteracion numero 1  
Iteracion numero 2  
Iteracion numero 3  
Iteracion numero 4  
Iteracion numero 5
```

Estructuras de control repetitivas

Ejemplo. Ejemplo de un fragmento de código donde se utiliza una estructura repetitiva o bucle **for**



Anatomy of a for loop (that prints powers of 2)

Estructuras de control repetitivas

- Esta estructura está especialmente indicada para **recorridos de arrays** y **matrices** que veremos más adelante.
- Respecto al **funcionamiento** de la estructura **for**, está gobernada por una variable de control que se inicializa a un valor inicial (**expr1**) y dicha variable se va incrementando o decrementando en cada paso o iteración en una cantidad fija que se indica en **expr3** hasta que la variable alcanza el valor final que se expresa en **expr2** .
- Existe también la posibilidad de que las sentencias que componen el cuerpo del bucle no se ejecuten ninguna vez. Por ejemplo:

```
for (int i = 7; i <= 5; i++) {  
    System.out.println("Iteracion numero " + i);  
}
```

Estructuras de control repetitivas

Bucles con **decremento** de la variable de control en cada iteración

```
int n = 5;  
for (int i = n; i >= 1; i--) {  
    System.out.println("Iteracion numero " + i);  
}
```

Salida

```
Iteracion numero 5  
Iteracion numero 4  
Iteracion numero 3  
Iteracion numero 2  
Iteracion numero 1
```

Estructuras de control repetitivas

Equivalencia entre bucles **while** y **for**

Un fragmento de código como el que sigue con un bucle **while**:

```
int i = 0;
int n = 10;
while (i <= 10) {
    System.out.println (n + " x " + i + " = " + (n * i));
    i++;
}
```

puede **abreviarse** si utilizamos un bucle **for**:

```
int n = 10;
for (int i = 0; i <= 10; i++) {
    System.out.println (n + " x " + i + " = " + (n * i));
}
```

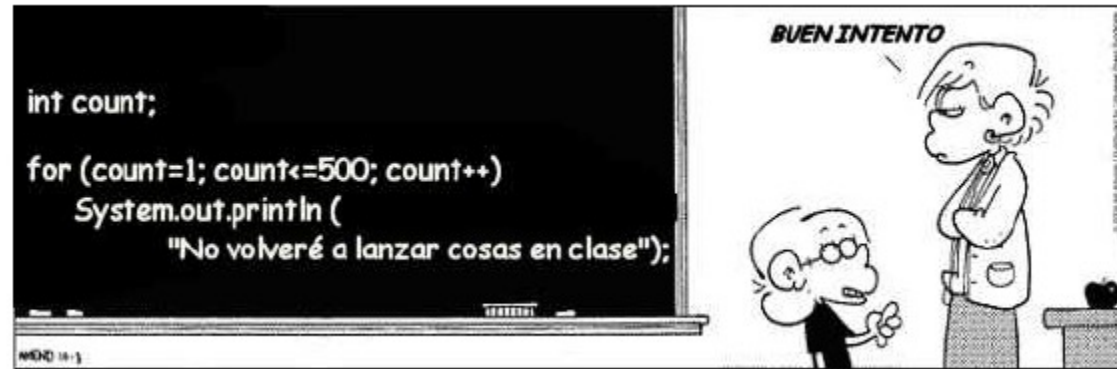

Estructuras de control repetitivas

En general,

```
for (expr1; expr2; expr3) {  
    bloque;  
}
```

equivale a

```
expr1;  
while (expr2) {  
    bloque;  
    expr3;  
}
```



Estructuras de control repetitivas

Equivalencia entre bucles **while** y **for**

Un fragmento de código como el que sigue con un bucle **while**:

```
while (dato != 0) {  
    // Cuerpo del bucle  
}
```

```
while (condición de continuidad) {  
    // Cuerpo del bucle  
}
```

puede expresarse si utilizamos un bucle **for**:

```
for (; dato != 0; ) {  
    // Cuerpo del bucle  
}
```

```
for (; condición de continuidad; ) {  
    // Cuerpo del bucle  
}
```

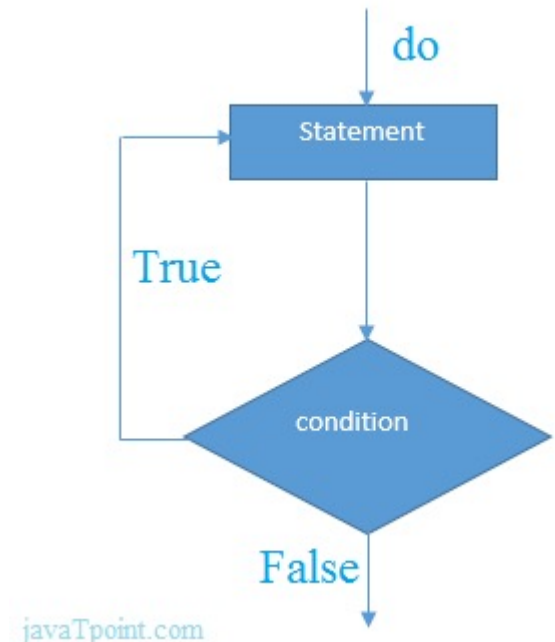
Tema 1. Fundamentos de Programación

El bucle **do-while**

Es un tipo de bucle similar al **while**, que realiza la comprobación de la condición **después** de ejecutar el cuerpo del bucle.

Sintaxis

```
do  
    sentencia;  
while (condición);  
  
do {  
    bloque  
} while (condición);
```



Estructuras de control repetitivas

- El funcionamiento del bucle **do-while** es el siguiente:
 1. Se ejecuta el bloque de instrucciones.
 2. Se evalúa la condición.
 3. Según el valor obtenido en la **condición** se termina el bucle si es `false` o se comienza en el punto 1 si es `true`.
- Debemos recordar que **do-while** es la única estructura de bucle que finaliza en punto y coma (;). Mientras que el bucle **while** se puede ejecutar de 0 a infinitas veces, el bucle **do-while** lo hace de 1 a infinitas veces. De hecho, la única diferencia con el bucle **while** es que el cuerpo del bucle **do-while** se ejecuta, al menos una vez.

Estructuras de control repetitivas

- El bloque de instrucciones se ejecuta **al menos una vez**.
- El bucle **do-while** resulta especialmente indicado para **validar datos** de entrada.
- Por ejemplo, **comprobar** que los valores de entrada obtenidos están dentro del rango de valores que el programa espera:

```
do {  
    System.out.println("Introduce un valor numérico para el mes");  
    mes = entrada.nextInt();  
} while (mes < 1 || mes > 12);
```

Salida

```
1 package org.ip.tema01;
2
3 import java.util.Scanner;
4
5 public class PruebaDoWhile {
6     @SuppressWarnings("resource")
7
8     public static void main(String[] args) {
9         int data;
10        int sum = 0;
11
12        // Crear un Scanner para leer datos de entrada
13        Scanner input = new Scanner(System.in);
14
15        // Continuar leyendo hasta que la entrada sea 0
16        do {
17            // Leer los datos siguientes
18            System.out.print( "Introduce un entero (la entrada termina con 0): ");
19            data = input.nextInt();
20
21            sum += data;
22        } while (data != 0);
23
24        System.out.println("La suma es " + sum);
25    }
26 }
```

Introduce un entero (la entrada termina con 0): 1
Introduce un entero (la entrada termina con 0): 3
Introduce un entero (la entrada termina con 0): 5
Introduce un entero (la entrada termina con 0): 0
La suma es 9

¿Qué bucle utilizar?

- Los bucles **while** y **for** se llaman bucles *pre-test* porque la condición de continuación se comprueba antes de ejecutar el cuerpo del bucle.
- El bucle **do-while** se llama *post-test* porque la condición se comprueba después de que el cuerpo del bucle se ha ejecutado.
- Las tres sentencias de bucles: **for**, **while** y **do-while** son EQUIVALENTES es decir podemos escribir un bucle con cualquiera de ellas, con la que estemos más cómodos, pero lo habitual es:
 - Si se conocen de antemano el número de iteraciones => **for**
 - Si no se conoce dicho número de iteraciones => **while**
 - Para validar datos => **do-while**

Estructuras de control repetitivas

Bucles anidados

Se trata de uno o varios bucles **contenidos** en otros, es decir, tendremos un bucle cuyo cuerpo contenga al menos una sentencia que sea a su vez otro bucle y éste a su vez puede tener otra sentencia que sea un bucle.

Por ejemplo:

```
int N = 3;
for (int i = 0; i < N; i++) {           // Bucle externo
    for (int j = 0; j < N; j++) {       // Bucle interno
        System.out.print("(" + i + "," + j + ") " + '\t');
    }
    System.out.println();
}
```

Salida	(0,0)	(0,1)	(0,2)
	(1,0)	(1,1)	(1,2)
	(2,0)	(2,1)	(2,2)

Estructuras de control repetitivas

Bucles anidados

En el uso de bucles es muy frecuente la anidación. Al hacer esto se **multiplica** el número de veces que se ejecuta el bloque de instrucciones de los bucles internos.

Los bucles anidados pueden encontrarse relacionados (**dependientes**) cuando los valores de las variables de los bucles más externos intervienen en el control de la iteración de un bucle interno de un bucle interno, o **independientes**, cuando no existe relación alguna entre ellos, siendo el número de iteraciones de un bucle ajeno a los valores de las variables utilizadas en otro bucle, ya sea externo o interno.

Anidar bucles es una herramienta que facilita el procesamiento de **tablas multidimensionales**, utilizándose cada nivel de anidación para manejar los índices de la dimensión correspondiente.

Estructuras de control repetitivas

Bucles anidados independientes

Cuando los bucles anidados **no dependen** unos de los otros para determinar el número de iteraciones se denominan, **bucles anidados independientes**.

```
for (int i = 1; (i <= 4); i++) {  
    System.out.println("+ " + i);  
    for (int j = 1; (j <= 3); j++) {  
        System.out.println(".. * " + j);  
    }  
}  
  
for (int i = 1; (i <= 2); i++) {  
    System.out.println("+ i = " + i);  
    for (int j = 1; (j <= 3); j++) {  
        System.out.println(".. * j = " + j);  
        for (int k = 1; (k <= 4); k++) {  
            System.out.println(".... - k = " + k);  
        }  
    }  
}
```

Estructuras de control repetitivas

Bucles anidados dependientes

Puede darse el caso que el número de iteraciones de un bucle **no sea independiente** de la ejecución de los bucles externos, y dependa de sus variables de control, con lo que nos encontramos con los denominados **bucles anidado dependientes**.

En este ejemplo la variable *i* se toma como base para comparar con los valores de la variable *j* que controla el bucle más interno.

```
for (int i = 1; (i <= 4); i++) {  
    System.out.println("+ i = " + i);  
    int j = 1;  
    while (j <= i) {  
        System.out.println(".. * j = " + j);  
        j++;  
    }  
}
```

Estructuras de control repetitivas

Diseño de un bucle

A partir del enunciado de un problema, ¿cómo puedo construir correctamente un bucle?. En primer lugar, optaremos por una estructura repetitiva (bucle) siempre que haya proceso que se **repite**. A continuación, puede ser de ayuda responder a las preguntas:

- ¿Cuál es la condición de terminación del bucle?
- ¿Cómo se inicializa y actualiza la condición?
- ¿Cuál es el proceso que se repite?
- ¿Cómo se inicializa y actualiza el proceso?

¡MUCHAS GRACIAS!



UNIVERSIDAD DE ALMERÍA

