

UT 2: Estructuras de control.

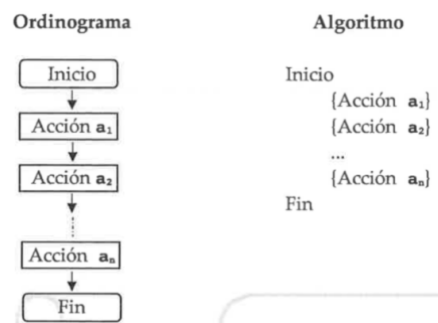
1.	<i>Estructuras de control</i>	2
1.1.	Estructura secuencial	2
1.2.	Estructura alternativa o condicional	2
1.3.	Estructura repetitiva o iterativa.....	3
2.	<i>Estructuras de control en Java</i>	4
2.1.	If-Else	4
2.2.	Else-if	4
2.3.	If-Else anidados.....	5
2.4.	Switch	5
2.5.	While.....	5
2.6.	Do-while.....	6
2.7.	For.....	6
3.	<i>Sentencias break y continue</i>	6
4.	<i>Depuración</i>	6
5.	<i>JavaDoc</i>	8

1. Estructuras de control

Según el Teorema de la estructura, cualquier programa con una única entrada y salida se puede ejecutar con tres tipos de estructuras de control: secuencial, alternativa y repetitiva.

1.1. Estructura secuencial

Una estructura secuencial es aquella en la que las acciones se ejecutan sucesivamente, una detrás de otra, desde la entrada hasta la salida, sin bifurcaciones y sin poder omitir ninguna.

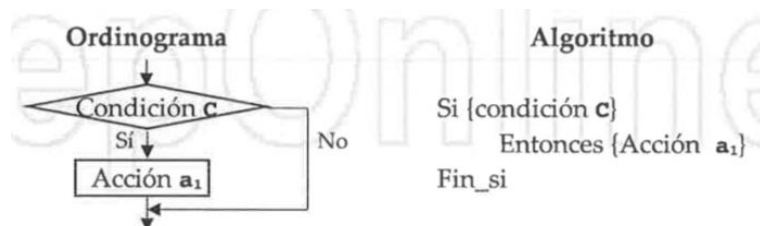


1.2. Estructura alternativa o condicional

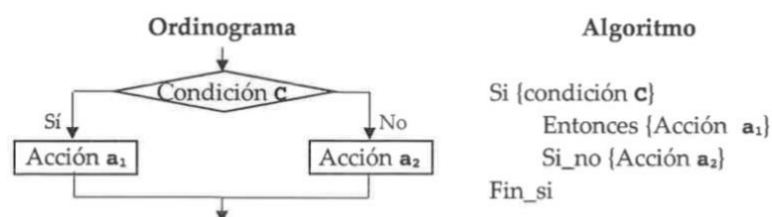
Es aquella estructura en la que se ejecuta una alternativa dependiendo del valor de una condición.

También se las conoce como condicionales y pueden ser de tres tipos:

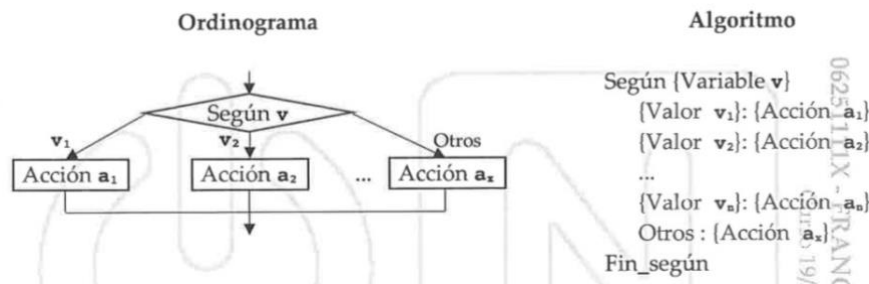
- **Estructura alternativa simple:** Estructura en la que el cumplimiento de una condición implica la ruptura de la secuencia y la ejecución de una determinada acción. En muchos lenguajes de programación se conoce como sentencia "if".



- **Estructura alternativa doble:** Permite la elección entre dos acciones en función de que se cumpla o no una condición. Se conoce como sentencia "if-else".



- **Estructura alternativa múltiple:** Esta estructura se da cuando la condición permite múltiples alternativas, por tanto, múltiples acciones. Se suele conocer como sentencia “switch”.



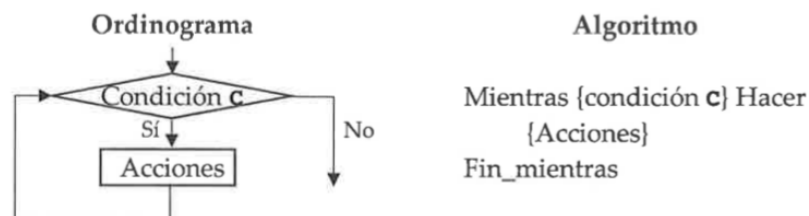
1.3. Estructura repetitiva o iterativa

También conocidas como iterativas, son aquellas que ejecutan un grupo de acciones un número determinado de veces, en función del cumplimiento de una condición. Este tipo de estructuras se utiliza mucho para sub-acciones más simples que se repiten varias veces en una acción.

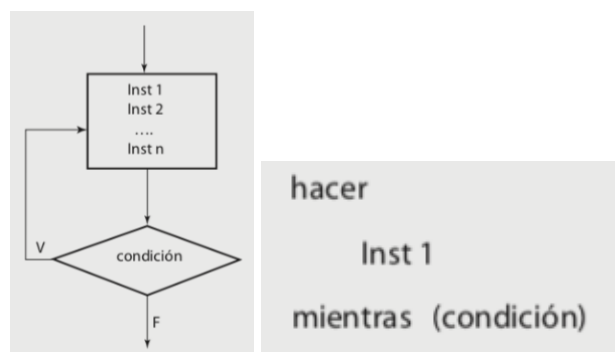
El número de veces que se repiten el grupo de acciones se denomina bucle.

La condición puede encontrarse al principio o al final de la iteración y en función de esto hay tres tipos de estructuras:

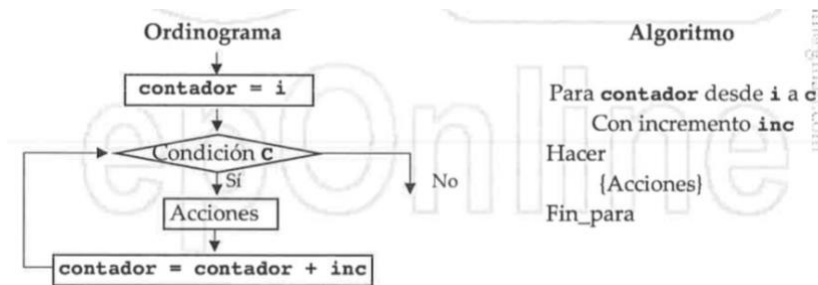
- **Estructura mientras:** El grupo de instrucciones se repite mientras la condición se cumpla. Se suele conocer como sentencia o bucle “while”.



- **Estructura repetir-mientras:** El grupo de instrucciones se repite mientras que la condición sea verdadera, es decir, se evalúa la condición al final del grupo de acciones y si NO se cumple, se sale de la estructura. Se conoce como sentencia o bucle “do-while”.



- **Estructura para:** La estructura se repite un número fijo de veces, por tanto, se podría decir que ese número se conoce previo a ejecutar la estructura. Se conoce como sentencia o bucle "for".



2. Estructuras de control en Java

2.1. If-Else

```
private static void ejemploIfElse() {
    //Utilizado en condiciones binarias (o verdadero o falso)

    boolean condicion;

    condicion = true;

    if(condicion) {
        System.out.println("La condición es verdadera");
    } else {
        System.out.println("La condición es falsa");
    }
}
```

2.2. Else-if

```
private static void ejemploElseIf() {
    //Utilizado en condiciones binarias (o verdadero o falso)

    int condicion;

    condicion = 30;

    if(condicion < 10) {
        System.out.println("La condición es menor de 10");
    } else if (condicion < 20){
        System.out.println("La condición es menor de 20");
    } else if (condicion < 30){
        System.out.println("La condición es menor de 30");
    } else {
        System.out.println("La condición es mayor o igual a 30");
    }
}
```

2.3. If-Else anidados

```
private static void ejemploIfElseAnidados() {  
    int condicion;  
  
    condicion = 5;  
  
    if(condicion < 10) {  
        if(condicion < 0) {  
            System.out.println("La condición es un número negativo");  
        } else {  
            System.out.println("La condición es un número entre 0 y 10");  
        }  
    } else {  
        System.out.println("La condición es mayor o igual a 10");  
    }  
}
```

2.4. Switch

```
private static void ejemploSwitch() {  
    //Para condiciones múltiples.  
  
    String cadena;  
    cadena = "uno";  
  
    //Ojo, esto solo es válido para enteros, cadenas y enumeraciones.  
    //Ojo2, si en los "case" tienes que realizar comparaciones, debe realizarse con if-else anidados.  
    switch(cadena) {  
        case "uno":  
            System.out.println("La condición es uno");  
            break;  
        case "dos":  
            System.out.println("La condición es dos");  
            break;  
        case "tres":  
            System.out.println("La condición es tres");  
            break;  
        default:  
            System.out.println("La condición no es ninguna de las anteriores");  
    }  
}
```

2.5. While

```
private static void ejemploWhile() {  
    int i = 0;  
  
    while(i < 10) {  
        System.out.println(i);  
        i++;  
    }  
}
```

2.6. Do-while

```
private static void ejemploDoWhile() {  
    int i = 0;  
  
    do {  
        System.out.println(i);  
        i++;  
    } while(i <= 0);  
}
```

2.7. For

```
for (inicialización; condición; incremento) {  
    bloque de instrucciones  
    ...  
}
```

```
private static void ejemploFor() {  
    for (int i = 0; i < 10; i++) {  
        System.out.println(i);  
    }  
}
```

3. Sentencias break y continue

La sentencia **break** se utiliza para interrumpir la ejecución de una estructura de repetición o de un *switch*. Cuando se ejecuta el *break*, el flujo del programa continúa en la sentencia inmediatamente posterior a la estructura de repetición o al *switch*.

La sentencia **continue** únicamente puede aparecer en una estructura de repetición. Cuando se ejecuta un *continue*, se deja de ejecutar el resto del bloque de sentencias de la estructura iterativa para volver al inicio de ésta.

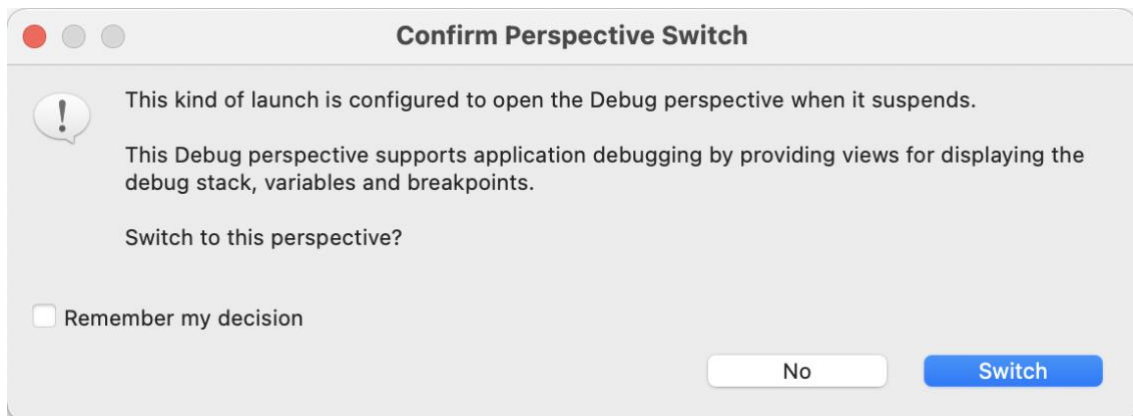
4. Depuración

Básicamente depurar un programa es recorrerlo paso a paso, viendo el valor de las variables del programa. Esto nos permite ver exactamente qué está pasando en nuestro programa cuando se está ejecutando una línea de código específica.

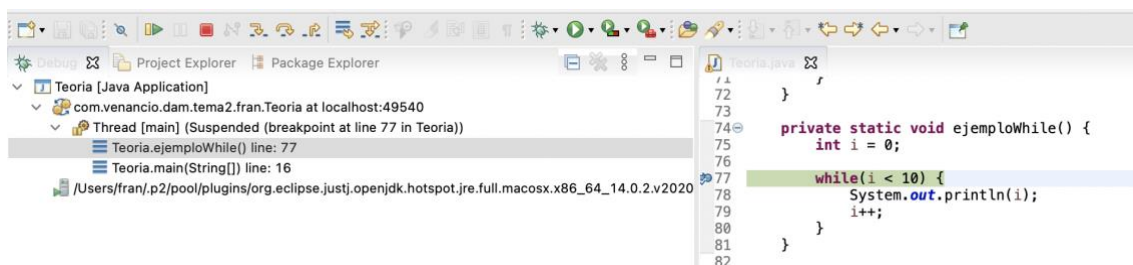
Lo primero es colocar un punto de interrupción (*breakpoint*), en una línea de código de nuestro programa. Doble click en el margen izquierdo de la ventana donde vemos el código para crear el *breakpoint*.

```
73  
74 private static void ejemploWhile() {  
75     int i = 0;  
76  
77     while(i < 10) {  
78         System.out.println(i);  
79         i++;  
80     }  
81 }
```

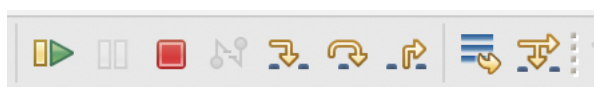
Lo siguiente es ejecutar el programa desde el modo de depuración (icono de la izquierda), y abrirá la siguiente ventana, que permite ir a la perspectiva de depuración:



Una vez aceptado (switch), se abrirá la perspectiva de depuración, e indicará en verde la línea en la que para el depurador:



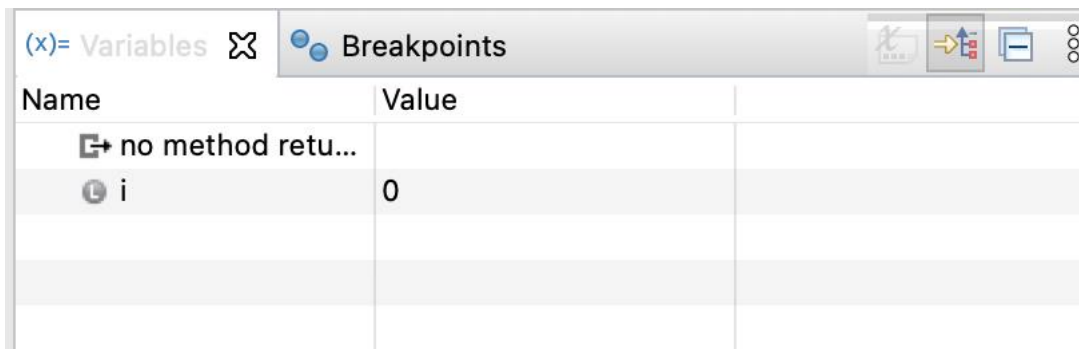
Los botones superiores permiten ejecutar línea a línea la depuración o ir al siguiente *breakpoint*.



Las funciones de los botones son las siguientes:

- **Resume (F8):** continúa con la ejecución (hasta el próximo *breakpoint*).
- **Suspend:** podemos detener la ejecución, aunque no alcancemos un *breakpoint* (muy útil cuando entramos en un ciclo infinito).
- **Stop:** detiene la depuración.
- **Step Into (F5):** la depuración continúa dentro del método en el que para el programa (solo si hay un método en el punto en el que está parado el *debug*).
- **Step Over (F6):** pasa a la siguiente línea que vemos en la vista de código.
- **Step Return (F7):** vuelve a la línea siguiente del método que llamó al método que se está depurando actualmente. O lo que es lo mismo, sube un nivel en la pila de ejecución, que vemos en la vista *Debug*.

Además, Eclipse dispone de una vista de variables, en la que se puede observar el contenido de cada una de ellas y otra de *breakpoints*:



Name	Value
no method retu...	
i	0

5. JavaDoc

Los comentarios *JavaDoc* están destinados a describir, principalmente, clases y métodos. Como están pensados para que otro programador los lea y utilice la clase (o método) correspondiente, se decidió fijar al menos parcialmente un formato común, de forma que los comentarios escritos por un programador resultaran legibles por otro. Para ello los comentarios *JavaDoc* deben incluir unos indicadores especiales, que comienzan siempre por '@' y se suelen colocar al comienzo de línea.

Los indicadores más comunes son:

- **@author nombreDelAutor:** indica quién escribió el código al que se refiere el comentario. Si son varias personas se escriben los nombres separados por comas o se repite el indicador, según se prefiera. Es normal incluir este indicador

en el comentario de la clase y no repetirlo para cada método, a no ser que algún método haya sido escrito por otra persona.

- **@version númeroVersión/fecha:** utilizado normalmente con clases, si se quiere dejar registro de la versión de la propia clase.
- **@param nombreParámetro descripción:** describir un parámetro de un método.
- **@return descripción:** describe el valor de salida de un método. No se aplica con constructores ni métodos que devuelven void.
- **@see descripción:** permite hacer referencia a otro código e incluso referencias web.
- **@throws nombreClaseExcepción descripción.** Cuando un método puede lanzar una excepción se indica así.
- **@deprecated descripción.** Indica que el método ya no se usa y se ha sustituido por otro.

```

1 package com.venancio.dam.tema2.fran.javadoc;
2
3 /**
4  * @author fran
5  *
6  * Clase para el cálculo de áreas y parámetros de un círculo
7  *
8  */
9 public class CirculoJD {
10
11     private double r;
12
13     /**
14      * Constructor de la clase CirculoJD.
15      * @param radio
16      *      Valor del radio del círculo
17      */
18     public CirculoJD(double radio) {
19         this.r = radio;
20     }
21
22     /**
23      * Método que calcula el área del círculo
24      * @return el valor del área.
25      */
26     public double calculoArea() {
27         return Math.PI*Math.pow(this.r, 2);
28     }
29
30     /**
31      * Método que calcula el perímetro del círculo
32      * @return el valor del perímetro.
33      */
34     public double calculoPerimetro() {
35         return 2 * Math.PI * this.r;
36     }
37 }
38

```

Para la generación de comentarios JavaDoc, se pueden utilizar los atajos Shift-Alt-J o ⌘-⌘-J en Mac. Simplemente colocas el cursor sobre el método o clase oportunos y pulsas las combinaciones de teclas.