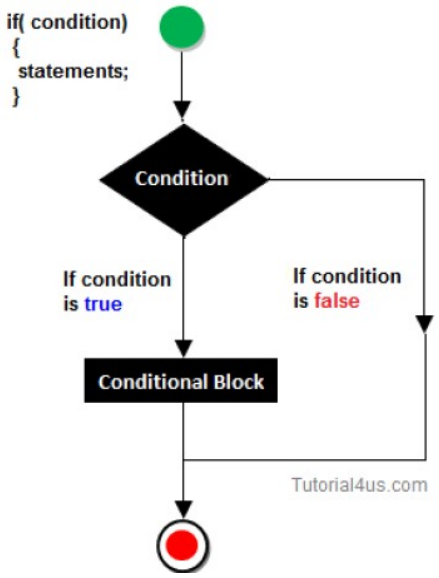
UT2 Las estructuras de control

RESULTADO DE APRENDIZAJE
3. Escribir y depurar código, analizando y utilizando las estructuras de control del lenguaje.

Tabla de contenido

1. Estructuras de selección.

Es una bifurcación o sentencia condicional de una o dos ramas. La sentencia de control evalúa la condición lógica o booleana. Si esta condición es cierta entonces se ejecuta la sentencia o sentencias (1) que se encuentra a continuación. En caso contrario, se ejecuta la sentencia (2) que sigue a else (si ésta existe). La sentencia puede constar opcionalmente de una o dos ramas con sus correspondientes sentencias.



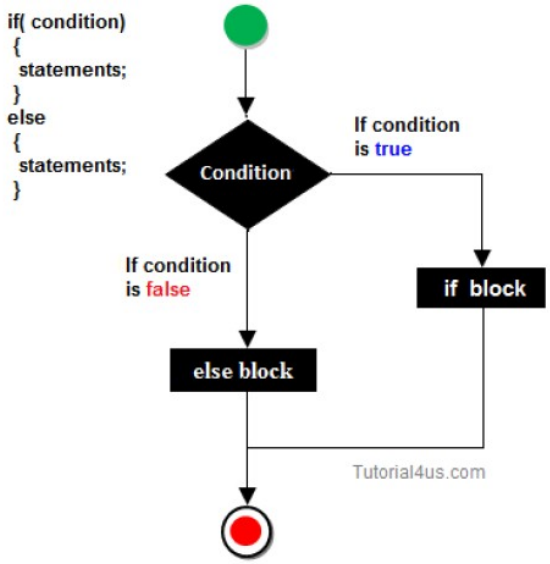
Sintaxis:

```
if (expresionLogica) {
    sentencia_1;
}
```

O bien

```
if (expresionLogica) {
    sentencia_1;
} else {
    sentencia_2;
}
```

a expresionLogica debe ir entre paréntesis. Las llaves sólo son obligatorias si las sentencias (1) ó (2) son compuestas (las llaves sirven para agrupar varias sentencias simples).



La parte else y la sentencia posterior entre llaves no son obligatorias. En este caso quedaría una sentencia selectiva con una rama.

```
if(numero>0) {
    System.out.println("El número es positivo");
}else {
    System.out.println("El número es negativo");
}
```

Sentencia switch: Se evalúa la expresión y dependiendo del valor obtenido se ejecutará alguno de los casos, ninguno o uno por defecto.

Switch (expresion){

Case value1: sentencias1; break;

Case value2: sentencias2; break;

Case value3: sentencias3; break;

Case value4: sentencias4; break;

[default sentencias5]

}

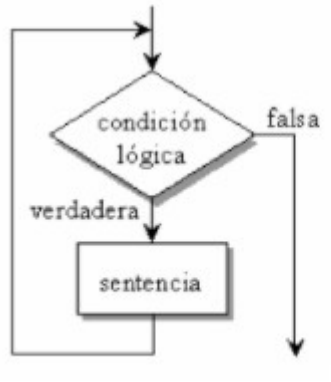
2. Estructuras de repetición.

SENTENCIA WHILE

Es un bucle o sentencia repetitiva con una condición al principio. Se ejecuta una sentencia mientras sea cierta una condición. La sentencia puede que no se ejecute ni una sola vez.

Sintaxis:

```
[inicializacion;]
while (expresionLogica) {
    sentencias;
    [iteracion;]
}
```

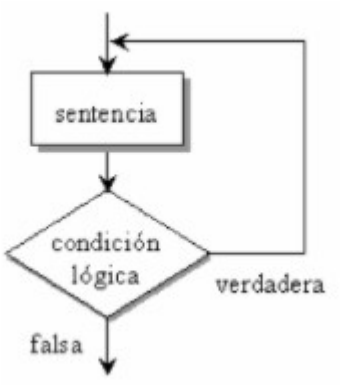


Por ejemplo, un contador de números

```
while(num>=0) {
        contador++;
        System.out.println("Introduce un numero: ");
        num=leer.nextInt();
}
```

SENTENCIA DO-WHILE

Es un bucle o sentencia repetitiva con una condición al final. Se ejecuta una sentencia mientras sea cierta una condición. En este caso, la sentencia se ejecuta al menos una vez.



Sintaxis:

Sintaxis:

```
do {
    sentencias;
    [iteracion;]
} while (expresionLogica);
```

Por ejemplo:

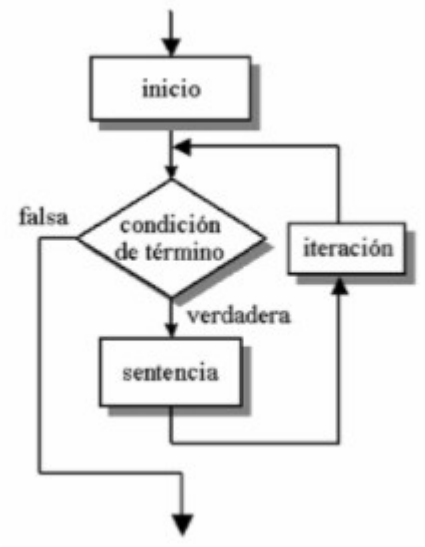
```
do{
        System.out.println("Dame un número");
        numero=s.nextInt();
        if(numero>0) {
            System.out.println("El número es positivo");
        }else {
            System.out.println("El número es negativo");
        }

    }while(numero!=0);
```

SENTENCIA FOR

Es un bucle o sentencia repetitiva que ejecuta la sentencia de inicio. Verifica la expresión booleana de término.

Sintaxis:



si es cierta, ejecuta la sentencia entre llaves y la sentencia de iteración para volver a verificar la expresión booleana de término. Si es falsa, sale del bucle.

```
for (inicio; termino; iteracion) {
    sentencia_1;
    sentencia_2;
    sentencia_n;
}
```

Por ejemplo, un acumulador de 5 números.

```
for (int i = 0; i < 5; i++) {
    System.out.println("Dame un número");
    num = sc.nextInt();
    suma += num;
}
```

3. Estructuras de salto.

SENTENCIA BREAK

La sentencia break puede encontrarse en sentencias **switch** o en bucles. Al ejecutarse, sale del bucle y/o pasa a la siguiente sentencia. Puede emplearse con etiquetas, especificando sobre qué sentencia se aplica si hay varias anidadas.

SENTENCIA CONTINUE

La sentencia continue se emplea sólo en bucles. Al ejecutarse la iteración en la que se encuentra, el bucle finaliza y se inicia la siguiente. También puede emplearse con etiquetas, especificando sobre que sentencia se aplica si hay varias anidadas.

SENTENCIA RETURN

La sentencia es otra sentencia para salir de una estructura de control. La diferencia con break y continue es que return sale del método que se está ejecutando y permite devolver un valor.

4. Control de excepciones.

Una excepción es una situación anómala a la que llega la ejecución de un programa. Estas situaciones anómalas pueden ser el intento de abrir un fichero que no existe, la división por cero o el acceso a un objeto no inicializado.

Java proporciona un mecanismo para detectar y solucionar las excepciones que se puede llegar a producir durante la ejecución de un programa. En Java estamos obligados a tratar las excepciones cuando se producen, bien gestionándolas directamente o bien desentendiéndonos de ellas, pero hasta esto último debemos hacerlo explícitamente.

En Java existen dos grandes tipos de excepciones: los Errores y las Excepciones propiamente dichas. Un caso particular son las excepciones que derivan de RuntimeException, como por ejemplo NullPointerException.

Para gestionar una excepción debe emplearse una sentencia try. La sintaxis de la sentencia se muestra a continuación:

```
try {

    System.out.println("Introduce una cadena");
    texto=br.readLine();
    System.out.println("La cadena escrita es: "+texto);
    in.close();

}catch(Exception e) {
    e.getStackTrace();
}
```

5. Prueba y depuración de programas.

Se evalúan los resultados de la ejecución y se comprueba que hay una falta de correspondencia entre:

- Resultados esperados
- Resultados obtenidos

El proceso de depuración siempre tiene uno de los resultados siguientes:

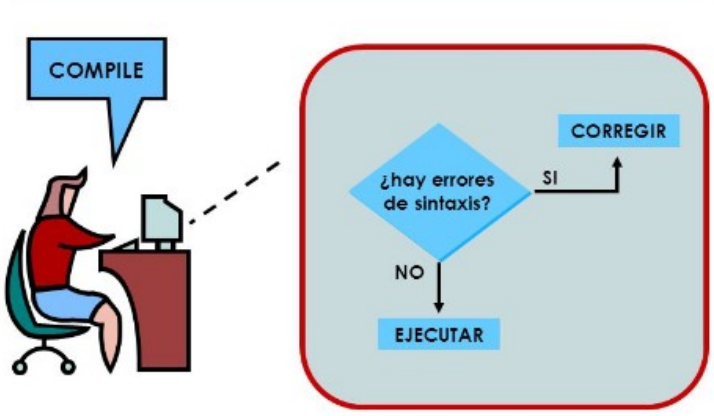
a) Se encuentra el error, se corrige y se elimina

b) No se encuentra el error

- Sospechar la causa
- Diseñar casos de prueba
- Repetir las pruebas

Entre los errores más comunes podemos encontrar:

- De compilación: Al codificar y escribir las sentencias
- Errores lógicos: Devuelven resultados inesperados y erróneos.

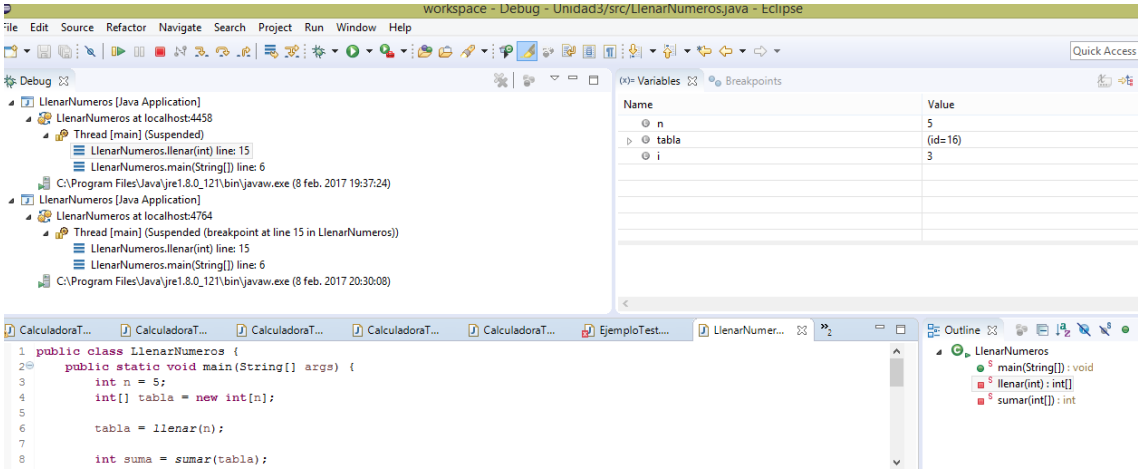


La herramienta de depuración que incorpora Eclipse se llama Debugger y nos permite analizar el código del programa mientras se ejecuta. Permite establecer puntos de ruptura o interrupción, suspender la ejecución del programa, ejecutar el código paso a paso, examinar el contenido de las variables.

Se puede lanzar el depurador de varias formas:

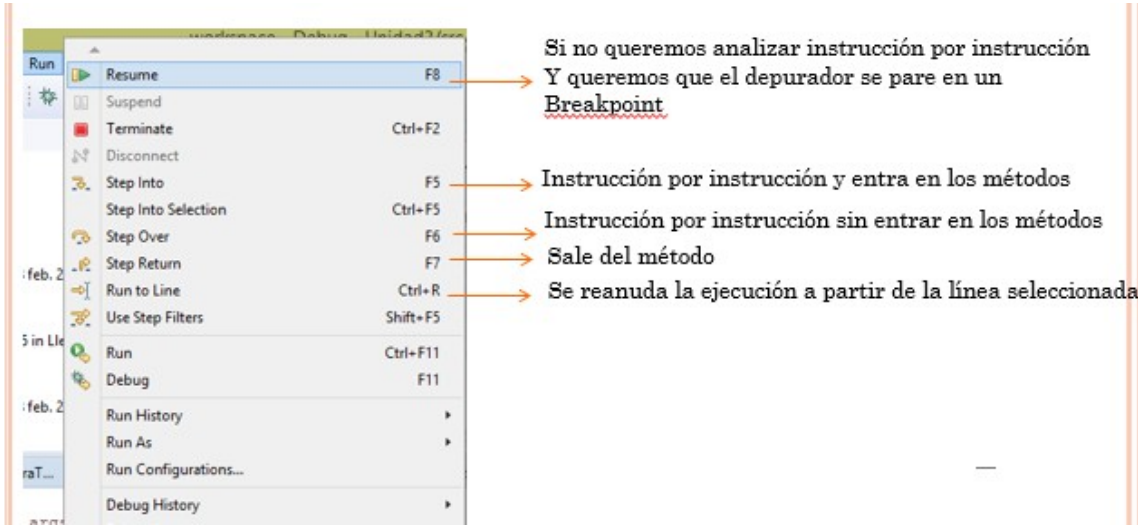
- Botón debug
- Menú Run→Debug
- Botón derecho del ratón en la clase que se va a ejecutar y Debug As→ Java Application

También hay que abrir la vista de depuración desde el menú Window→Open perspective→Debug



En esta vista se pueden ver varias zonas:

- Vista EDITOR: Se va marcando la traza de ejecución del programa mostrándose una flecha azul en el margen izquierdo de la línea que se está ejecutando
- Vista DEBUG: Muestra los hilos de ejecución
- Vistas de INSPECCIÓN: Permiten ver los valores de las variables y de los puntos de ruptura que intervienen en el programa
- Vista CONSOLA: Muestra la consola de ejecución del programa que se está depurando.



Una vez establecido ejecutamos el programa en modo depuración. El programa se ejecutará de forma normal hasta que llegue a ese punto donde se detendrá. En la ventana Debug aparece la pila de llamada donde se ven cada uno de los hilos de ejecución

6. Documentación del código del programa.

Documentar un proyecto es algo fundamental de cara a su futuro mantenimiento. Cuando programamos una clase, debemos generar documentación lo suficientemente detallada sobre ella como para que otros programadores sean capaces de usarla sólo con su interfaz. **Javadoc** es una utilidad para la generación de documentación en formato HTML a partir de código fuente Java.

7. Bibliografía

- OpenWebinars
- Manualweb.net
- https://www.arkaitzgarro.com/
- Aprenderaprogramar.com