Programación. 1º DAW

Unidad 3: Uso de estructuras de control

Uso de estructuras de control

- Estructuras de programación
  - Funciones
  - Estructuras alternativas o condicionales: if, if-else, switch
  - Estructuras repetitivas o ciclos: while, do-while, for.
  - Control de flujo
  - Excepciones en Java.

Funciones

- Una función en Java dentro de una clase, puede ser llamada por cualquier método o función de dicha clase, siempre y cuando este definida antes de ser llamada (tal y como ocurre en C).

- El código de una función es el siguiente:

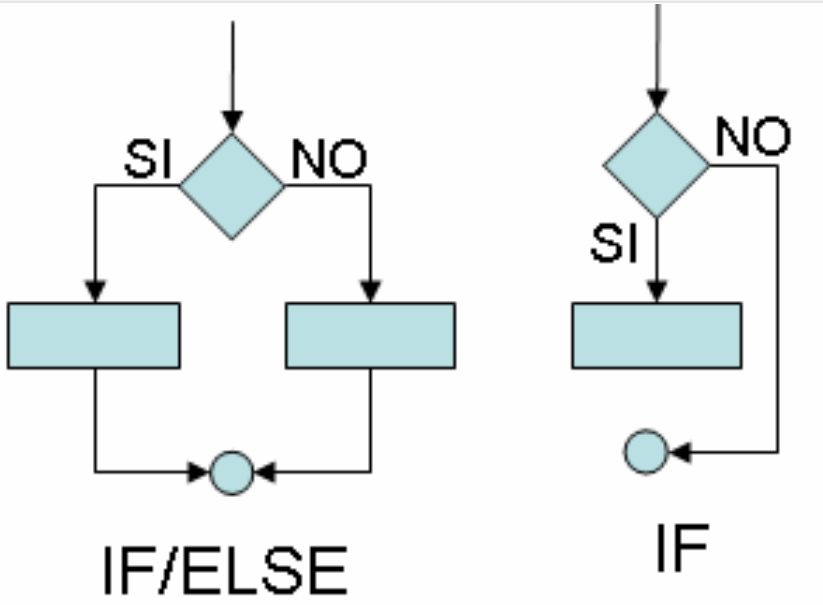
```
static tipo NombreFuncion(tipodato dato){
    Sentencias;
}
```

- ***Ejercicio:*** Realizar una función que reciba tres palabras y las muestre concatenadas separadas por un espacio en blanco

Estructuras alternativas o condicionales

- **if/else**

```
if( Boolean ) {
    sentencias;
} else {
    sentencias;
}
```

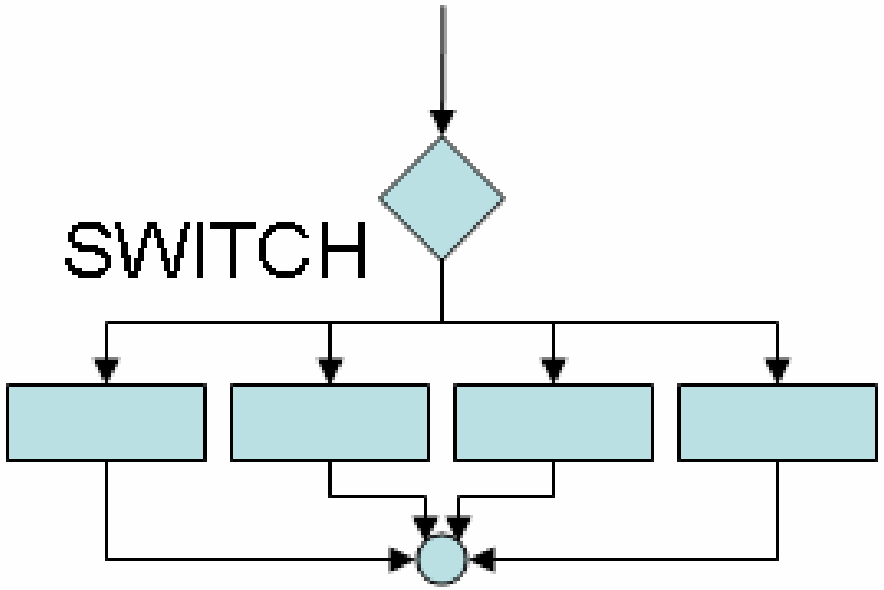


- ***Ejercicio:*** **Crear un programa que nos diga si el único argumento que se le pasa es par o impar.**

Estructuras alternativas o condicionales

- **switch**

```
switch( expr1 ) {
        case expr2:
                sentencias;
                break;
        case expr3:
                sentencias;
                break;
        default:
                sentencias;
                break;
}
```



- ***Recuerda:*** Si no se escribe la sentencia **break**, el programa seguirá ejecutando las siguientes sentencias hasta encontrarse con un break o el fin del swicth.

- ***Ejercicio:*** **Crear un programa utilizando switch que nos diga si el único argumento que se le pasa es par o impar.**

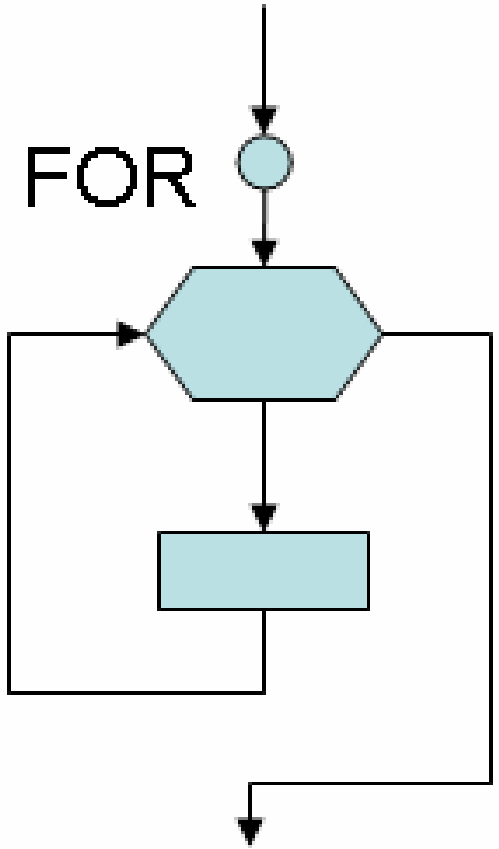
Estructuras repetitivas o ciclos

- **Bucles for**

```
for ( expr1 inicio; expr2 test; expr3 incremento ) {
    sentencias;
}
```

  - También se soporta el operador *coma* (,) en los bucles for

```
for ( a=0,b=0; a < 7; a++,b+=2 )
```



- ***Ejercicio:*** **Crear un programa que sume los 100 primeros números.**

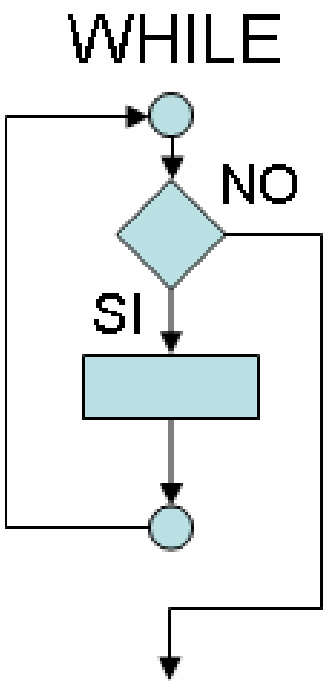
Estructuras repetitivas o ciclos

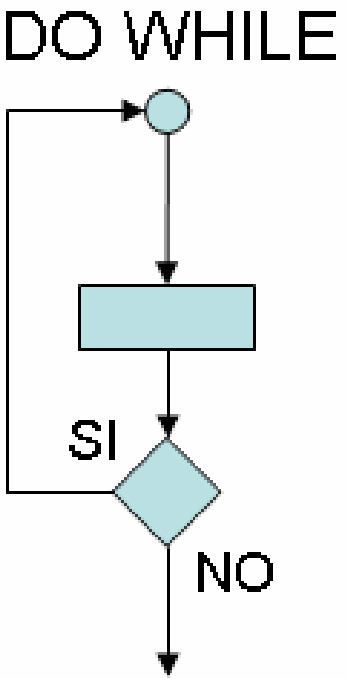
- **Bucles while**

```
while( Boolean ) {
    sentencias;
}
```

- **Bucles do/while**

```
do {
    sentencias;
} while( Boolean );
```





- ***Ejercicio:*** **Crear dos programas que sume los 100 primeros números utilizando las dos estructuras anteriores.**

Control de Flujo. Etiquetas Break y Continue

- **break** hace que dejen de ejecutarse las instrucciones del resto del bloque y si es el bloque de un ciclo, se termina el ciclo.
- **continue** deja de ejecutar las instrucciones del bloque y el ciclo continúa en el siguiente paso.
- Por ejemplo:

```
uno: for( ) {
    dos: for( ) {
        continue; // seguiría en el bucle interno
        continue uno;//seguiría en el bucle principal
        break uno; // se saldría del bucle principal
    }
}
```

Control de Flujo

- En el código de una función siempre hay que ser consecuentes con la declaración que se haya hecho de ella.
  - Por ejemplo, si se declara una función para que devuelva un entero, es imprescindible que se coloque un *return* final para salir de esa función, independientemente de que haya otros en medio del código que también provoquen la salida de la función.
  - En caso de no hacerlo se generará un *Warning*, dando un error de compilación.
  - Por Ejemplo:

```
int func() {
        if( a == 0 )
                return 1;
        return 0; // es imprescindible porque se retorna un entero
}
```

  - ***Ejercicio:*** **Crear un programa que nos diga si el único argumento que se le pasa es par o impar sin utilizar la palabra "else".**

Excepciones en Java

- **Las excepciones son una manera cómoda y muy útil de tratar las anomalías previsibles en el funcionamiento de un programa.**
- **Cuando un método se encuentra con una situación que le impide terminar su trabajo, arroja una excepción. El programador puede (y normalmente debe) controlar esas excepciones.**
- **try-catch**

```
try {
    sentencias;
} catch ( Exception e ) {
    sentencias;
}
```

  - Java implementa excepciones para facilitar la construcción de código robusto.
  - Java proporciona muchas excepciones predefinidas.

Excepciones en Java

- Ejemplo de uso de excepciones con try-catch:

```
public class Test {
    public static void main(String[] args) {
        int a=10, b=0, c;
        try{
            c=a/b;
        } catch(ArithmeticException e){
            System.out.println("Error: "+e.getMessage());
            return;
        }
        System.out.println("Resultado:"+c);
        }
    }
```

Excepciones en Java

- **try-finally**

```
try {
    sentencias;
} finally {
    sentencias;
}
```

  - Sirve para ejecutar un bloque de instrucciones antes de salir del otro, independientemente de cómo se hizo la salida.

Excepciones en Java

- Se pueden usar construcciones compuestas como
  - ```
    try {
        sentencias;
    } catch (Exception e) {
        sentencias;
    } finally {
        sentencias;
    }
    ```
  - ```
    try {
      sentencias;
    } catch (type1Exception e1) {
      sentencias;
    } catch (type2Exception e2) {
      sentencias;
    }
    ...
    } catch (typenException en) {
      sentencias;
    } finally {
      sentencias;
    }
    ```
- En estas se ejecuta el código de la excepción correspondiente y de cualquier manera, haya una excepción o no, el código de finally siempre se ejecuta.
- Cuando se recibe una excepción e con catch, basta usar el método e.toString() o e.toPrintStackTrace() para obtener información sobre la excepción.
- El programador puede crear sus propias excepciones si lo estima conveniente.

Excepciones en Java

- Ejemplo de uso de excepciones con try-catch-finally:

```
import java.io.*;
public class existe {
        public static void main(String[] args) {
                int i=0;
                while (i<args.length) {
                        System.out.print(args[i]);
                        try {
                          RandomAccessFile f=new RandomAccessFile(args[i],"r");
                          f.close();
                          System.out.print(" si");
                        } catch (IOException e) {
                          System.out.print(" no");
                        } finally {
                          System.out.println(" existe");
                          i++;
                        }
                }
        }

}
```