La Sénia
Institut d'Educació Secundària
Paiporta

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Programación estructurada

Curso 2023 - 2024

1. Programación Estructurada

- La Programación Estructurada es un modelo de programación que se basa en el uso de **funciones**, **procedimientos** y las **estructuras de control.**
- Las **estructuras de control** facilitan las tareas de mantenimiento, la reutilización de código y la mejora y corrección de errores
- Las estructuras básicas son: **secuencial**, alternativa o de **selección** y repetitiva o **iterativa**.

2. Estructuras Secuenciales

- Las sentencias del programa se ejecutan de forma **secuencial:** una tras otra y en el mismo orden escrito.
- Cada una de las sentencias o instrucciones están separadas por el carácter punto y coma (**;**).
- Las instrucciones se suelen agrupar en bloques. El bloque de sentencias inicia con el carácter llave de apertura (**{**) y finaliza con el carácter llave de cierre (**}**). **En Java**, si el bloque es una única sentencia no es obligatorio su uso, aunque sí recomendable.

Ejemplo:

```
{
    instrucción 1;
    instrucción 2;
    instrucción 3;
}
```

Ejemplo: Escribe un programa que pide 2 números enteros y los muestra por pantalla.

2. Estructuras Secuenciales

```java
import java.util.Scanner;

public class Secuencial {

    public static void main(String[] args){
        //Declaración de variables
        int n1, n2;
        Scanner sc = new Scanner(System.in);
        //Leer el primer número
        System.out.println("Introduce un número entero: ");
        n1 = sc.nextInt();
        //Leer el segundo número
        System.out.println("Introduce otro número entero: ");
        n2 = sc.nextInt();
        //Mostrar resultado
        System.out.println("Los números son: " + n1 + " y " + n2);
    }
}
```

3. Estructuras de Selección

- La estructura de selección, alternativa o condicional permite al programa bifurcar el flujo de ejecución dependiendo de una expresión (**condición**).
- La condición se evalúa como booleano (true o false).
- Se puede diferenciar entre **selección simple**, **selección doble** y **selección múltiple**.

En Java:

Selección simple: if …

Selección doble: if … else … u operador condicional ? :

Selección múltiple: if … else if … else o usando switch case

3. Estructuras de Selección

Selección simple:

- Se evalúa la condición y si ésta se cumple se ejecuta una determinadas instrucciones. En caso contrario se saltan esas instrucciones.

```
if (condicion) {
   instrucción 1
   instrucción 2
   .......
}
```

Nota: Siempre usar las llaves { } para todos los bloques

Ejemplo: Programa que pide la edad al usuario y determina si es mayor de edad.

3. Estructuras de Selección

```java
import java.util.Scanner;

public class Edad {

    public static void main(String[] args){
        //Declaración de variables
        int edad;
        Scanner sc = new Scanner(System.in);
        //Leer la edad
        System.out.println("Introduce tu edad: ");
        edad = sc.nextInt();
        //Comprobar si es mayor de edad
        if (edad >= 18){
            System.out.println("Eres mayor de edad");
        }
    }
}
```

3. Estructuras de Selección

Selección doble:

- Se evalúa la condición y si se cumple se ejecutan unas instrucciones. Si no se cumple se ejecutan otras instrucciones.

```
if (condicion) {
   instrucciones 1
}else{
   instrucciones 2
}
```

Operador condicional '? :':

Es un operador ternario, igual que if-else, para una instrucción.

<condición> ? <InstruccionTrue> : <InstruccionFalse>;

Ejemplo: Programa que pide por teclado un número y devuelve si es par o impar.

3. Estructuras de Selección

```java
import java.util.Scanner;

public class Par {

    public static void main(String[] args){
        //Declaración de variables
        int num;
        Scanner sc = new Scanner(System.in);
        //Leer el número
        System.out.println("Introduce el número: ");
        num = sc.nextInt();
        if (num % 2 == 0){
            System.out.println("El número " + num + " es par");
        }else{
            System.out.println("El número " + num + " es impar");
        }

        //System.out.println((num%2)==0 ? "Es PAR" : "Es IMPAR");
    }
}
```

3. Estructuras de Selección

Selección múltiple:

- se obtiene anidando sentencias if-else. Permite construir estructuras de selección más complejas.

```
if (condicion) {
   Instrucciones 1
}else if(condicion){
   Instrucciones 2
}else{
   Instrucciones 3
}
```

- Cada else se corresponde con el if más próximo que no haya sido emparejado.

Ejemplo: Programa que muestra un saludo según la hora indicada (mañana/tarde/noche).

3. Estructuras de Selección

```
import java.util.Scanner;

public class Saludo {

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

        System.out.println("Introduzca una hora (un valor entero): ");
        hora = sc.nextInt();
        if (hora >= 0 && hora < 12)
            System.out.println("Buenos días");
        else if (hora >= 12 && hora < 21)
            System.out.println("Buenas tardes");
        else if (hora >= 21 && hora < 24)
            System.out.println("Buenas noches");
        else
            System.out.println("Hora no válida");

    }
}
```

Selección múltiple – Instrucción SWITCH:

- Se utiliza para seleccionar una de entre múltiples alternativas.

```
switch (expresión){
   case valor1:
      instrucciones;
      break;
   case valor2:
      instrucciones;
      break;
   default:
      instrucciones;
}
```

1. *Se evalúa la **expresión** y según su valor salta al **case** correspondiente.*
2. *Se ejecutan las **instrucciones** que siguen al case seleccionado hasta que se encuentra un **break** o el final del switch.*
3. *Si ningún case se cumple se ejecuta el bloque **default** (si existe). No es obligatorio y no tiene porqué ponerse siempre al final, aunque es lo habitual.*

- Se puede usar con datos de tipo byte, short, char e int.
- También con tipos enumerados y clases envoltorio (Character, Byte, Short, Integer y String)

Ejemplo: Programa que pide el número del mes y muestra el nombre correspondiente a ese mes

3. Estructuras de Selección

```
import java.util.Scanner;

public class Meses {
    public static void main(String[] args) {
        int mes;
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduzca un numero de mes: ");
        mes = sc.nextInt();
        switch (mes){
            case 1: System.out.println("ENERO"); break;
            case 2: System.out.println("FEBRERO"); break;
            case 3: System.out.println("MARZO"); break;
            case 4: System.out.println("ABRIL"); break;
            case 5: System.out.println("MAYO"); break;
            case 6: System.out.println("JUNIO"); break;
            case 7: System.out.println("JULIO"); break;
            case 8: System.out.println("AGOSTO"); break;
            case 9: System.out.println("SEPTIEMBRE"); break;
            case 10: System.out.println("OCTUBRE"); break;
            case 11: System.out.println("NOVIEMBRE"); break;
            case 12: System.out.println("DICIEMBRE"); break;
            default : System.out.println("Mes no válido");
        }
    }
}
```

4. Estructuras de Repetión

- Permiten ejecutar de forma repetida un bloque específico de instrucciones.
- Las instrucciones se repiten mientras se cumpla una condición, llamada **condición de salida**.
- La condición de salida debe cambiar para evitar una **repetición infinita.**
- Tipos de estructuras repetitivas:

  while

  do - while

  for

  foreach

4. Estructuras de Repetión

while:

- Las instrucciones se repiten mientras la condición sea verdadera.
- La condición se comprueba al principio por lo que las instrucciones podrián no ejecutarse nunca (si la condición es inicialmente falsa) ó más veces.

```
while (condicion){
    instrucciones;
}
instrucciones;
```

1. *Se evalúa la **condición**.*
2. *Si el resultado de la condición es **true** se ejecutan las instrucciones de dentro del while y se vuelve al paso 1.*
3. *Si condición es **false** el bloque no se ejecuta y el programa continúa con las instrucciones a continuación del while.*

Ejemplo: Programa que imprime las líneas de un fichero. Lo recorre hasta que no quede ninguna por imprimir.

3. Estructuras de Selección

```
import java.util.Scanner;
import java.io.*;

public class ImprimirFichero{

    public static void main(String[] args) throws FileNotFoundException{

        Scanner sc = new Scanner(new File("fichero.txt"));
        String s = "";

        //Mientras hay líneas por leer…
        while(sc.hasNext()){

            //…leer la línea del fichero…
            s = sc.nextLine();

            //…e imprimirla por pantalla.
            System.out.println( s );

        }
    }
}
```

4. Estructuras de Repetión

do ... while:

- Las instrucciones se ejecutan mientras la condición sea cierta.
- La condición se comprueba al final del bucle por lo que el bloque de instrucciones se ejecutarán al menos una vez, aunque la condición sea falsa.

```
do {
    instrucciones;
} while (condicion);
instrucciones;
```

1. *Se ejecutan todas las instrucciones a partir de **do{** hasta llegar a la condición.*
2. *Se evalúa la **condición** de **while**.*
3. *Si el resultado es **true** se vuelve al paso 1.*
4. *Si el resultado es **false**, el programa sigue la ejecución con las siguientes instrucciones.*

4. Estructuras de Repetión

Ejemplo: Programa que lee e imprime números hasta que se introduce un 0.

3. Estructuras de Selección

```
import java.util.Scanner;

public class Hasta0 {

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

        do {

            System.out.print("Introduzca un numero: ");

            num = sc.nextInt();

            System.out.print("Has introducido un " + num);

        } while (num != 0);

        System.out.print("FIN DEL PROGRAMA");

    }
}
```

4. Estructuras de Repetión

for:

- Hace que una instrucción o bloque de instrucciones se repitan un número determinado de veces, mientras se cumpla la condición.

```
for( inicialización; condición;incremento/decremento){
    instrucciones;
}
instrucciones;
```

1. *Se inicializa la **variable de control** (inicialización)*
2. *Se evalúa la **condición**.*
3. *Si el resultado es **true**, se ejecutan las instrucciones del bucle.*
4. *Al final del bucle se actualiza la variable de control (**incremento/decremento**) y se vuelve al punto 2.*
5. *Si el resultado es **false**, el programa continúa con las instrucciones después del for.*

4. Estructuras de Repetión

- A continuación de la palabra **for** y entre paréntesis debe haber siempre **tres zonas separadas**:
  - **Inicialización:** La/s variable/s de control del bucle toman su valor inicial. Se realiza una vez y puede haber una o más variables inicializadas.
  - **Condición:** Mientras sea cierta ejecuta el contenido del bucle. Generalmente, la condición compara las variables de control con un valor límite.
  - **Incremento/decremento:** Decrementa o incrementa la/s variable/s de control del bucle.

```
for(int x = 0, y = 1; x < 100 && y < 100; x=x+1, y=y+2) {
    . . .
}
```

4. Estructuras de Repetión

Ejemplo: Función que imprime los 50 primeros números enteros.

3. Estructuras de Selección

```
public class Imprime50 {

    public static void main(String[] args) {

        //Recorre imprimiendo los 50 numeros

        for (int i = 1; i <= 50; i++) {

            //Imprime el número i
            System.out.println( i );

        }
    }
}
```

4. Estructuras de Repetión

foreach (**for** extendido):

- Esta forma de usar el for es muy util para recorrer los objetos de una colección sin especificar el número de elementos.

```
for (TipoARecorrer nombreVariable : nombreDeLaColección ) {
   instrucciones
}
```

```
public class ForEach { // Nuestra clase principal o programa
    public static void main(String[] args) { // Función principal a ejecutar
        String[] listaNombres = { "Alex", "Ana", "Luisa", "Jose" };
        for (String nombre : listaNombres) {
            System.out.println(nombre);
        }
    }
}
```

4. Estructuras de Repetión

- Java permite la posibilidad de construir bucles infinitos, los cuales se ejecutarán indefinidamente, a no ser que provoquemos su interrupción. Tres ejemplos:

```
for(;;){
    instrucciones
}

for(;true;){
    instrucciones
}

while(true){
    instrucciones
}
```

¡¡¡¡Cuidado!!!!

Los bucles infinitos también suelen producirse de forma involuntaria al no modificar correctamente la condición de salida de un bucle.
Un ERROR muy común para el programador novel.

4. Estructuras de Repetión

Bucles anidados:

- Son aquellos que incluyen instrucciones for, while o do-while unas dentro de otras.
- Debemos tener en cuenta que las variables de control que utilicemos deben ser distintas.
- Los anidamientos deben mantener una estructura anidada dentro de otra completamente.

```
for(int i = 0; i < 10; i++) {
    for(int j = 0; j < 10; j++) {
        System.out.println("El número es "+i+j);
    }
    System.out.println("Fin del programa");
}
```

5. Estructuras de Salto

- Las **estructuras de salto** son instrucciones que permiten romper el orden natural de ejecución de nuestros programas, pudiendo saltar a un determinado punto o instrucción.
- Las instrucciones de salto son: **break**, **continue** y **return**
- En estructuras de repetición:
  - para salir de forma abrupta de un bucle (**break**).
  - para finalizar la iteración y evaluar la siguiente (**continue**).
- **break** se usa para interrumpir la ejecución de un switch.
- **return** se utiliza para finalizar una función o método.

5. Estructuras de Salto

break:

- Se utiliza para interrumpir la ejecución de una estructura de repetición o de un switch.
- El flujo del programa continúa en la sentencia inmediatamente posterior a la estructura de repetición o switch.

```
public class SaltoBreak {
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            System.out.println("Dentro del bucle "+i);
            break;
            System.out.println("Nunca lo escribirá");
        }
        System.out.println("Tras el bucle");
    }
}
```

5. Estructuras de Salto

Continue:

- Aparece dentro de una estructura de repetición.
- Finaliza la iteración para volver a evaluar la condición inicial y continuar con la siguiente iteración.

```
public class SaltoContinue {
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            System.out.println("Dentro del bucle "+i);
            continue;
            System.out.println("Nunca lo escribira");
        }
        System.out.println("Tras el bucle");
    }
}
```