ESTRUCTURAS DE CONTROL

MÓDULO: Programación en Java

ÍNDICE DE CONTENIDOS

1. **OBJETIVOS**
2. **INTRODUCCIÓN**
3. **ESTRUCTURAS DE CONTROL**
   - 3.1. ESTRUCTURA CONDICIONAL SIMPLE
   - 3.2. ESTRUCTURA CONDICIONAL MÚLTIPLE
   - 3.3 ESTRUCTURAS REPETITIVAS
4. **ESTRUCTURAS CÍCLICAS**
   - 4.1. BUCLES

4. **RESUMEN/IDEAS CLAVE**
5. **BIBLIOGRAFÍA**
6. **RECURSOS WEB DE INTERÉS**
7. **ACTIVIDADES**

1.OBJETIVOS

- Aprender a utilizar las estructuras condicionales que tanta versatilidad nos aportan.
- Familiarizarse con la sintaxis y semántica de los bucles, una de las estructuras más importantes dentro de la programación.
- Aprender a diseñar códigos simples con las estructuras que se abordan en este capítulo.
- Entender y utilizar la estructura Switch, cuando tenemos diferentes valores para una misma variable.

2.INTRODUCCIÓN

Dominar el uso de Microsoft Word para el tratamiento de texto, incluyendo el formato básico y avanzado, la inserción de objetos, la creación y gestión de tablas, la organización del texto, la revisión y autocorrección de textos, el uso de plantillas y formularios, la gestión de correspondencia, la creación y aplicación de macros, así como aspectos relacionados con la seguridad y el trabajo en equipo.

3. ESTRUCTURAS DE CONTROL
3.1. Estructura condicional simple

- **Sentencia condicional simple:**

Recuerda que en pseudocódigo utilizamos el **Si-Entonces**:

Si (cond) **Entonces**
    sentencia(s)
Fin-Si

- Tras evaluar una expresión lógica, ejecuta una serie de acciones si el resultado es verdadero.

```
if (expresión-lógica) {
instrucción 1;
...
}
```

```
if (num < 0){
System.out.println("Es negativo");
}
```

3. Estructuras de control
3.2 Estructura condicional doble

- **Sentencia condicional doble:**

Es igual que la anterior, sólo que se añade un apartado **else** que contiene las acciones que se ejecutarán si la **expresión lógica** evaluada es falsa.

```
if (expresión-lógica) {
instrucción 1;
...
}
else {
instrucción 2;
...
}
```

```
if (num % 2 == 0){
        System.out.println("Es par");
}
else
        System.out.println("NO Es par");
}
```

3. Estructuras de control

3.3 Estructura condicional múltiple Switch

- **Sentencia condicional múltiple**:

Permite evaluar varios valores a la vez. En realidad sirve como sustituta de algunas expresiones de tipo if-else-if, pero SOLO permite evaluar **expresiones enteras** (no lógicas), y de tipo **cadena (String)**

```
switch (expresión) {
case n1:
instrucciones para el valor n1;
case n2:
instrucciones para el valor n2;
[default:
instrucciones si no coincide con n1 o n2;]
}
```

3. Estructuras de control

3.3 Estructura condicional múltiple Switch

DIGI TECH
Centro Oficial FP Digital & Tech

- **Sentencia condicional múltiple (Switch):**

- Esta instrucción evalúa una expresión (que debe ser **short**, **int**, **byte**, **char o String**), y según el valor de la misma, ejecuta instrucciones.

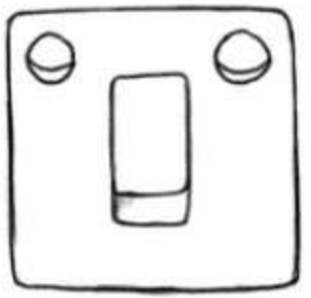
- Cada case contiene un valor. Si al evaluar la expresión el resultado es igual a ese valor, **se ejecutan las instrucciones de ese *case* y de los siguientes**.

- La instrucción **break** se utiliza para salir del switch. Es decir, si queremos que para un determinado valor se ejecuten las instrucciones de un apartado case y sólo las de ese apartado entonces habrá que finalizar ese case con un break.

- El bloque **default** sirve para ejecutar instrucciones para los casos en los que la expresión no se ajuste a ningún case.

3. Estructuras de control

3.3 Estructura condicional múltiple Switch



- Ejemplo 1

```
switch (dia){
case 1:
      System.out.println("es lunes... (sniff)");
break;
case 2:
      System.out.println("Es martes");
break;
case 3:
      System.out.println("Es Miercoles!");
break;
case 4:
      System.out.println("Es Jueves!!");
break;
case 5:
      System.out.println("ES VIERNES!!!");
break;
default:
      System.out.println("**NO HAY CLASE **");
}
```

3. Estructuras de control
3.4 Expresión con String

- **Ejemplo 2: La expresión es una cadena (objeto String)**

- Como String no es un tipo primitivo sino un objeto, éste podría tener el valor **null**.

- La sentencia switch, al comprobar el valor de la cadena, si ésta es null, lanzará una excepción *NullPointerException*.

- Para evitar que se produzca una excepción en ejecución, habrá que comprobar antes el caso de null.

3. Estructuras de control
3.4 Expresión con String

- **Ejemplo 2: La expresión es una cadena (objeto String)**

```
if (mes != null){
      switch (mes){
      case “Enero”:
            System.out.println("Frío");
      break;
      case “Febrero”:
            System.out.println("Más frío” ");
      break;
      ...
      }
}
```

4. Estructuras de cíclicas
4.1 Bucles

Recuerda que en pseudocódigo utilizamos el **Mientras**: repetía un conjunto de sentencias mientras la condición fuese verdadera.

- **Sentencia while**

La sentencia while permite crear bucles. Un bucle es un conjunto de sentencias que se repiten **mientras se cumpla una determinada condición** (la expresión lógica se evalúa a true).

Por tanto, NO se sale de un bucle while hasta que la condición que se evalúa sea falsa.

Esta condición **se evalúa antes** de entrar dentro el bucle, por tanto, el cuerpo del bucle podrá no ejecutarse nunca si la primera vez la condición es falsa.

4. Estructuras cíclicas
4.1 Bucles

- **Sentencia while**

```
while (expresión lógica) {
        instrucciones;
}
```

Ejemplo 1:

```
// Escribe los números del 0 al 99
int i=0;
while (i<100){
        System.out.printf("%d\n", i);
        ++i;
}
```

Ejemplo 2:

```
// Bucle infinito, escribe "..." continuamente
while (true){
        System.out.printf("...", );
}
```

4. Estructuras cíclicas
4.1 Bucles

- **Sentencia for**

Es una sentencia muy similar al while y funciona de la misma manera ejecuta instrucciones mientras la **condición sea verdadera**.

Es algo más compleja porque está pensada para rellenar arrays o para ejecutar instrucciones controladas por un contador. La ventaja que tiene es que el código se reduce.

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

- **inicialización:** aquí se escribe la instrucción (o instrucciones) que normalmente inicializan variables. Se ejecutan **solo la primera vez**.

- **condición:** aquí se escribe una expresión lógica que se evalúa para ver si se sigue ejecutando el bucle o no.

4. Estructuras cíclicas
4.1 Bucles

- **Sentencia for**

- **incremento:** aquí se escribe la instrucción (o instrucciones) que normalmente incrementan o decrementan el valor de alguna variable contador. Se ejecuta **antes de volver a evaluar la condición**, una vez que se han ejecutado las instrucciones del bucle.

Ejemplo:

```
// Escribe una linea con tres asteriscos
int lado=3;
for (int columna=0; columna<lado; columna++){
      System.out.printf("%c", '*');
}
```

4. Estructuras cíclicas
4.1 Bucles

- **Sentencia for**

1)se ejecuta la instrucción de inicialización: **columna = 0**

2) se evalúa la condición: **¿columna < lado?** 0<3 es true, **escribe** en consola

3) se ejecuta la instrucción de incremento: **columna++**

4) se evalúa la condición: **¿columna < lado?** 1<3 es true, **escribe** en consola

5) se ejecuta la instrucción de incremento: columna++

6) se evalúa la condición: **¿columna < lado?** 2<3 es true, **escribe** en consola

7) se ejecuta la instrucción de incremento: **columna++**

8) se evalúa la condición: **¿columna < lado?** 3<3 es false por tanto **termina**.

4. Estructuras cíclicas
4.1 Bucles

- **Sentencia do-while**

La única diferencia con respecto a la sentencia **while**, es que la expresión lógica **se evalúa después** de haber ejecutado las instrucciones del bucle. Es decir, **el bucle se ejecuta al menos una vez**.

Esta sentencia es útil para la **validación** de datos de entrada.

Ejemplo 1:

```
do {
      valorTecleado = JOptionPane.showInputDialog
      ("Teclea número mayor de 0");
      num = Integer.parseInt(valorTecleado);
}while (num<=0);
```

lee un número por tecleado y lo convierte a entero, hasta que deje de cumplirse la condición, es decir, hasta que el número sea mayor que cero.

4. Estructuras cíclicas
4.1 Bucles

- **Sentencia do-while**

Ejemplo 2:

```
int cont=-1;
do {
      System.out.println(cont);
      cont--;
}while(cont>=0);
```

1) se escribe el contenido de cont que es -1 (no se evalúa la condición).
2) se decrementa en 1 la variable cont.
3) ahora se evalúa la condición; como (-2 >= 0) es falso, el bucle termina.

RESUMEN DE LA UNIDAD

- Permiten ejecutar bloques de código basados en la evaluación de condiciones booleanas, controlando el flujo del programa según los resultados verdaderos o falsos de las expresiones lógicas.
- Los bucles nos permiten repetir un bloque de código mientras se cumpla una condición específica, con for siendo útil para iteraciones con un número conocido de repeticiones y while y do-while adecuados para repeticiones basadas en condiciones.
- La estructura Switch permite seleccionar y ejecutar uno de entre varios bloques de código basándose en el valor de una expresión, siendo útil para manejar múltiples casos posibles de manera más organizada y legible que múltiples sentencias if-else.Utilizar el pensamiento computacional para aprender la dinámica de la metodología de la programación.
- El uso de bucles, nos permite ejecutar un conjunto de instrucciones varias veces.

BIBLIOGRAFÍA/WEBGRAFÍA

- Thierry Richard. “Los fundamentos del lenguaje Java”.
- Vozmediano,A.M. “Java para novatos”.
- Jimenez, Alfonso.”Aprender a programar en Java”.
- Gervás, Luc. “Aprender los fundamentos del lenguaje Java”