TEMA 3.- ESTRUCTURAS BÁSICAS DE CONTROL.

Objetivos

- Conocer y saber aplicar las estructuras básicas del lenguaje de programación Java.
- Controlar las excepciones básicas que pueda generar un programa.
- Reconocer la importancia de la documentación en el desarrollo de aplicaciones.
- Conocer el proceso de prueba y depuración de programas.

Contenidos

3.1.-Estructuras de Selección.

Sentencia if.

En Java hay tres tipos de estructuras if.

a) if.

Si se cumple una condición ejecuta unas sentencias.

```
if (condicion) {
      sentencias;
}
```

b) if else.

Si se cumple una condición ejecuta unas sentencias y el caso contrario ejecuta otras.

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

c) if else if.

Si se cumple una condición ejecuta unas sentencias y el caso contrario comprueba otra condición ejecutando unas sentencias si se cumple y así sucesivamente.

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

Ejemplos:

Si el sueldo del empleado es mayor de 1500 euros calcular una retención del 12% sobre el sueldo.

```
if (sueldo > 1500) {
      retencion = sueldo * 0.12;
      System.out.println("La retención es de: " + retencion);
}
```

Si el sueldo del empleado es mayor de 1500 euros calcular una retención del 12% sobre el sueldo en caso contrario calcular una retención del 5%.

```
if (sueldo > 1500) {
      retencion = sueldo * 0.12;
} else {
      retencion = sueldo * 0.05;
}
System.out.println("La retención es de: " + retencion);
```

Si el sueldo del empleado es menor de 1200 euros calcular una retención del 12% sobre el sueldo en caso contrario si el sueldo es mayor o igual a 1200 y menor o igual a 1500 calcular una retención del 18% sobre el sueldo y si no calcular una retención del 25% sobre el sueldo.

```
if (sueldo < 1200) {
      retencion = sueldo * 0.12;
} else if (sueldo >= 1200 && sueldo <= 1500){
      retencion = sueldo * 0.18;
} else {
      retencion = sueldo * 0.25;
}
System.out.println("La retención es de: " + retencion);
```

Sentencia switch.

Evalúa una expresión y según el valor que tome ejecuta un bloque de sentencias.

```
switch (expresion){
      case valor1: sentencias1; break;
      case valor1: sentencias1; break;
            :     :     :     :
      case valorn: sentenciasn; break;
      [default: sentenciasdef;]
}
```

Si no se escribe la sentencia break, el programa seguirá ejecutando las siguientes sentencias hasta el fin del switch.

Ejemplo:
Si un empleado tiene categoría laboral 1 se le aplica un 10% de retención sobre el sueldo, si tiene categoría laboral 2 se le aplica un 15% de retención sobre el sueldo, si tiene categoría laboral 3 se le aplica un 20% de retención sobre el sueldo, si tiene categoría laboral 4 se le aplica un 25% de retención sobre el sueldo el cualquier otro caso se le aplica un 30% de retención sobre el sueldo.

```
switch (catLaboral){
      case 1: retencion = sueldo * 0.10; break;
      case 2: retencion = sueldo * 0.15; break;
      case 3: retencion = sueldo * 0.20; break;
      case 4: retencion = sueldo * 0.25; break;
      default: retencion = sueldo * 0.30;
}
```

3.2.- Estructuras de repetición.

Las estructuras de repetición o bucles son utilizadas cuando unas sentencias han de ser ejecutadas cero, una o más veces.
Hay que tener cuidado con los bucles infinitos que no terminan nunca pues el programa se seguirá ejecutando y el equipo se quedará colgado.

a) Bucle while.

El bucle **while** se utiliza cuando se tiene que ejecutar un grupo de sentencias un número determinado de veces. Las sentencias podrían no llegar a ejecutarse ya que inicialmente podría no cumplirse la condición de ejecución de bucle.

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

Ejemplo:

```
int contador = 1;
while (contador<=10){
      System.out.println(contador);
      contador++;
}
```

Ejemplo:

```
int contador = 0;
String[] poblacion = {"Valencia", "Castellón", "Alicante"};
while (contador <=2) {
      System.out.println(poblacion[contador]);
      contador++;
}
```

b) Bucle do ... while.

El bucle **do ... while** se utiliza cuando se tiene que ejecutar un grupo de sentencias un número determinado de veces. Las sentencias se ejecutan al menos una vez ya que la comprobación de la condición de salida del bucle se encuentra después de las sentencias del bucle.

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

Ejemplo:

```
int contador = 1;
do {
      System.out.println(contador);
      contador++;
} while (contador<=10);
```

Ejemplo:

```
int contador = 0;
String[] poblacion = {"Valencia", "Castellón", "Alicante"};
do {
      System.out.println(poblacion[contador]);
      contador++;
} while (contador<=2);
```

b) Bucle for.

El bucle **for** se utiliza cuando se tiene que ejecutar un grupo de sentencias un número fijo y conocido de veces. La sentencia **for** se puede conseguir utilizando el bucle while.

```
for(inicializacion; condicion; incremento){
      sentencias;
}
```

Ejemplo:

```
int contador;
for(contador = 1; contador<=10; contador++){
      System.out.println(contador);
}
```

Ejemplo:

```
int contador;
            for(contador = 10; contador>=1; contador--){
                  System.out.println(contador);
            }
```

Ejemplo:

```
            int contador;
            for(contador = 1; contador<=9; contador+=2){
                  System.out.println(contador);
            }
```

Ejemplo:

```
      int contador;
      String[] poblacion = {"Valencia", "Castellón", "Alicante"};
      for(contador = 0; contador<=2; contador++){
            System.out.println(poblacion[contador]);
      }
```

3.3.- Estructuras de salto.

a) Sentencias break y continue.

La sentencia **break** se puede utilizar tanto en estructuras de selección como en estructuras de repetición y permite salir de un bloque de sentencias.
La sentencia **continue** se puede utilizar solo en estructuras de repetición y permite saltar desde el bloque de sentencias a la sentencia de evaluación de la condición.

Ejemplo:

```
            int contador = 0;
            while (contador < 10){
                  contador++;
                  if (contador==5) {
                        break;
                  }
                  System.out.println(contador);
            }
```

Ejemplo:

```
int contador = 0;
while (contador < 10){
      contador++;
      if (contador==5) {
            continue;
      }
      System.out.println(contador);
}
```

b) Sentencias break y continue con etiquetas de salto.

Podemos utilizar las sentencias break y continue con etiquetas de salto. Las etiquetas de salto tienen el formato:

nombre_etiqueta:

Ejemplo de blucles anidados:

```
int contador_int;
int contador_ext = 1;
while (contador_ext <= 10){
      System.out.println("-----------------------------");
      System.out.println("Exterior: "+contador_ext);
      System.out.println("-----------------------------");
      for(contador_int=1;contador_int <= 5;contador_int++){
            System.out.println("Interior: "+contador_int);
      }
      contador_ext++;
}
```

Ejemplo de blucles anidados con sentencia break de etiqueta:

```
int contador_int;
int contador_ext = 1;
bucleext:
while (contador_ext <= 10){
      System.out.println("-----------------------------");
      System.out.println("Exterior: "+contador_ext);
      System.out.println("-----------------------------");
      bucleint:
      for(contador_int=1;contador_int <= 5;contador_int++){
            if (contador_ext == 5) {
                  break bucleext;
            }
            System.out.println("Interior: "+contador_int);
      }
      contador_ext++;
}
```

c) Sentencias return.

La sentencia return permite salir de una función o método que se esté ejecutando y permite devolver un valor.

```
return valor;
```

Ejemplo:

```
int contador;
for(contador=1;contador <= 10;contador++){
      if (contador == 5) {
            return;
      }
      System.out.println(contador);
}
```

3.4.- Control de Excepciones.

Permite al programador controlar la ejecución del programa evitando que este falle de forma inesperada.

```
try {
      sentencias_protegidas;
} catch (excepcion_1) {
      senrencias_de_control;
}
      :     :     :     :
catch (excepcion_1) {
      senrencias_de_control;
} [finally{
      senrencias_de_control_opcional;
}]
```

El programa intentará protege las sentencias del bloque **try** y si ocurre un erro controlará la excepción mediante los bloques **catch**. El bloque **finally** es opcional y si existe se ejecutará siempre.

Ejemplo:

```
      int dividendo=10;
      int divisor=0;
      int resultado;
      resultado = dividendo/divisor;
      System.out.println("El resultado es "+resultado);
```

La ejecución de este programa muestra el error de división por 0.

Ejemplo tratamiento de excepción división por 0:

```
int dividendo=10;
int divisor=0;
int resultado;
try {
      resultado = dividendo/divisor;
} catch (ArithmeticException e){
      System.out.println("No se permite división por 0");
      return;
}
System.out.println("El resultado es "+resultado);
```

Excepciones más usuales.

NOMBRE	DESCRIPCION
FileNotFoundException	Lanza una excepción cuando el fichero no se encuentra.
ClassNotFoundException	Lanza una excepción cuando no existe la clase.
EOFException	Lanza una excepción cuando llega al final del fichero.
ArrayIndexOutOfBoundsException	Lanza una excepción cuando se accede a una posición de un array que no exista.
NumberFormatException	Lanza una excepción cuando se procesa un numero pero este es un dato alfanumérico.
NullPointerException	Lanza una excepción cuando intentando acceder a un miembro de un objeto para el que todavía no hemos reservado memoria.
IOException	Generaliza muchas excepciones anteriores. La ventaja es que no necesitamos controlar cada una de las excepciones.
Excepcion	Es la clase padre de IOException y de otras clases. Tiene la misma ventaja que IOException.
ArithmeticException	Se lanza por ejemplo, cuando intentamos dividir un número entre cero.