MÓDULO PROFESIONAL PROGRAMACIÓN

UD 2 – Estructuras de control de flujo

Estructuras de control de flujo

Hasta el momento, todos los programas y aplicaciones que hemos realizado y abordado, no alteraban el curso de ejecución de las sentencias. Es decir, las **sentencias se ejecutaban ordenadamente desde la primera a la ultima dentro de nuestro código fuente**.

Este tipo de estructura de programa se denomina:

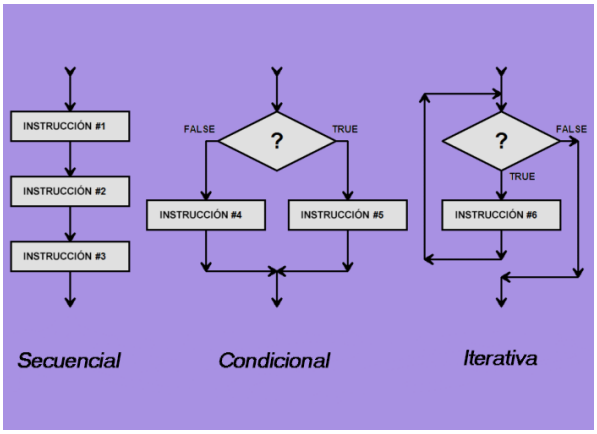
a) **Secuencial**.

En este tema se van a abordar las principales estructuras de control que nos van a permitir alterar el orden en que se ejecutan nuestras sentencias y estas estructuras son:

b) **Selección,** decisión o condicionales

c) **Repetición,** iterativa o bucles

Estructuras de control de flujo



Estructuras de control de flujo

El lenguaje Java las **estructuras de control de flujo** con las que se puede crear desde una sentencia condicionada hasta bloques alternativos, bloques repetitivos y combinaciones anidadas de todos estas son:

Estructuras de **selección**

- La sentencia IF-ELSE
- La sentencia SWITCH

Estructuras de **repetición**

- Bucle WHILE
- Bucle Do-WHILE
- Bucle FOR

Sentencias de selección

Las sentencias de selección nos permiten decidir si queremos ejecutar o no un bloque de código entre una o más opciones, mediante la evaluación de una condición.

- Sentencias **if-else**: Nos permiten decidir si queremos ejecutar o no un fragmento de código dependiendo de si se cumple o no una condición
- Sentencias **switch**: Permite ejecutar un bloque diferente de código para cada valor posible que pueda tomar una expresión

SENTENCIA IF-ELSE

La sentencia if permite decidir si ejecutar o no un fragmento de código en función del resultado de una condición

```
if (condición) {
    bloque_de_sentencias;
}
```

Si se cumple **condición** se ejecutará las instrucciones contenidas entre las llaves. Por el contrario si condición no se cumple Java ignorará este fragmento de código y continuará ejecutando después de las llaves.

SENTENCIA IF-ELSE

Para evaluar estas condiciones suelen utilizarse los operadores de relación (comparación) y lógicos: ==, != , < , > , <=, >=, y &&, ||, !

```
public class EstructuraIfElse {

    public static void main(String[] args) {

        double  nota = 7.5;

        if (nota >= 5) {
            System.out.println("aprobado");
        }

        if (nota < 5) {
            System.out.println("suspenso");
        }

    }
}
```

Nota: los bloques alternativos deben ir entre {} si afectan a más de una sentencia

SENTENCIA IF-ELSE

¿ Qué se muestra por pantalla si x tiene un valor negativo?

```
if (x > 0)
    System.out.println("x is positive");
    System.out.println("x is not zero");
```

SENTENCIA IF-ELSE

¿Qué se muestra por pantalla?

```
int x = 1;
if (x % 2 == 0); {
    System.out.println("x is even");
}
```

SENTENCIA IF-ELSE

¿Qué se muestra por pantalla?

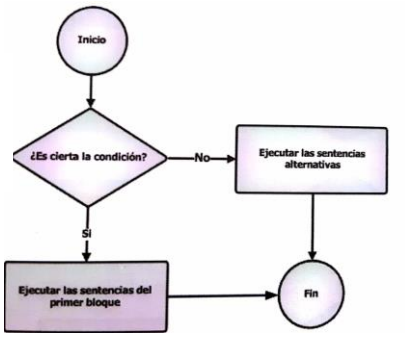
```
int x = 1;
if (x % 2 == 0); {
    System.out.println("x is even");
}
```

```
int x = 1;
if (x % 2 == 0)
    ;  // empty statement
{
    System.out.println("x is even");
}
```

SENTENCIA IF-ELSE

La sentencia **if** puede completarse de manera **opcional** con el bloque **else**. De esta forma, se ejecutarán las sentencias del bloque if si la condición es cierta y las del bloque else si no lo son:

```
if (condición) {
    bloque_de_sentencias_si_condición_es_cierta;
} else {
    bloque_de_sentencias_si_condición_NO_es_cierta;
}
```



SENTENCIA IF-ELSE

Por tanto utilizando un bloque de instrucciones **if-else** podemos llevar a cabo unos pasos u otros en función de la evaluación de una condición

```
public class EstructuraIfElse {

    public static void main(String[] args) {

            double nota = 7.5;
            if (nota >= 5) {
                System.out.println("aprobado");
            } else {
                System.out.println("suspenso");
            }
    }
}
```

¿Cuál es más eficiente?

Estos dos programas producen la misma salid. ¿cuál es mejor?

```java
public class EstructuraIfElse {

    public static void main(String[] args) {

        double  nota = 7.5;

        if (nota >= 5) {
            System.out.println("aprobado");
        }

        if (nota < 5) {
            System.out.println("suspenso");
        }

    }
}
```

```java
public class EstructuraIfElse {

    public static void main(String[] args) {

            double nota = 7.5;
            if (nota >= 5) {
                System.out.println("aprobado");
            } else {
                System.out.println("suspenso");
            }
    }
}
```

Actividad

1. Realiza una aplicación que pida al usuario su **edad**. La aplicación comprobará si el usuario es mayor de edad o no y en función de esto indicará por pantalla al usuario si puede ejercer su derecho a voto en las próximas elecciones
2. Realiza una aplicación que pida al usuario una talla de una prenda y que diga si existe esa talla. Las tallas menor y mayor serán definidas como **constantes**. La menor es 36 y la mayor 54.

SENTENCIA IF-ELSE

Los bloques **if-else** se pueden **anidar** cuantas veces se desee para comprobar más condiciones. Es decir las instrucciones que se ejecutan en un if o un else pueden contener a su vez más bloques **if-else**

```
public class EstructuraIfElseAnidada {

    public static void main(String[] args) {

            double nota = 7.5;

            if (nota >= 5)
                if (nota<7)
                    System.out.println("aprobado");
                else
                    System.out.println("notable");
            else
                System.out.println("suspenso");
    }
}
```

Actividad

3. Modifica la aplicación del **derecho a voto,** para que además pida el **sexo (H o M).** La aplicación deberá indicar si tiene derecho a voto o no según su edad, pero añadiendo al mensaje **Sr.** O **Sra** en función del valor leído.

Actividad

4. Realiza una aplicación que pida al usuario 2 números enteros y muestre el siguiente menú:

   Calculadora v.1.1
   ---
   1. Sumar
   2. Restar
   3. Multiplicar
   4. Dividir
   5. Salir

Implementa las distintas opciones y muestra el resultado. ¿Qué ocurre si intentamos dividir por 0?

Actividad

5. Modifica la aplicación de las notas para que pida al usuario su nota y las salidas en función de la nota sean:

   - Suspenso: Entre 0 y menor que 5.
   - Aprobado: Mayor o igual que 5 y menor que 6.
   - Bien: Mayor o igual que 6 y menor que 7.
   - Notable: Mayor o igual que 7 y menor que 9.
   - Sobresaliente: Mayor o igual que 9 y menor que 10.
   - Matricula de honor: Igual que 10.

SENTENCIA IF-ELSE

Ámbito de las variables: Una **variable** sólo **existe dentro del bloque de llaves** dentro del que se creó. Es decir si intentamos acceder a ella desde fuera de dicho bloque se producirá un error:

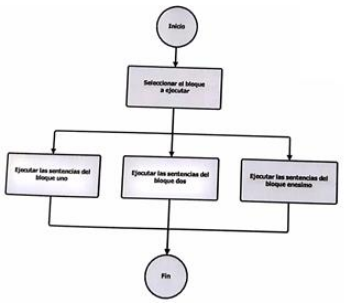
```
import java.util.Scanner;
public class AmbitoVariables {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);
        double num =  sc.nextDouble();
        if (num>0) {
            String valor="es positivo";
        } else {
            String valor="es positivo";
        }
        System.out.println(valor);
    }
}
```

Como sólo existen dentro de un bloque {} pueden existir dos variables con el mismo nombre en el mismo programa siempre que estén en bloques diferentes.

SENTENCIA SWITCH

La sentencia **switch** nos permite elegir entre varios bloques de código cual deseamos ejecutar, dependiendo del valor de una **expresión**:

```
switch (expresion) {
    case valor1:
        conjunto de instrucciones;
        break;
    case valor2:
        conjunto de instrucciones;
        break;
    case valor3:
        conjunto de instrucciones;
        break;
    default:
        conjunto de instrucciones;
        break;
}
```



Además, con el bloque **default** (que es **opcional**) se cubren todos los valores que no estén indicados en un case especifico.

SENTENCIA SWITCH

- La expresión puede ser de tipo **byte, short, int, char** o una enumeración. A partir de JDK7, la expresión también puede ser de tipo **String**.
- Los valores duplicados de case no están permitidos.
- La declaración predeterminada **default** es opcional.
- La declaración de interrupción **break**; se usa dentro del switch para finalizar una secuencia de instrucción.
- La declaración **break**; es opcional. Si se omite, la ejecución continuará en el siguiente **case**.

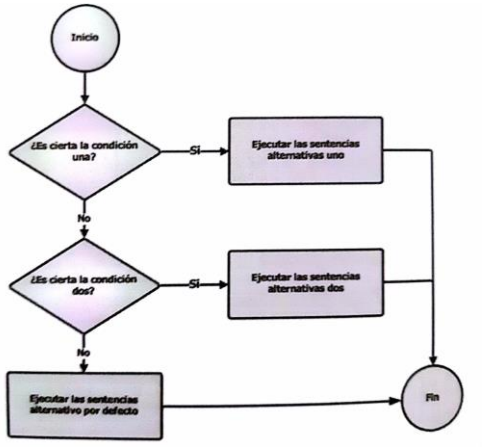
```
import java.util.Scanner;
public class EjemploSwitch {

    public static void main(String[] args) {

            Scanner sc=new Scanner(System.in);
            System.out.println("Introduce el núemro de mes: ");
            short mes = sc.nextShort();
            switch (mes){
                case 1: case 3: case 5: case 7: case 8: case 10: case 12:
                    System.out.println("el mes número " + mes + " tiene 31 días");
                    break;
                case 2:
                    System.out.println("el mes número " + mes + " tiene 29 días");
                    break;
                case 4: case 6: case 9: case 11:
                    System.out.println("el mes número " + mes + " tiene 30 días");
                    break;
                default:
                    System.out.println("Mes inexistente");
            }
    }
}
```

Actividad

6. Modifica el ejercicio de la **calculadora** implementándolo mediante un **switch**.
7. ¿Es posible modificar el ejemplo de las **notas** para implementarla con un **switch**? Si es posible, hazlo.

Sentencia IF-ELSE IF

```
if (condición 1) {
    [bloque de sentencias 1]
} else if (condición 2) {
    [bloque de sentencias 2]
} else if (condición 3) {
    [bloque de sentencias 3]

…

} else {
    [bloque de sentencias alternativo]
}
```



Sentencia IF-ELSE IF

- Si (if) se cumple la primera condición, entonces se ejecuta el bloque de sentencias; si no se cumple esta primera condición, se evalúa la condición siguiente; si se cumple esta segunda condición; se ejecuta el bloque de sentencias correspondiente.
- En el caso de que no se cumpla ninguna condición, se ejecuta el bloque else. Si una condición se cumple ya no se evalúan las condiciones restantes.
- Al finalizar la ejecución de la última sentencia del bloque elegido, se pasa el control a la sentencia siguiente a la última llave de la estructura de bloques alternativos, es decir la primera sentencia después de la llave que cierra el bloque iniciado con if.

Sentencia IF-ELSE IF

Cuando hay **más de 3 o 4 bloques alternativos** debe utilizarse preferentemente la estructura switch ya que resulta más eficiente que la estructura if… else if…else

Esta última tiene que evaluar todas las condiciones hasta que encuentra una cierta, frente a **switch que solo evalúa la variable que determina el bloque una vez, independientemente del número de bloques que hay en la estructura.**

Actividad

```java
public class EjemploIfElseIf {

    public static void main(String[]args) {

        boolean nevando = true;
        int temperatura = -1;

        if (temperatura > 25) {
            // Si la temperatura es mayor que 25 ...
            System.out.println("A la playa!!!");
        } else if (temperatura > 15) {
            // si es mayor que 15 y no es mayor que 25 ..
            System.out.println("A la montaña!!!");
        } else if (temperatura < 5 && nevando) {
            // si es menor que 5 y esta nevando y no es mayor que 15 ni mayor que 25
            System.out.println("A esquiar!!!");
        } else {
            // si la tempera  no es mayor que 25 ni que 15 ni menor que 5 si esta nevando
            System.out.println("A descansar... zZz");
        }

    }
}
```

Actividad

Realizar la relación de ejercicios disponible en Moodle para ejercitar estructuras de selección.

Estructuras de repetición

Los ordenadores se utilizan a menudo para automatizar tareas repetitivas, como la búsqueda de texto en los documentos.

Repetir tareas sin cometer errores es algo que los los ordenadores hacen bien y las personas hacen mal.

En este apartado, aprenderás a utilizar los **bucles** para añadir repetición a tu código.

Estructuras de repetición

Las estructuras de repetición o bucles nos van a permitir ejecutar un bloque de código cero, una o más veces.

Existen tres tipos de bucles:

- Bucle **While**: Permite ejecutar un bloque de código o o más veces.
- Bucle **Do-While:** Permite ejecutar un bloque de código 1 o más veces.
- Bucle **For**: Permite ejecutar un bloque de código un número fijo y conocido de veces.

Estructuras de repetición

Las **estructuras de repetición** o iterativas permiten repetir una misma secuencia de instrucciones varias veces, mientras se cumpla una cierta condición.

Estructuras de repetición

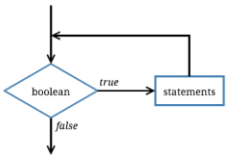
Llamamos **bucle** o ciclo el conjunto de instrucciones que se debe repetir un cierto número de veces, y llamamos **iteración** cada ejecución individual del bucle.

Bucle while

El bucle **while** es una estructura de repetición que permite ejecutar un bloque de código o o más veces mientras se cumpla una condición. El número de veces que se ejecuta el bucle no tiene por que conocerse de antemano.

Sintaxis:

```
while (condición) {
    bloque_de_sentencias;
}
```



El bloque de sentencias se ejecutará **mientras la condición tenga valor verdadero**. Esta condición se verifica al principio de cada iteración. Si la primera vez, justo cuando se ejecuta la sentencia por primera vez, ya no se cumple, no se ejecuta ninguna iteración.

Bucle while

```
import java.util.Scanner;
public class EjemploWhile {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);
        int num=1;
        while (num!=0){
            System.out.println("Pulse 0 para salir ");
            num=sc.nextInt();
        }
    }
}
```

Actividad

8. Realiza una aplicación que pida al usuario un **número del 0 al 9** y que muestre al usuario la **tabla de multiplicar** de dicho número. ¿Cómo se podría hacer para comprobar que el número introducido es correcto y volverlo a pedir si no lo es?
9. Realiza una aplicación que pida al usuario un número entero. El programa imprimirá por pantalla **todos los números desde el 0 hasta el número** introducido.
10. Realiza una aplicación que pida al usuario el **año actual**. Mientras el año leído no sea el actual (2021) la aplicación le volverá a pedir que lo introduzca. Cuando se introduzca el año actual, el programa lo mostrará junto con el número de veces que le ha pedido al usuario que lo introduzca.

Bucle DO-WHILE

El bucle **do-while** es una estructura de repetición muy parecida al bucle while con la diferencia que do-while evalúa la condición al final mientras que while la evalúa al principio. Con esto el bucle do-while asegura que el bloque de código se va a ejecutar al menos una vez.

```
do {
    bloque_de_sentencias;
} while (condición);
```

El bucle do-while se emplea cuando queremos asegurarnos que el bloque de sentencias se ejecuta al menos una vez

BUCLE DO-WHILE

While

```
while (condición) {
    bloque_de_sentencias;
}
```

Do While

```
do {
    bloque_de_sentencias;
} while (condición);
```

BUCLE DO-WHILE

```
import java.util.Scanner;
public class EjemploDoWhile {

    public static void main(String[] args) {

        Scanner teclado=new Scanner(System.in);
        int num;
        do{
            System.out.println("Introduce un número mayor que 100:");
            num=teclado.nextInt();
        } while (num<=100);
    }
}
```

Actividad

11. Realiza una aplicación que pida un **número al usuario hasta que este introduzca el número 0**. Cuando se introduzca el 0 se mostrará al usuario la media de dichos números y cuantos números se han introducido.
12. Los bucles while y do-while son completamente equivalentes. Como podríamos hacer que un bucle while se comporte como un do-while y viceversa. Es decir ¿como podríamos hacer para que el bucle while se ejecute siempre al menos una vez y el bucle do while pueda ejecutarse o veces?

El bucle FOR

```
for(int i=0; i<10; i++){
    potencia=Math.pow(i,2);
    System.out.println(potencia);

}
```

El bucle for es una estructura de repetición que nos permite repetir un bloque de sentencias un número conocido y fijo de veces.

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

Donde:

- **inicialización** es una sentencia que se ejecuta antes de empezar el bucle y que declara la variable que va a controlar el bucle. Se ejecuta una sola vez y puede estar compuesta por varias expresiones separadas mediante el operador *coma* (,). Por ejemplo,

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

- La **condición** se evaluará al comienzo de cada iteración y mientras se cumpla, el bucle se repetirá.
- El **incremento** es la acción a realizar cuando la condición es cierta y al finalizar la iteración actual.

El bucle FOR

1. La **inicialización** se ejecuta una vez al principio del bucle. Es equivalente a la línea anterior a la sentencia while.
2. La **condición** se comprueba cada vez que se itera (se pasa por el bucle). Si es falsa, el bucle bucle termina. En caso contrario, se ejecuta el cuerpo del bucle (de nuevo).
3. Al final de cada iteración, se ejecuta el **incremento** y se vuelve al paso 2.

El bucle FOR

```
import java.util.Scanner;
public class EjemploFor {

    public static void main(String[] args) {

        double potencia;
        for(int i=0; i<10; i++){
            potencia=Math.pow(i,2);
            System.out.println(potencia);

        }
    }
}
```

Actividad

13. Desarrolla una aplicación que pida al usuario **una letra y un número entero**. El programa escribirá **por pantalla la letra tantas veces como indique el número** leído.
14. Repite el ejercicio 8 (**tablas de multiplicar**) utilizando un bucle for
15. ¿Cómo podríamos modificar la estructura de un bucle for para que en lugar de repetirse un número fijo de veces lo haga como máximo ese número de veces?
16. Desarrolla una aplicación mediante un bucle for que calcule las **potencias** de un número leído desde o hasta 10 preguntando después de cada iteración si desea continuar ('s','n'). Si introduce 'n' el bucle terminará independientemente de en que iteración se encuentre.

El bucle FOR

¿Por qué no compila este código?

```
for (int n = 3; n > 0; n--) {
    System.out.println(n);
}
System.out.println("n is now " + n);
```

El bucle FOR

¿Por qué no compila este código?

```
for (int n = 3; n > 0; n--) {
    System.out.println(n);
}
System.out.println("n is now " + n);
```

Si se declara una variable en la parte de inicialización, sólo existe dentro del bucle for. Si necesita utilizar una variable de bucle fuera del bucle, tienes que declararla fuera del bucle

```
int n;
for (n = 3; n > 0; n--) {
    System.out.println(n);
}
System.out.println("n is now " + n);
```

Bucles anidados

Al igual que las sentencias condicionales, los bucles pueden anidarse unos dentro de otros. Los bucles anidados le permiten iterar sobre dos o más variables.

El siguiente programa muestra como se pueden anidar dos o más estructuras for. En el ejemplo, por cada valor de i, j toma los valores desde i a cero. El resultado es el triangulo formado por asteriscos que muestra la figura siguiente:

```
public class EjemploForAnidado {

    public static void main(String[] args) {
            for (int i=0;i<=5;i++){
                for(int j=i; j> 0;j--)
                    System.out.print("*");

                System.out.println("");
            }

    }
}
```

```
*
**
***
****
*****
```

Bucles anidados

¿Qué muestra el siguiente código?

```
for (int x = 1; x <= 10; x++) {
    for (int y = 1; y <= 10; y++) {
        System.out.printf("%4d", x * y);
    }
    System.out.println();
}
```

Bucles anidados

¿Qué muestra el siguiente código?

```
for (int x = 1; x <= 10; x++) {
    for (int y = 1; y <= 10; y++) {
        System.out.printf("%4d", x * y);
    }
    System.out.println();
}
```

Nota: printf() se utiliza para darle formato a la salida.

Lo veremos en detalle más adelante.

```
   1   2   3   4   5   6   7   8   9  10
   2   4   6   8  10  12  14  16  18  20
   3   6   9  12  15  18  21  24  27  30
   4   8  12  16  20  24  28  32  36  40
   5  10  15  20  25  30  35  40  45  50
   6  12  18  24  30  36  42  48  54  60
   7  14  21  28  35  42  49  56  63  70
   8  16  24  32  40  48  56  64  72  80
   9  18  27  36  45  54  63  72  81  90
  10  20  30  40  50  60  70  80  90 100
```

Control de estructuras repetitivas

Las estructuras de repetición no tienen sentido si no es que la condición lógica depende de alguna variable que pueda ver modificado su valor para diferentes ejecuciones.

En caso contrario, la condición siempre valdrá lo mismo para cualquier ejecución posible y usar una estructura de control no será muy útil.

Control de estructuras repetitivas

En este caso, si la condición siempre es **false**, nunca se ejecuta el bucle, por lo que es código inútil.

Pero, para las estructuras de repetición, si la condición siempre es **true** el problema es mucho más grave. Como absolutamente siempre que se evalúa si es precisa una nueva iteración, la condición se cumple, el bucle no se deja nunca de repetir. **¡El programa nunca acabará!**

Control de estructuras repetitivas

Un **bucle infinito** es una secuencia de instrucciones dentro de un programa que itera indefinidamente, normalmente porque se espera que se alcance una condición que nunca se llega a producir.

Control de estructuras repetitivas

Teniendo en cuenta el peligro que un programa acabe ejecutando indefinidamente, forzosamente dentro de todo bucle debe haber instrucciones que manipulen variables el valor de las que permita controlar la repetición o el final del bucle. Estas variables se llaman **variables de control**.

Garantizar la asignación correcta de valores de las variables de control de una estructura repetitiva es extremadamente importante.

Cuando genere un programa, es imprescindible que el código permita que, en algún momento, la variable cambie de valor, por lo que la condición lógica se deje de cumplir. Si esto no es así, tendrá un **bucle infinito.**

Control de estructuras repetitivas

Normalmente, las variables de control dentro de un bucle se pueden englobar dentro de alguno de estos tipos de comportamiento:

- **Contador**: una variable de tipo entero que va aumentando o disminuyendo, indicando de manera clara el número de iteraciones que habrá que hacer.

Sintaxis y ejemplo,

Inicialización:

```
contador = valor inicial
```

Incremento/decremento:

```
contador = contador + valor_constante
contador = contador - valor_constante
```

```
vueltas = vueltas + 1;
goles = goles + 1;
final = final - 1;
numero = numero + 5;
```

Control de estructuras repetitivas por contador

Ciclo controlado por **contador**

```
public class Contador{
    public static void main(String[] args) {
        int i= 0;
        while (i < 5) {
            i= i+ 1;
            System.out.println(i);
        }
    }
}
```

El ejemplo usa una variable i para contar las repeticiones y se imprime por la consola el número de cada repetición: 1,2,3,4,5.

Control de estructuras repetitivas

- **Acumulador**: Una versión ampliada de contadores son los acumuladores. Un acumulador es una variable en la que se van acumulando directamente los cálculos que se quieren hacer, de manera que al alcanzar cierto valor se considera que ya no es necesario hacer más iteraciones.

Sintaxis y ejemplo,

Inicialización:

acumulador = valor inicial

Acumulación:

acumulador = acumulador + variable

acumulador = acumulador * variable

acumulador = acumulador - variable

acumulador = acumulador / variable

```
saldo = saldo + deposito;
saldo = saldo - retiro;
suma = suma + numero;
```

Control de estructuras repetitivas

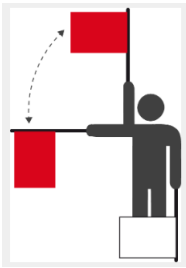
Nota: el uso de contadores y acumuladores no es excluyente, sino que puede ser complementario.

```
import java.util.Scanner;

public class Acumulador {
    public static void main(String[] args) {
        Scanner teclado=new Scanner(System.in);
        int x,suma,valor,promedio;
        x=1;
        suma=0;

        while (x<=10) {
            System.out.print("Ingrese un valor:");
            valor=teclado.nextInt();
            suma=suma+valor;
            x=x+1;
        }

        promedio=suma/10;
        System.out.print("La suma de los 10 valores es: ");
        System.out.println(suma);

        System.out.print("El promedio es: ");
        System.out.print(promedio);
    }
}
```

El ejemplo suma 10 valores y la reporta junto con la media de los mismos

Control de estructuras repetitivas

- **Bandera –flag-**: una variable que sirve como interruptor explícito de si hay que seguir haciendo iteraciones. Cuando ya no queremos hacer más, el código simplemente se encarga de asignarle el valor específico que servirá para que la condición evalúe false. Ejemplo,



Control de estructuras repetitivas

Ciclo controlado por **bandera**

```
public class Bandera{
    public static void main(String[] args) {
        boolean bandera = true;
        int i=1;
        while (bandera) {
            System.out.println(i);
            i= i+ 1;
            if (i==11)
                bandera = false;
        }
    }
}
```

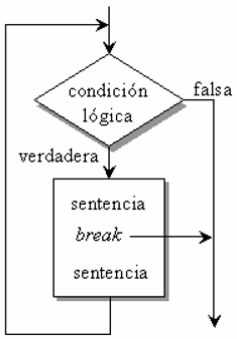
Sentencias break y continue

En cualquier punto de una estructura repetitiva se puede poner uno o varios puntos de ruptura de la ejecución, para lo cual se cuenta con las sentencias **break** y **continue**.

- La sentencia **break** interrumpe el bucle. Detener un bucle significa salirse de él y dejarlo todo como está para continuar con el flujo del programa inmediatamente después del bucle.
- La sentencia **continue** transfiere el control a la iteración siguiente del bucle. Sirve para volver al principio del bucle en cualquier momento, sin ejecutar las líneas que haya por debajo de la palabra continue.

Nota: vimos con anterioridad que el break también se utiliza para terminar una secuencia en una instrucción switch.

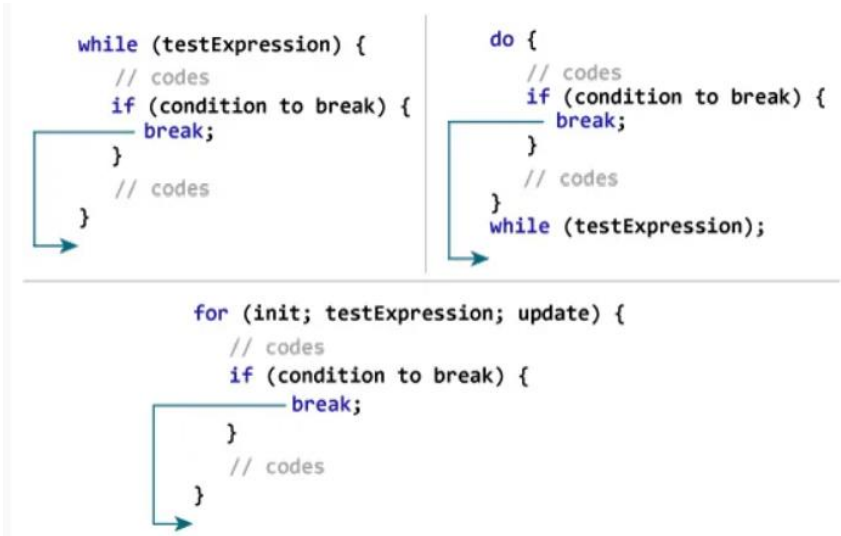
Sentencias break y continue



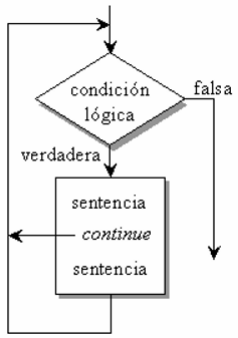

```
import java.util.Scanner;
public class EjemploBreak {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);
        String escribe;

        for (int i=0;i<10;i++){
            System.out.println(i) ;
            System.out.println("dime si continuo preguntando...si/no");
            escribe = sc.next();
            if (escribe.equals("no"))
                break ;
        }
    }
}
```

Actividad

```
for (int i = 0; i < 10; i++) {
  System.out.println("Dentro del bucle");
  break;
  System.out.println("Nunca lo escribira");
}
System.out.println("Tras el bucle");
```



Sentencias break y continue




```
import java.util.Scanner;
public class EjemploContinue{

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);
        String incrementar;

        int i=0;
        while (i<7){
            System.out.println("La cuenta está en " + i + ", dime si incremento. si/no");
            incrementar = sc.next();
            if (incrementar.equals("no"))
                continue;
            i++;
        }
    }
}
```

Actividad

```
for (int i = 0; i < 10; i++) {

  System.out.println("Dentro del bucle");

  continue;

  System.out.println("Nunca lo escribira");

}
```

Actividad

```java
public class EjemploBreakYContinue {

	public static void main(String[] args) {

		System.out.println("Primer bucle");
		for(int i = 0; i < 100; i++) {
			if(i == 74)
				break;
			if(i % 9 != 0)
				continue;
			System.out.println(i);
		}

		System.out.println("Segundo bucle");
		int j, i = 0;
		while(true) {
			i++;
			j = i * 27;
			if(j == 1269)
				break;
			if(i % 10 != 0)
				continue;
			System.out.println(i);
		}
	}
}
```

```java
public class EjemploBreakYContinue {

	public static void main(String[] args) {

		/* Se inicia un ciclo for con cien iteraciones empleando la variable i,
		 * cada iteración es impreso el valor la variable a pantalla,
		 * salvo los siguientes casos:
		 * - Si el valor de la variable i equivale a 74 se termina todo el ciclo debido al break.
		 * - Si la variable i no es múltiplo de 9 se interrumpe la iteración actual debido al continue.
		 */

		System.out.println("Primer bucle");
		for(int i = 0; i < 100; i++) {
			if(i == 74)
				break;
			if(i % 9 != 0)
				continue;
			System.out.println(i);
		}

		/* Antes de iniciarse un ciclo while se reinicializa la variable i a cero.
		 * Dentro del ciclo se imprimen los valores de la variable i en cada iteración,
		 * excepto en los siguientes casos:
		 * - Si el valor de la variable j equivale a 1269 se termina todo el ciclo debido al break.
		 * - Si la variable i no es múltiplo de 10 se interrumpe la iteración actual debido al continue.
		 */

		System.out.println("Segundo bucle");
		int j, i = 0;
		while(true) {
			i++;
			j = i * 27;
			if(j == 1269)
				break;
			if(i % 10 != 0)
				continue;
			System.out.println(i);
		}
	}
}
```

Actividad

17. Desarrolla una aplicación que muestre los números del 1 al 20, excepto el 15.
18. Desarrolla una aplicación que muestre los números del 1 al 10, usando un bucle for que vaya del 1 al 20.

¿El uso de break y continue es una mala práctica de programación?

¿Somos malos programadores si utilizamos estas estructuras de salto? **Depende de lo que estés haciendo**

¿El uso de break y continue es una mala práctica de programación?

Úsalos con moderación. En líneas generales:

- Cuando se usan al comienzo de un bloque, cuando se realizan las primeras verificaciones, actúan como condiciones previas, por lo que es bueno. Si mejora la legibilidad del código, úsalas.
- Cuando se usan en el medio del bloque, con algo de código, actúan como trampas ocultas, por lo que es malo.

Los buenos programadores usan la solución más simple y limpia posible.

¿El uso de break y continue es una mala práctica de programación?

Tal vez el error esté en comparar las sentencias break y continue con la sentencia **GOTO** (una sentencia propia de los primeros lenguajes de programación como Basic)

El propósito de la instrucción GOTO es transferir el control a un punto determinado del código, donde debe continuar la ejecución.

```
goto etiqueta;

...
...
...

etiqueta: sentencia;
```

¿El uso de break y continue es una mala práctica de programación?

La instrucción GOTO ha sido menospreciada en los lenguajes de alto nivel, debido a la dificultad que presenta para poder seguir adecuadamente el flujo del programa.

Esto podía derivar en **código espagueti**

El código espagueti, es un nombre peyorativo utilizado para designar aquellos programas cuyo flujo de ejecución se asemeja a una caótica maraña de espaguetis entrelazados, convirtiéndolo en algo casi imposible de seguir.

Código espagueti en C++

```
//Mayor de 3 números
#include <iostream>
#include <stdio.h>
#include <cstdlib>
#include <conio.h>
using namespace std;
int main(){
     int c=0, x,y,z, fin=-1;
  p10:
     cin >>x >>y >>z;
     if(x==fin) goto p50;
     c=c+1;
     if(x>y) goto p30;
     if(y>z) goto p40;
```

Código espagueti en C++

```
p20:
     cout << "Mayor es : " <<z <<endl;
     goto p10;
p30:
     if(x<z)goto p20;
     cout << "Mayor es : " <<x <<endl;
     goto p10;
p40:
     cout << "Mayor es : " <<y <<endl;
     goto p10;
p50:
     cout << "contador : " <<c <<endl;
     system("pause");
     return(0);
}
```

Código estructurado en C++

```
//Mayor de 3 números
#include <iostream>

using namespace std;

int main(){
   int c=0, x,y,z, fin=-1;
   cin >>x;
   while(x!=fin){
      c=c+1;
      cin >>y >>z;
      if(x>y && x>z){
         cout << "Mayor es : " <<x <<endl;
      }
```

Código estructurado en C++

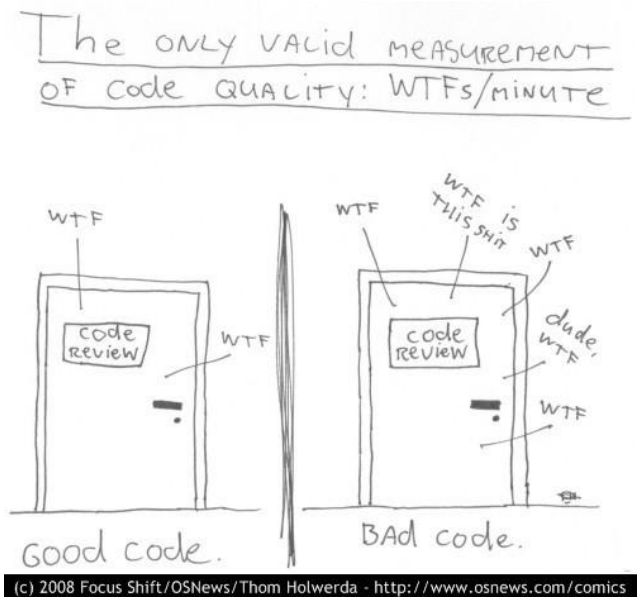
```
      else{
         if(y>x && y>z){
            cout << "Mayor es: " <<y <<endl;
         }
         else{
            cout << "Mayor es: " <<z <<endl;
         }
      }
      cin >>x;
   }
   cout << "contador : " <<c <<endl;
   system("pause");
   return(0);
}
```

¿El uso de break y continue es una mala práctica de programación?

Volviendo a las sentencias break y continue, estas no permiten transferir el control a cualquier parte del código como GOTO.

"Los malos programadores hablan en términos absolutos. Los buenos programadores utilizan la solución más clara posible.

El abuso en el uso de break y continue hace que el código sea difícil de seguir. Pero si el no usarlos hace que el código sea aún más difícil de leer, entonces es un mal cambio."



Recursos

- Curso youtube : Java desde 0
- Libro Java 9. Manual imprescindible. F. Javier Moldes Teo. Editorial Anaya
- App SoloLearn: Aprende a Programar. Curso Java