



# Programación 1

Iterativas en JAVA


# Sentencias iterativas

Un bucle o sentencia repetitiva se utiliza cuando se requiere hacer una o un conjunto de tareas varias veces

En una sentencia repetitiva hay una **expresión lógica** (condición simple o múltiple) que:

- si es cierta, ejecuta las sentencias entre llaves, y posteriormente vuelve a verificar la expresión lógica de terminación.
- si es falsa, sale del bucle.

```
Ciclo (expresión lógica) {  
    sentencia 1;  
    sentencia n;  
    ...  
}
```

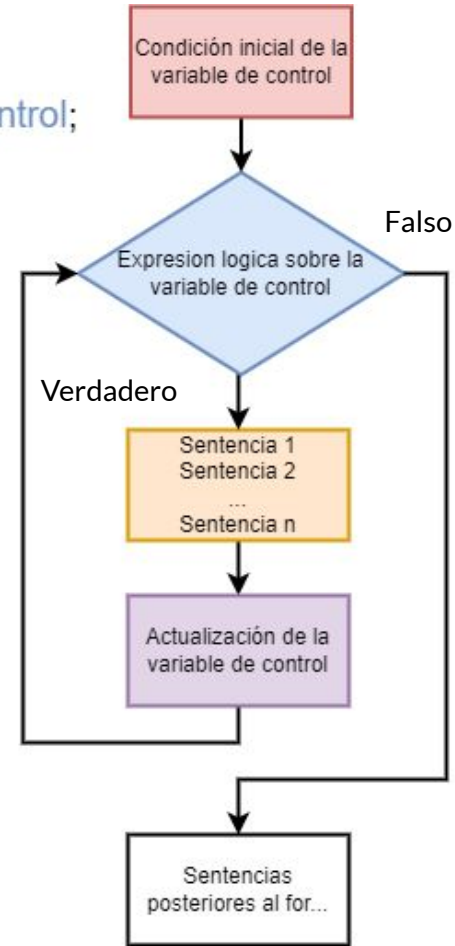


Las sentencias se ejecutan 1 o N veces mientras el valor de verdad de la **expresión lógica** sea **true**

# Sentencia for

```
for( condición inicial de variable de control; expresión lógica sobre variable de control;  
    actualización de la variable de control) {  
    sentencia_1;  
    sentencia_2;  
    ...  
    sentencia_n;  
}
```

La sentencia **for** SOLO se utiliza cuando se conoce exactamente la cantidad de iteraciones.



# Ejemplo

```
//Sentencia for
//Dado un número entero con valor inicial 5, imprimir la tabla de multiplicar de dicho número

public class Clase_2_Ejemplo_3 {
    public static void main(String args[]) {
        final int MAX = 10;
        final int MULTIPLO = 5;
        System.out.println("Tabla de multiplicar del" + MULTIPLO);
        // Se puede declarar o no la variable multiplicador dentro del inicio del for
        for (int multiplicador = 1; multiplicador <= MAX; multiplicador++) {
            System.out.println(MULTIPLO+" * "+multiplicador+" = "+(MULTIPLO * multiplicador));
        }
    }
}
```

# Ejemplo

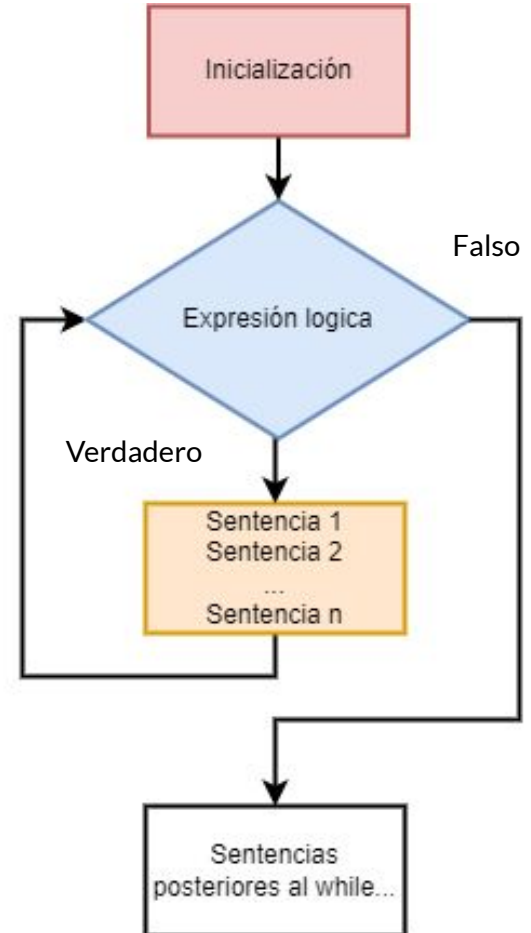
//Sentencia for anidada

```
public class Clase_2_Ejemplo_4 {  
    public static void main(String args[]) {  
        final int MAX = 10;  
        final int MULTIPLO = 3;  
        System.out.println("Tablas de multiplicar del 1, 2 y 3");  
        for (int i = 1; i <= MULTIPLO; i++) {  
            System.out.println("Tabla de multiplicar del " + i);  
            for (int j = 1; j <= MAX; j++) {  
                System.out.println(j + " * " + i + " = " + j * i);  
            }  
        }  
    }  
}
```

# Sentencia While

```
[inicialización]  
while( expresión lógica ) {  
    sentencia_1;  
    sentencia_2;  
    ...  
    sentencia_n;  
}
```

La sentencia while se utiliza cuando se conoce exactamente o **NO** la cantidad de iteraciones.



# Sentencia While



- Una sentencia iterativa se utiliza cuando se requiere realizar una o muchas tareas varias veces (varias veces puede ser ninguna, una o más).
- En la sentencia iterativa **while (expresión lógica){...}** hay una expresión lógica que:
  - si es **verdadera**, **ejecuta** las sentencias entre llaves, y luego vuelve al **inicio** de la sentencia iterativa para evaluar nuevamente la expresión lógica.
  - si es **falsa**, **sale** de la iteración.
- Mientras la **expresión lógica** sea verdadera va a ejecutarse lo que está dentro de las {...} en el orden en que aparecen.
- Algún valor de las **variables** de la expresión lógica deberá **cambiar** en algún ciclo de la repetición dentro de las sentencias en las {...}, sino se producirá un **ciclo infinito**.

# Ejemplo While

*/\*Dado un número entero con valor inicial 1, hacer una iteración que haga incrementar el número de a uno hasta un valor MAX = 4 (constante). Mientras itera deberá imprimir número \*/*

```
public class Clase_2_Ejemplo_7 {  
    public static void main (String [] args) {  
        final int MAX = 4;  
        int numero = 1;  
        while (numero <= MAX){  
            System.out.println("El numero es: " + numero);  
            //al cambiar de valor el número significa que el valor de la  
            //expresión lógica va a cambiar  
            numero++;  
        }  
    }  
}
```



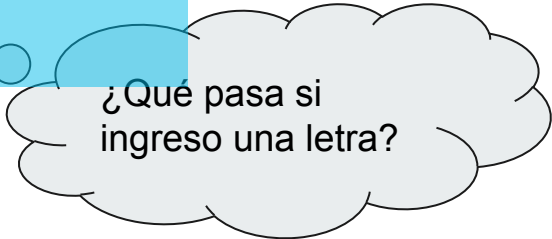
# A tener en cuenta

La utilización de sentencias iterativas implica **reubicar** sentencias declarativas como la de **BufferedReader**, para no iterar con una declaración dentro de las llaves {...} del ciclo y así evitar que en cada **ciclo** se **declare** la misma variable.

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
public class Clase_2_Ejemplo_8 {
    public static void main(String[] args) {
        //ubicar el buffer entrada cerca de la región de declaración de variables
        BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
        try{
            while (...){
            }
        }
        catch (Exception exc){
            System.out.println(exc);
        }
    }
}
```

# Ejemplo try-while

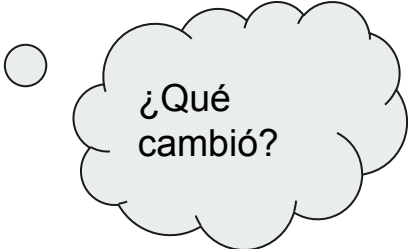
```
//Hacer un programa que mientras el usuario cargue un número entero distinto de 0 lo imprima
import java.io.BufferedReader;
import java.io.InputStreamReader;
public class Clase_2_Ejemplo_9{
    public static void main(String[] args) {
        int numero = 0;
        //la declaración del buffer entrada la ubico al principio junto con las otras declaraciones
        BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
        try{
            //el usuario carga un valor la primera vez
            System.out.println("Ingrese un numero entero (0 para salir): ");
            numero = Integer.valueOf(entrada.readLine());
            while (numero != 0){
                //si número es distinto de 0 lo imprime, vuelve a pedir su carga, y regresa al while
                System.out.println("El valor es: " + numero);
                System.out.println("Ingrese un numero entero (0 para salir): ");
                numero = Integer.valueOf(entrada.readLine());
            }
        }
        catch (Exception exc){
            System.out.println(exc);
        }
    }
}
```



¿Qué pasa si ingreso una letra?

# Ejemplo while-try

```
//Hacer un programa que mientras el usuario cargue un número entero distinto de 0 lo imprima
import java.io.BufferedReader;
import java.io.InputStreamReader;
public class Clase_2_Ejemplo_10{
    public static void main(String[] args) {
        int numero = 9;
        //la declaración del buffer entrada la ubico al principio junto con las otras declaraciones
        BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
        while (numero != 0) {
            try {
                System.out.println("Ingrese un número entero (0 para salir): ");
                numero = Integer.valueOf(entrada.readLine());
                if (numero != 0)
                    System.out.println("El valor es: " + numero);
            }
            catch (Exception exc) {
                System.out.println(exc);
            }
        }
    }
}
```



¿Qué cambió?

¿Cómo podemos hacer si queremos lo imprima solo si es par?

## Ejemplo capital acumulado



Para este problema deberá utilizar todos los tipos de sentencias desarrolladas hasta el momento.

Utilizando una sentencia repetitiva: calcular el capital acumulado a 10 años para un capital inicial de \$100 y una tasa de interés de 4% anual.

# Ejemplo capital acumulado

```
public class Clase_2_Ejemplo_11 {  
    public static void main(String args[]) {  
        final int MAX = 10;  
        final int INTERES = 4;  
        double capital = 100.0;  
        int anios = 1;  
        while (anios <= MAX) {  
            capital += capital*INTERES/100;  
            anios++;  
        }  
        System.out.println("Capital final es = " + capital);  
    }  
}
```

## Ejemplo valor positivo válido



Para este problema deberá utilizar todos los tipos de sentencias desarrolladas hasta el momento.

Ingresar un valor positivo **válido** y mostrarlo.

# Ejemplo valor positivo válido

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
public class Clase_2_Ejemplo_12 {
    public static void main(String args[]) {
        int valor = 0;
        BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
        while (valor <= 0) { // los valores positivos no incluyen al 0
            try { // try define un bloque de manejo de posibles excepciones
                System.out.println("Ingrese valor positivo: ");
                valor = Integer.valueOf(entrada.readLine());
                System.out.println("El valor ingresado es: " + valor);
            } catch (Exception exc) {
                System.out.println(exc);
            }
        }
    }
}
```

## Ejemplo valor válido



Para este problema deberá utilizar todos los tipos de sentencias desarrolladas hasta el momento.

Ingresar un valor entero **válido** y mostrarlo.



# Ejemplo valor válido

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
public class Clase_2_Ejemplo_13 {
    public static void main(String args[]){
        int valor = 0;
        boolean esValido = false;
        BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
        while (!esValido) { // mientras no haya ingresado un nro válido
            try { // try define un bloque de manejo de posibles excepciones
                System.out.println("Ingrese un valor entero: ");
                valor = Integer.valueOf(entrada.readLine());
                System.out.println("El valor ingresado es: " + valor);
                esValido = true;
            } catch (Exception exc) {
                System.out.println("El valor ingresado no es válido");
            }
        }
    }
}
```

## Ejemplo de problema



Para este problema deberá utilizar todos los tipos de sentencias desarrolladas hasta el momento.

Hacer un programa que dado un valor ingresado por el usuario entre 1 y 3 inclusive (**si ingresa otro valor termina**), imprima como salida “Bajo” en el caso de que ingrese 1, “Medio” si ingresa 2, y “Alto” si ingresa 3.

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
public class Clase_2_Ejemplo_11 {
    public static void main(String[] args) {
        final int MINIMO = 1;
        final int MAXIMO = 3;
        int valor = 1;
        BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
        while ((valor >= MINIMO) && (valor <= MAXIMO)) {
            try {
                System.out.println("Ingrese integer entre 1 y 3: ");
                valor = Integer.valueOf(entrada.readLine());
                switch (valor) {
                    case 1:
                        System.out.println("Bajo");
                        break;
                    case 2:
                        System.out.println("Medio");
                        break;
                    case 3:
                        System.out.println("Alto");
                        break;
                }
            }
            catch (Exception e) {
                System.out.println(e);
            }
        }
    }
}
```

Se puede escribir de otra forma  
esta expresión lógica?

`!((valor < MINIMO) || (valor > MAXIMO))`

# Algunos tips



- La sentencia while (expresión lógica) {...} se utiliza cuando se conoce exactamente o no la cantidad de iteraciones.
- La sentencia for ( ; ; ) {...} se utiliza cuando se conoce exactamente la cantidad de iteraciones.
- **No modificar la variable de control del for para forzar el corte.**
- **No forzar el corte con un break ni con return sea en for o while (aprender a armar bien la expresión lógica para que corte solo).**
- Por ejemplo cuando la expresión lógica tiene una variable que se carga dentro de las {...} o su valor es resultado de un cálculo (no de un incremento o decremento) usar while (expresión lógica) {...} ya que se desconoce cuántas iteraciones hará la sentencia.