INTRODUCCIÓN A LA PROGRAMACIÓN

Semana 3

ING. MARTIN POLIOTTO
Docente a cargo del modulo

Julio 2020

DIPLOMATURA EN
NUEVAS TECNOLOGÍAS

SECRETARÍA DE EXTENSIÓN UNIVERSITARIA SEU UTN Facultad Regional Córdoba
Ministerio de PROMOCIÓN DEL EMPLEO Y DE LA ECONOMÍA FAMILIAR
Ministerio de CIENCIA Y TECNOLOGÍA
CBA ENTRE TODOS
Con el apoyo de Oficina de Montevideo
Organización de las Naciones Unidas para la Educación, la Ciencia y la Cultura

Semana 03

1. Estructuras repetitivas

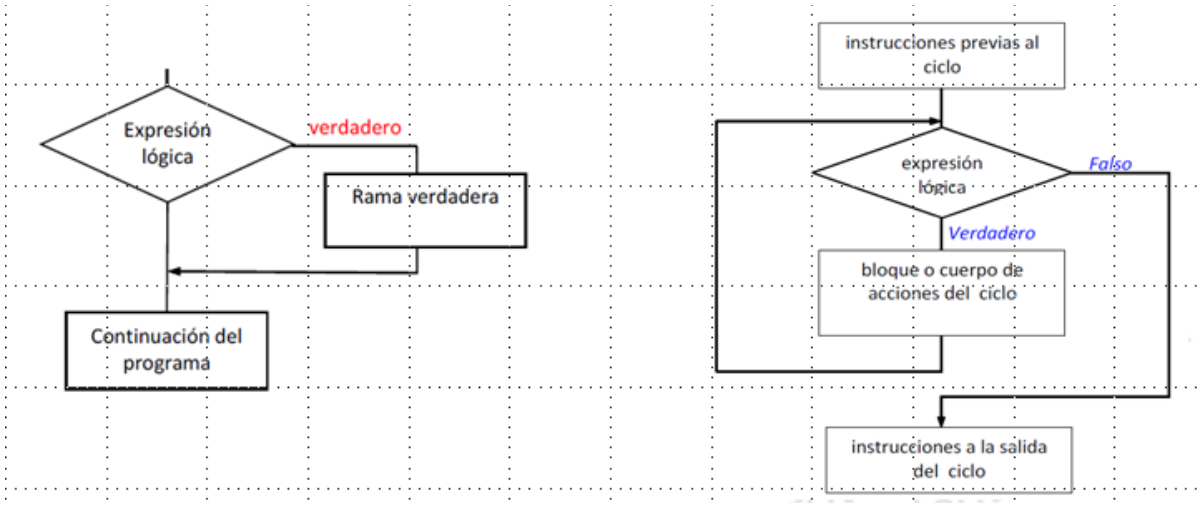
Las estructuras repetitivas permiten ejecutar un conjunto de sentencias repetidamente una cierta cantidad de veces o hasta que se cumpla una determinada condición. Se debe establecer un mecanismo para determinar las tareas repetitivas. Este mecanismo es una condición que puede ser verdadera o falsa y que se comprueba una vez a cada paso o iteración del bucle o ciclo.

Existen tres tipos de ciclos en Java:

- Ciclo **while:** se repite mientras una expresión lógica se evalúa como true.
- Ciclo **do-while:** se ejecuta una vez, y a continuación, se sigue repitiendo mientras la expresión es true.
- Ciclo **for:** se repite un número definido de veces,

1.1 Ciclo while

Un ciclo **while** o estructura **MIENTRAS**, puede pensarse como un condicional simple pero en vez de ejecutar solo un vez (en caso de ser verdadera) la rama verdadera se continuará ejecutando mientras la condición sea verdadera. El siguiente gráfico muestra esta comparación:



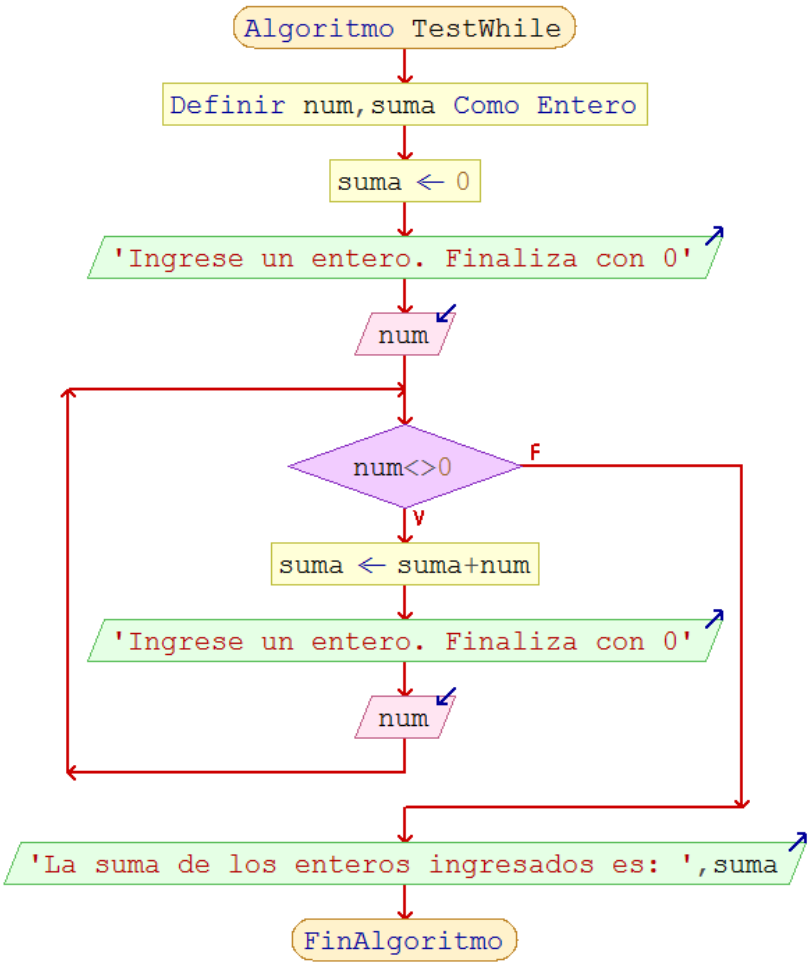
La sintaxis en Java es:

while(expresión lógica){

//bloque de acciones.

}

- Se lo conoce como **ciclo 0-N** porque puede no ejecutarse (si la condición es falsa inicialmente) o ejecutarse **N veces** mientras la condición es verdadera.
- Se utiliza cuando no conocemos de antemano cuántas veces se repite el bloque de acciones que forma el ciclo.
- Por ejemplo si queremos mostrar las suma de **n** enteros ingresados por teclado hasta que se ingresa un valor 0 (cero):



Que en código Java podemos escribir como:

```
public class TestWhile {

    public static void main(String[] args) {

        int num, suma;

        suma = 0;

        Scanner input = new Scanner(System.in);

        System.out.println("Ingrese un entero. Finaliza con 0");
```

```
        num = input.nextInt();
        while (num != 0) {
          suma += num;
          System.out.println("Ingrese un entero. Finaliza con 0");
          num = input.nextInt();
        }
        System.out.println("La suma de los enteros ingresados es: " + suma);
      }
}
```

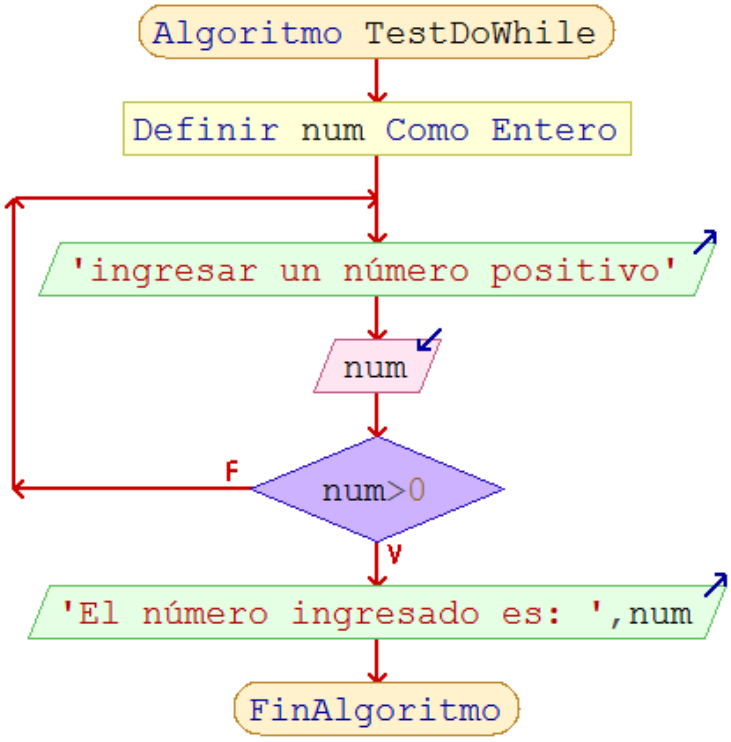
A tener en cuenta:

- Como se puede observar para el ingreso de los números fue necesario hacer una primera lectura fuera del ciclo y luego una segunda dentro de este. A este tipo de carga de datos se lo conoce como **ciclo de doble lectura.**
- Siempre es importante tener presente que la condición de corte debe afectarse dentro del bloque de sentencias del bloque, porque de otro modo, podemos estar en presencia de un ciclo infinito y nuestro programa no se ejecutaría como se espera.

1.2 Ciclo do-while

Una variante del ciclo anterior es el ciclo **HACER-MIENTRAS**. La idea es garantizar que el bloque de acciones que forman el ciclo se ejecute siempre la primera vez. Esto es posible porque la condición de corte se evalúa al final y no al comienzo como en la estructura **While.**

Supongamos que necesitamos un programa que permita leer y mostrar un número positivo. En caso de que el número ingresado sea menor a cero, deberá volver a pedirlo. Es decir que nuestro algoritmo deberá **pedir** un número **hasta que** sea positivo. El diagrama de flujo que modela esta situación es el siguiente:



Que en código Java podemos escribir como:

```
public class TestDoWhile {
    public static void main(String[] args) {
        int num;
        Scanner input = new Scanner(System.in);
        do {
            System.out.println("Ingrese un número positivo");
            num = input.nextInt();
```

```
    } while (num < 0);

    System.out.println("El número es: " + num);

  }

}
```

- Notar que en la línea resaltada la condición se plantea al revés que la que se modela en el diagrama de flujo. Esta diferencia es solo una cuestión semántica, en el diagramador la estructura es **Repetir-Hasta que**, en cambio en Java se programa **do (hacer) mientras que.**
- Se utiliza cuando no conocemos de antemano cuantas veces se repite, pero necesitamos que se ejecute al menos una vez.
- Se lo conoce como **ciclo 1-N** porque puede se ejecuta al menor 1 vez o bien puede ejecutarse **N veces** mientras la condición es verdadera.

1.2.1 Usos típicos del **do-while**

Típicamente esta estructura se utiliza para:

- Validar datos de entrada

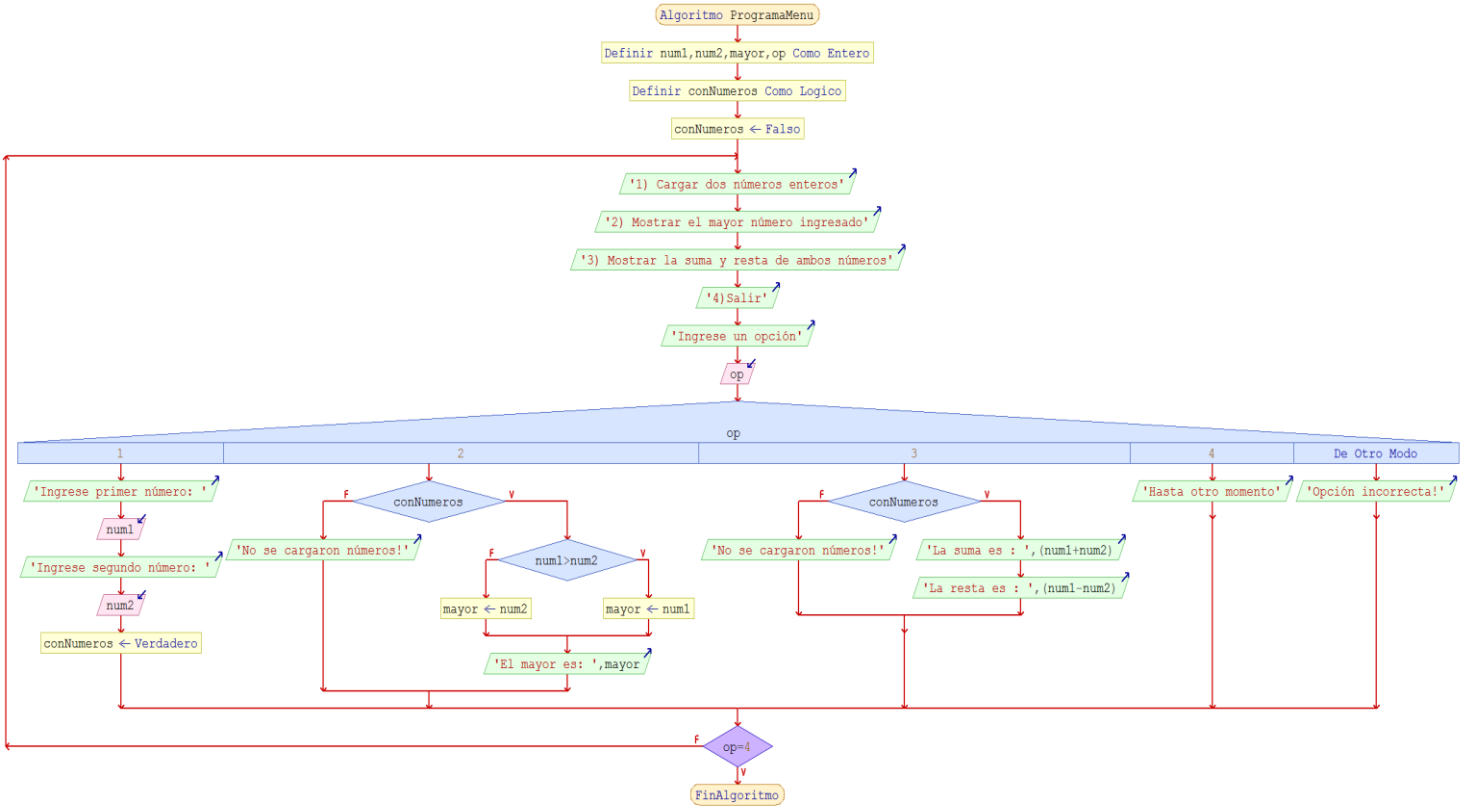
  El ejemplo anterior, es un caso de validación de datos de entrada. La idea es pedir un valor y mientras **sea incorrecto**, pedir nuevamente hasta que se ingrese un valor correcto.

- Armar un menú de opciones

  Supongamos que se necesita desarrollar un programa que muestre un menú de opciones para:

- 1) Cargar dos números enteros
- 2) Mostrar el mayor número ingresado
- 3) Mostrar la suma y resta de ambos números
- 4) Salir

En diagrama de flujo:



En Java:

```
public class ProgramaMenu {
    public static void main(String[] args) {
        int num1, num2, mayor, op;
        boolean conNumeros = false;
```

```
Scanner input = new Scanner(System.in);
num1 = num2 = 0;
do {
   System.out.println("1) Cargar dos números enteros");
   System.out.println("2) Mostrar el mayor número ingresado");
   System.out.println("3) Mostrar la suma y resta de ambos números");
   System.out.println("4) Salir");
   System.out.println("Ingrese una opción: ");
   op = input.nextInt();
   switch (op) {
      case 1:
         System.out.println("Ingrese primer número: ");
         num1 = input.nextInt();
         System.out.println("Ingrese segundo número: ");
         num2 = input.nextInt();
         conNumeros = true;
         break;
      case 2:
         if (conNumeros) {
            if (num1 > num2) {
               mayor = num1;
            } else {
               mayor = num2;
```

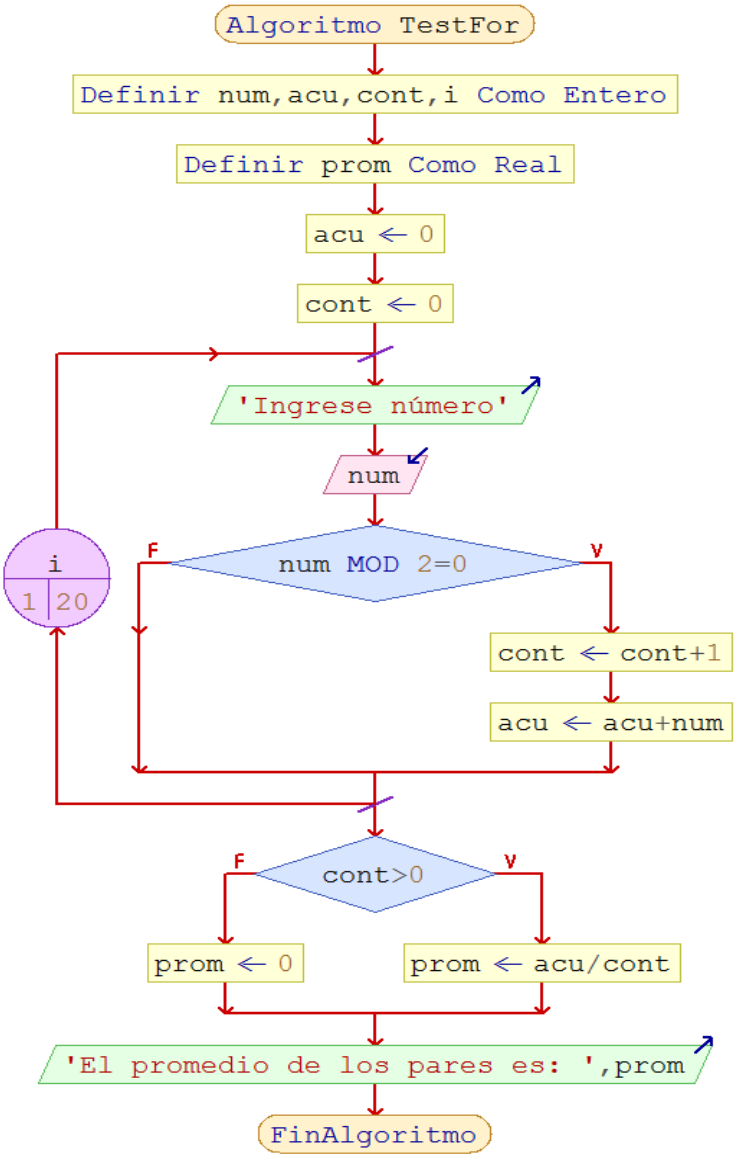
```
                }
                System.out.println("El mayor es: " + mayor);
            } else {
                System.out.println("No se cargaron números!");
            }
            break;
        case 3:
            if (conNumeros) {
                System.out.println("La suma es: " + (num1 + num2));
                System.out.println("La diferencia es: " + (num1 - num2));
            }else{
                System.out.println("No se cargaron números!");
            }
            break;
        case 4:
            System.out.println("Hasta otro momento!");
            break;
        default:
            System.out.println("Opción incorrecta!");
      }
    } while (op != 4);
  }
}
```

- La primera línea resaltada define una variable de tipo **boolean** que se utiliza para detectar si se ingresaron al menos 1 vez el par de números solicitados mediante la opción 1). Las variables utilizadas para indicar la ocurrencia de ciertos eventos dentro de un programa reciben el nombre de **variables centinela** o **banderas**.
- La segunda línea resaltada corresponde a la inicialización de las variables **num1** y **num2**. En Java las variables definidas dentro de los métodos necesitan ser inicializadas antes de usarse por primera vez.

1.3 Ciclo for

El ciclo for comúnmente se usa cuando se conoce de antemano la cantidad exacta de repeticiones que deben realizarse. A diferencia de los dos ciclos anteriores, sintácticamente presenta algunas diferencias que se verán a continuación.

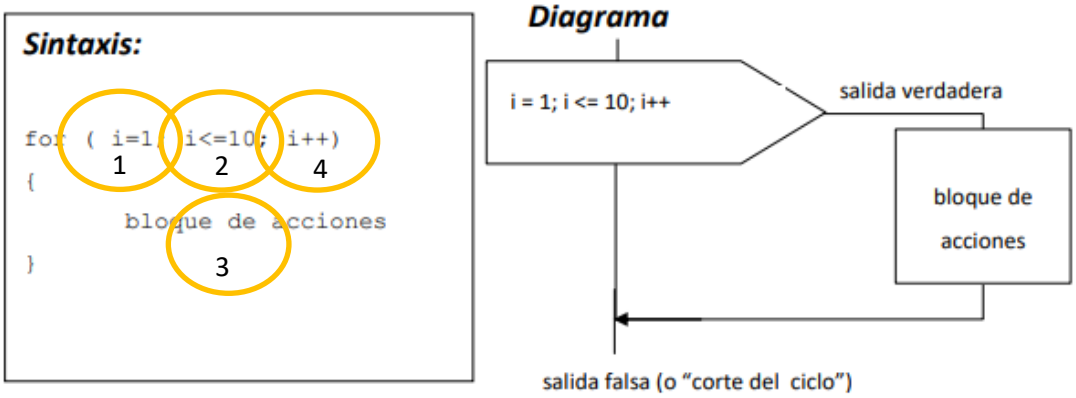
Supongamos que se quieren ingresar 20 números enteros y luego mostrar el promedio pero solo de los valores pares ingresados por teclado. El diagrama flujo puede representarse como:



En Java:

```
public class TestFor {
   public static void main(String[] args) {
      int num, acu, cont;
      float prom;
      Scanner input = new Scanner(System.in);
      cont = acu = 0;
      for (int i = 0; i < 20; i++) {
         System.out.println("Ingrese número:");
         num = input.nextInt();
          if(num % 2 == 0){
            cont++;
            acu = acu + num;
          }
      }
     if(cont > 0){
         prom = acu / cont;
      }else{
         prom = 0;
      }
      System.out.println("El promedio de los pares es: " + prom);
   }
}
```

- La línea resaltada corresponde a la cabecera del ciclo. Ésta se compone de tres secciones:
    - **Inicialización**: indica el valor inicial de la/s variables de control del ciclo.
    - **Condición de corte**: permite controlar la ejecución del ciclo. Mientras sea verdadera el ciclo se ejecuta. Caso contrario
    - **Incremento**: en esta sección se indica la forma en que cambiará el valor de cada variable de control del ciclo.
- Otra forma de representarlo sería:



Donde el orden ejecución es el que se indica en los círculos. Primero se inicializan las variables de control, luego se valida la condición de corte. Si es verdadera se pasa a ejecutar el ciclo y por último se incrementan las variables. Luego se controla nuevamente la condición de corte y se continúa con la ejecución del bloque.

- Ciclo **N**: si la condición de corte está exclusivamente vinculada a validar que la variable de control llegue a un valor límite, el ciclo se ejecuta **exactamente N veces**.

- En el ejemplo anterior se utilizaron dos tipos de variables que con frecuencia están presentes dentro de los ciclos: **contadores y acumuladores**.
    - Contador: cont = cont + 1 // también cont++
    - Acumulador: acu = acu + num // también acu+=num;

      En general los contadores permiten acumular un valor constante (k) mientras que los acumuladores lo hacen para el valor de una variable (var).

1.3.1 Casos especiales

1)
```
int c = 0;
for ( ; c != 20; c++) ----------;
```

2)
```
int a;
for (a = 1; a <= 100; a += 2) ----------;
```

3)
```
for (a = 0;  a < 10 && b != 20; a++)
{
      --------;
      b = miEscaner.nextInt();
}
```

4)
```
int i;
for (i = 1; i<=10; i++);
```

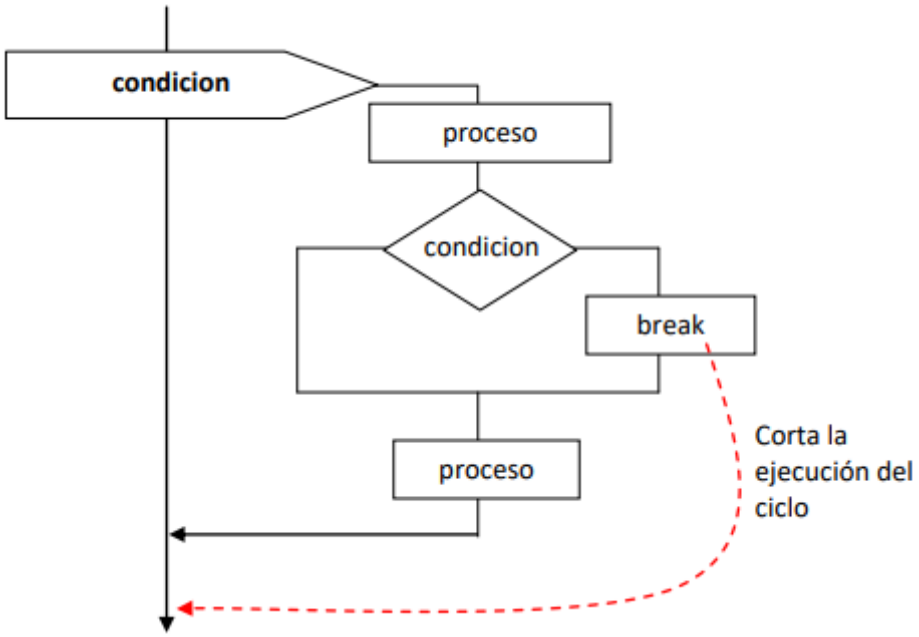
1) En el primer caso la sección de inicialización queda declarada fuera del ciclo.
2) En este caso el incremento de la variable de control es 2, con lo cual la variable asumirá los valores: 1, 3, 5, .., 99.

3) Notar que en la condición de corte la condición es compuesta.
4) En éste caso, el ciclo no tiene bloque de acciones (observar el punto y coma inmediatamente después del paréntesis del cierre de la cabecera). Simplemente efectúa diez repeticiones.

1.4 Elementos de control

El bloque de acciones de un ciclo (while o for) en Java se puede incluir una instrucción **break** para cortar el ciclo de inmediato sin retornar a la cabecera para evaluar la expresión lógica de control.

En gráfico:

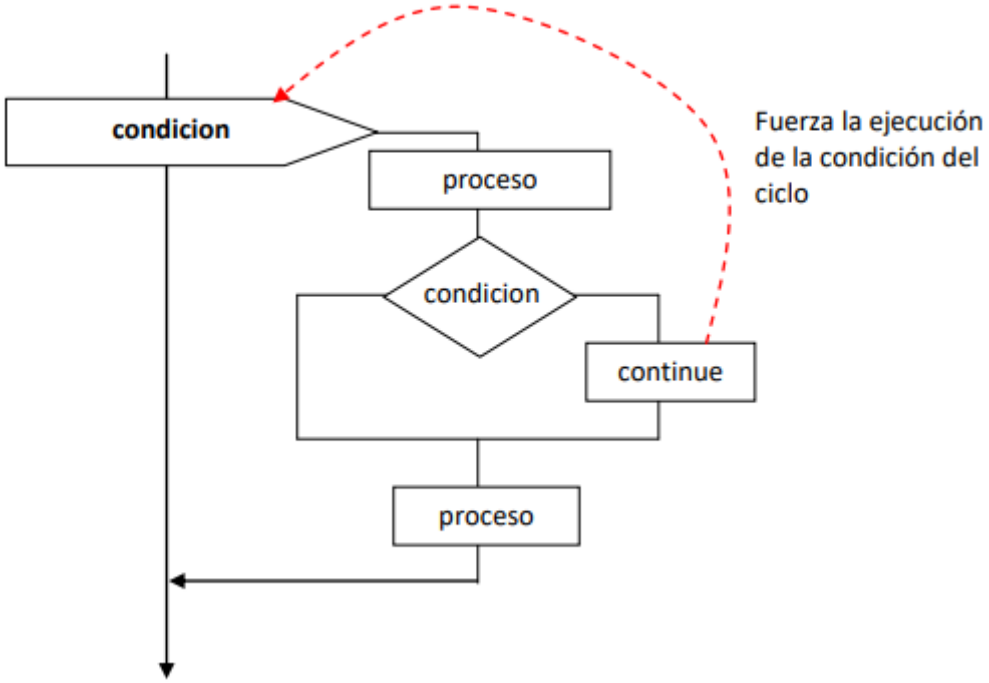


- La sentencia **break** interrumpe la estructura repetitiva inmediata superior
- Generalmente se asocia con alguna condición lógica, de modo que si dentro del bloque de acciones de un ciclo ocurre cierta situación, el ciclo se interrumpe y el resto de las instrucciones quedarán sin ejecutarse.

- En Java se indica como *break;*

De forma similar, pero a la inversa, un ciclo cualquiera puede incluir una instrucción **continue** para forzar una repetición del ciclo sin terminar de ejecutar las instrucciones que queden por debajo de la invocación a **continue.**

En gráfico:



- La sentencia **continue** se utiliza generalmente para omitir pasos en función del valor de ciertas variables.
- Si bien, es una herramienta que ofrece el lenguaje no es una instrucción utilizada con frecuencia en el desarrollo de los algoritmos.
- En Java se indica como *continue;*