

Sintaxis en Java: Iteración

1. Sentencias de Repetición

1.1 Introducción

Las sentencias de repetición, nos van a permitir repetir un bloque de código varias veces, según alguna condición o una cierta cantidad fija de veces.

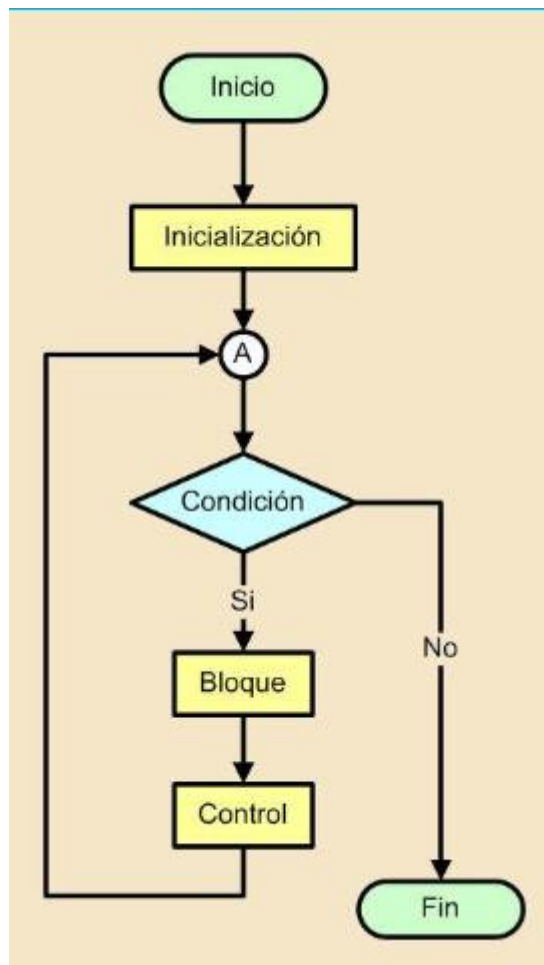
Tenemos entonces, dos sentencias posibles para repetir. La sentencia llamada “for” (que en pseudocódigo la utilizamos como “PARA”) y la sentencia “while” (“MIENTRAS”). La primera, nos brinda la posibilidad de repetir una cantidad N (fija) de veces un código, mientras que la segunda, nos permite repetir un cierto bloque de código mientras se cumpla determinada condición.

1.2 Sentencias FOR

En esta sección, veremos la sintaxis que tiene esta sentencia y cómo utilizarla. La idea de esta sentencia es la repetición en una cantidad N (fija) de veces, un bloque de código.

La sintaxis de esta sentencia es la siguiente:

```
for (variable_de_control ; condición_de_parada ; paso_de_la_iteración){ //  
Aquí van las sentencias que queremos repetir  
}
```



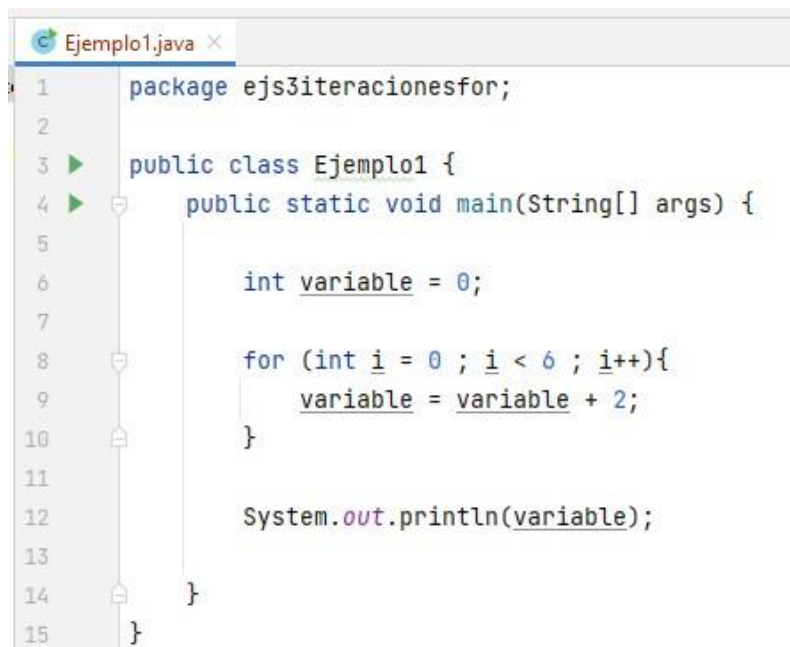
Donde:

- **variable_de_control** es la variable numérica que utilizaremos para ir repitiendo los bloques, esta irá cambiando su valor de acuerdo a el paso de iteración. El nombre de la variable puede ser cualquiera, pero suelen usarse letras como i, j, k.
- **condición_de_parada** es la condición booleana que se debe cumplir sobre la variable de control para que termine la repetición. Por cada valor que tome i, se debe evaluar esta condición, si esta evalúa en true, se debe ejecutar el bloque de código for, si evalúa en false no se ejecuta el bloque y se termina el for.

- **paso_de_la_iteración** es el paso con el que incremento o decremento la variable de control en cada repetición.
- Luego, lo que se encuentre entre las llaves, será el bloque de sentencias que se repetirán.

Veamos algunos ejemplos:

Ejemplo 1



```
Ejemplo1.java x
1 package ejs3iteracionesfor;
2
3 public class Ejemplo1 {
4     public static void main(String[] args) {
5
6         int variable = 0;
7
8         for (int i = 0 ; i < 6 ; i++){
9             variable = variable + 2;
10        }
11
12        System.out.println(variable);
13
14    }
15 }
```

Antes de la explicación de la “corrida a mano” del programa, cabe destacar que en el for es posible declarar y asignar valor inicial a la variable de control dentro de el mismo:

```
for (int i = 0 ; i < 6 ; i++){
```

Pero a esta variable únicamente la podremos utilizar dentro del bloque de for.

```
for (int i = 0 ; i < 6 ; i++){  
    //Bloque for.  
}  
  
System.out.println(i);
```

Vemos que al intentar utilizarla por fuera del bloque (fuera de su alcance), no compila el programa. Ahora, si declaramos esta variable fuera del for:

```
int i = 4;  
  
for ( i = 0 ; i < 6 ; i++){  
    //Bloque for.  
}  
  
System.out.println(i);
```

El programa compila, y podremos utilizar esta variable por fuera del bloque for. En este caso *i* antes del for toma el valor de 4, al ingresar al for se le asigna el valor de 0, y al terminar la ejecución del for y al imprimir la variable *i*, esta tomará el último valor que le asignó el for.

Volviendo al Ejemplo 1:

- En este ejemplo, la variable de control del for (llamada “*i*” en este caso) va iniciar en 0. Como vimos se puede realizar la declaración y asignación del valor inicial de la variable de control al mismo tiempo dentro de los paréntesis del for. Pero de esta forma sólo podremos utilizarla dentro del bloque del for.
- Luego en cada repetición, se ejecutará el paso de iteración, en este caso se incrementará en uno (esto es por el “*i++*” que aparece en el for).

¿Cuándo se saldrá del bloque? cuando la condición “*i*<6” sea falsa.

- Por lo tanto hagamos, la simulación de la ejecución de este for:

- En la primera repetición, tenemos que $i=0$, ¿ $0<6$? true.

- Por lo tanto, ejecuta el bloque del for.

- En este caso, tenemos sólo una suma, por lo tanto, a la variable “variable” le sumamos 2.

- Luego, termina el bloque del for, por lo tanto, debemos ejecutar el paso de iteración del for, en este caso: “ $i++$ ”, por lo tanto, ahora $i=1$.

- Ahora tenemos $i=1$, ¿ $1<6$? true.

- Por lo tanto, ejecutamos lo que tenemos dentro del for, que es la suma. Por lo tanto, sumamos nuevamente el valor 2 a la variable llamada “variable”. Y termina el bloque for, y ejecutamos el incremento “ $i++$ ”, lo que nos queda $i=2$.

- Ahora tenemos $i=2$, ¿ $2<6$? true.

- Entonces ejecutamos el bloque dentro del for nuevamente.

- Es claro que vamos a llegar a tener a la variable de control i , con valor $i=6$.

- En este caso se va a preguntar ¿ $6<6$?

- La respuesta será **false**, por lo que en ese momento no voy a entrar al bloque del for, o sea, se termina la ejecución del for y continuaré con la ejecución del programa, con las instrucciones que estén debajo.

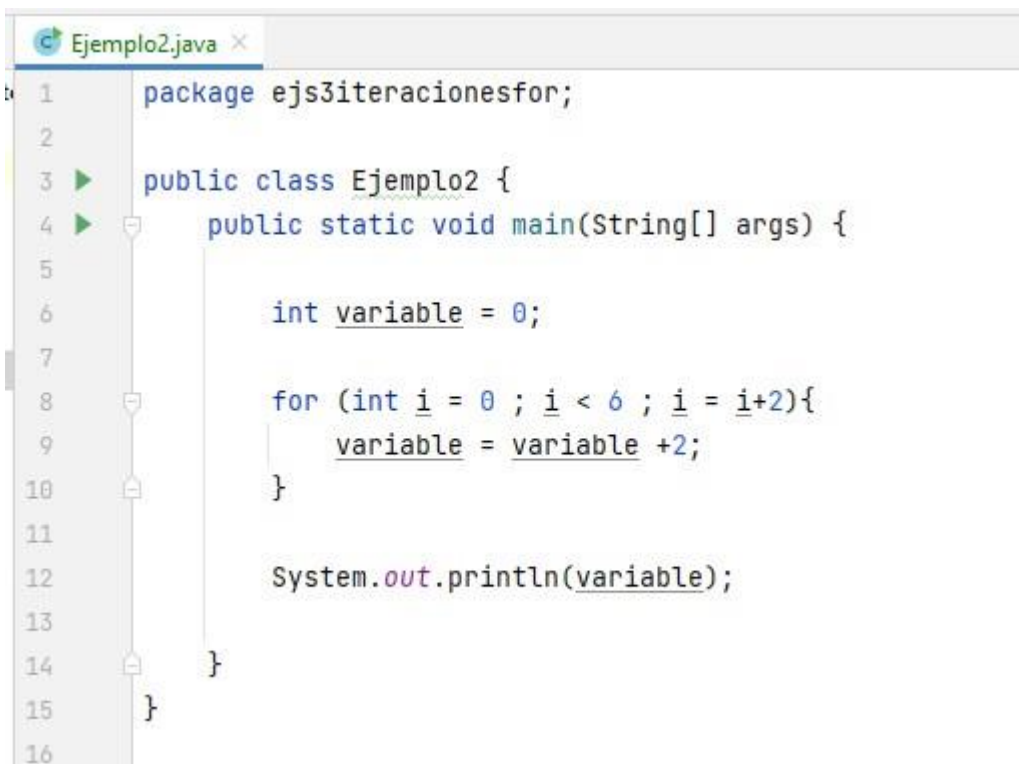
¿Cuántas veces se repite la sentencia que está dentro del for en el ejemplo?

La variable i tomará los siguientes valores: 0, 1, 2, 3, 4, 5.

Por lo tanto, se ejecutará 6 veces la sentencia “ $variable = variable + 2$ ”. Entonces, estoy sumándole 2 a una variable 6 veces, por lo tanto, le estoy sumando un total de 12, y si esta variable antes del for tiene como valor 0, luego de la repetición, variable toma el valor de 12.

Como paso de iteración podremos utilizar la sentencia que necesitemos, en el ejemplo 2 vemos que la variable de control, aumenta de a dos.

Ejemplo 2



```
1 package ej3iteracionesfor;
2
3 public class Ejemplo2 {
4     public static void main(String[] args) {
5
6         int variable = 0;
7
8         for (int i = 0 ; i < 6 ; i = i+2){
9             variable = variable +2;
10        }
11
12        System.out.println(variable);
13
14    }
15 }
16
```

- En este caso, tenemos que la variable de control del for, inicialmente, arranca en el valor 0, pero en cada iteración, en vez de sumarle uno y continuar, le sumamos **dos**.

- Por lo tanto, si tenemos que definir el flujo de ejecución para este for, sería el siguiente:

- o Inicialmente, tenemos que $i=0$. ¿ $0<6$?, true.

Pasamos a ejecutar las sentencias del bloque (en este caso la suma “ $variable = variable + 2$ ”). Luego que terminamos, incrementamos la variable de control, según el paso de iteración ($i=i+2$), por lo tanto, ahora tiene el valor 2.

o Ahora tenemos $i=2$, ¿ $2<6$?, true.

Por lo tanto, ejecutamos lo que tenemos dentro del for, que es la suma. Por lo tanto, sumamos nuevamente el valor 2 a la variable llamada “variable”. Y termina el bloque for, y ejecutamos el incremento ($i=i+2$), lo que nos queda $i=4$

- Y debemos continuar los pasos hasta que la condición evalúe como false.

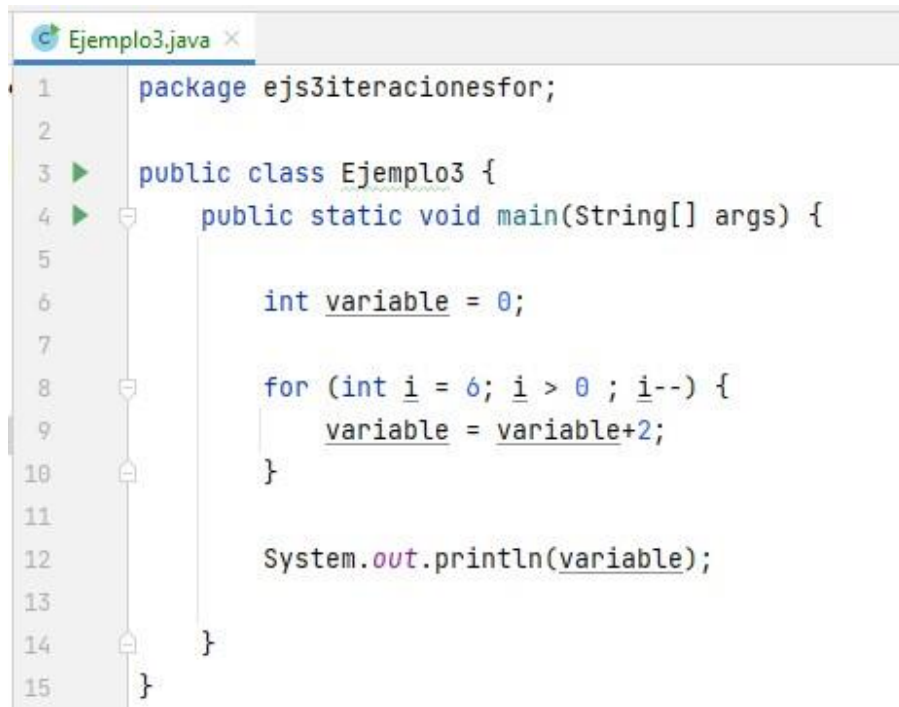
¿Cuántas veces se repite la sentencia que está dentro del for en el ejemplo?

Se ejecutará para los siguientes valores de i : 0, 2, 4. Por lo tanto se ejecutará 3 veces la sentencia “variable = variable + 2”. Entonces, estoy sumándole 2 a una variable 3 veces, por lo tanto, le estoy sumando un total de 6, y si esta variable estaba con valor 0, es claro que luego de la repetición, va a quedarnos con el valor 6.

Podemos también, cambiar el valor de inicio del for y/o ir de valores decrecientes (que el paso de iteración sea una resta).

Veamos algún ejemplo de estos casos:

Ejemplo 3



```
1 package ejs3iteracionesfor;
2
3 public class Ejemplo3 {
4     public static void main(String[] args) {
5
6         int variable = 0;
7
8         for (int i = 6; i > 0 ; i--) {
9             variable = variable+2;
10        }
11
12        System.out.println(variable);
13
14    }
15 }
```

- Podemos ver que, en este ejemplo, la variable de control comienza en 6 y va decreciendo en 1 (i-- equivale a $i = i - 1$). Se puede ver también que ahora la condición para que se siga el bloque, cambio, y vamos a terminar de repetir cuando la variable de control sea 0.

¿Cuántas veces repetiremos el bloque for? La respuesta es 6. Se ejecutará para los siguientes valores de i: 6,5,4,3,2,1.

Se puede ver, comparando con el Ejemplo 1, que el código hace exactamente lo mismo, únicamente que cambia la condición que define al for.

Dentro del bloque de for, podemos utilizar la variable de control. Esta, tendrá el valor que corresponda a la repetición en la que se encuentre.

Veamos un último ejemplo de esto:

Ejemplo 4

```
Ejemplo4.java x
1 package ejs3iteracionesfor;
2
3 public class Ejemplo4 {
4     public static void main(String[] args) {
5
6         int variable = 0;
7
8         for (int i = 0; i < 6; i++) {
9             variable = variable+i;
10            System.out.println("La variable de control i vale = "+i);
11        }
12
13        System.out.println(variable);
14
15    }
16 }
17
```

- En este ejemplo, dentro del bloque de repetición, tenemos dos sentencias. Una de asignación y una de print.
- Además, en ambas se utiliza la variable de control del for, que toma el valor que corresponda a la repetición en la que esté. De esta manera, si seguimos el flujo de ejecución del for, sería el siguiente:
 - o Comenzamos con $i=0$. ¿ $0<6$?, true.
Entramos en bloque de for, a la variable “variable” le sumamos el valor de la variable de control i , que vale 0.
Luego imprimimos el mensaje: La variable de control i vale = 0.
Ejecutamos el paso de iteración: le sumamos uno a la variable de control i (ahora $i=1$).
 - o Tenemos que $i=1$. ¿ $1<6$?, si.

Por lo tanto, ejecutamos el bloque del for. A la variable “variable” le sumamos el valor de la variable de control i (que en esta repetición vale 1).

Luego imprimimos el mensaje: La variable de control i vale = 1.

Ejecutamos el paso de iteración: le sumamos uno a la variable de control i (ahora vale $i=2$).

o Continuamos con la ejecución hasta $i < 6$ sea false. Esto se da, en este caso, cuando i tome el valor de 6.

- ¿Qué valor tiene la variable “variable” luego de la sentencia for?

En cada repetición le sumamos el valor de i . Podemos ver que la variable i va tomando los valores 0,1,2,3,4,5. Por lo tanto a la variable “variable” le terminamos sumando estos valores, o sea que queda con valor 15.

Cabe destacar, que podemos ir mezclando las diferentes sentencias y combinando sentencias condicionales con sentencias de repetición. De esta manera podemos tener un código como el siguiente:

```
Ejemplo5.java x
1 package ej3iteracionesfor;
2
3 public class Ejemplo5 {
4     public static void main(String[] args) {
5
6         int cantidadPar = 0;
7         int cantidadImpar = 0;
8
9         for (int i = 0; i < 10; i++) {
10             boolean esPar = i % 2 == 0;
11
12             if(esPar){
13                 cantidadPar++;
14             }else{
15                 cantidadImpar++;
16             }
17
18         }
19
20         System.out.println("La cantidad de índices pares que recorrimos fue: "+cantidadPar);
21         System.out.println("La cantidad de índices impares que recorrimos fue: "+cantidadImpar);
22
23     }
24 }
```

- En dicho código tenemos una repetición de tipo “for” que se ejecuta 10 veces. En cada iteración, chequea si el índice que se está procesando es par o impar. Dependiendo de esto acumula en una u otra variable para i contando.

Para chequear si el valor de i es par o impar se está utilizando la operación modulo (%), que devuelve el resto de la división entera. En este caso que es $i \% 2$ (i módulo 2), el resultado es el resto de la división del valor de i dividido 2, por ejemplo: $4 \% 2$ da 0 (el resto de la división entera es 0), $7 \% 2$ da 1 (el resto de la división entera es 1).

- ¿Que se imprimirá en la consola para este ejemplo?

La variable de control toma los valores desde 0 al 9. Luego dentro de estos tenemos 5 pares (0,2,4,6,8), por lo tanto, tendremos en la cuenta 5 pares. Luego el mismo razonamiento para los impares (1,3,5,7,9). Tendremos también 5, por lo tanto, se imprimirá lo siguiente:

La cantidad de índices pares que recorrimos fue: 5

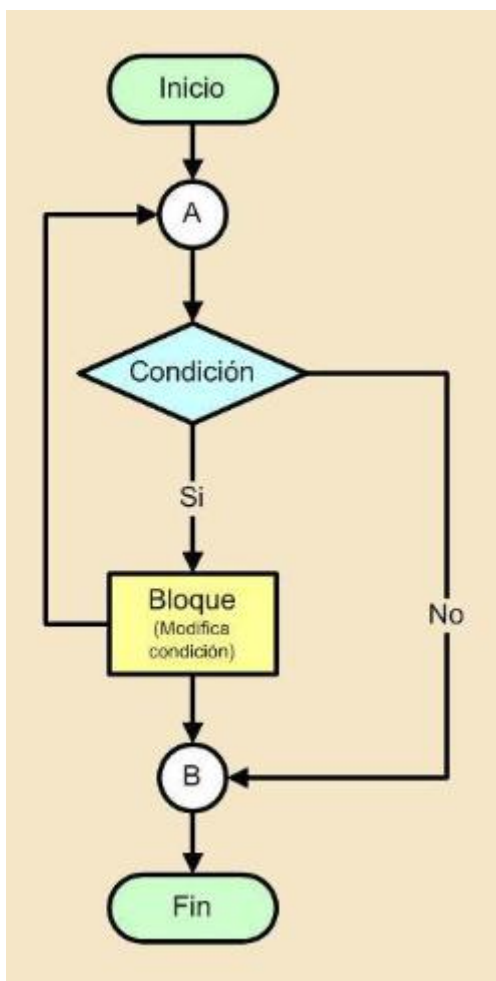
La cantidad de índices impares que recorrimos fue: 5

1.3 Sentencia WHILE

En esta sección, veremos la sintaxis que tiene esta sentencia y cómo utilizarla. Utilizaremos esta sentencia cuando queramos ejecutar un bloque de código “mientras” se cumpla una cierta condición dada.

La sintaxis de esta sentencia, se muestra a continuación:

```
while(condición_booleana){  
//instrucciones que se ejecutan mientras condición_booleana sea true  
}
```



Donde:

- **condición_booleana** es una expresión que devuelve un booleano o variable de tipo booleano, que indica si debo ejecutar el bloque while. Mientras esta condición sea true, ejecuto el bloque, cuando esta condición sea false, no entro a ejecutar el bloque y sigo corriendo el resto del programa.
- Para poder parar la repetición del bloque de mientras, se debe modificar algún componente de la condición para hacer que evalúe en false y poder parar su ejecución. ● Dentro de las llaves, irá el código que queremos que se repita.

Veamos algunos ejemplos:

Ejemplo 1

```
Ejemplo1.java x
1 package ejs3iteracioneswhile;
2
3 public class Ejemplo1 {
4     public static void main(String[] args) {
5
6         String texto = "aaaaaaaaaaaaa";
7         String nuevoTexto = "";
8
9         while (!nuevoTexto.equals(texto)){
10             nuevoTexto = nuevoTexto + "a";
11             System.out.println(nuevoTexto);
12         }
13
14         System.out.println("Acá finaliza el programa de prueba de letra a");
15
16     }
17 }
```

- En el ejemplo, tenemos dos variables iniciales, una que tiene muchas letras a, y otra tira (string) vacía.
- Nuestro bloque de while se va a ejecutar, mientras que ambos textos sean distintos y en cada iteración, ampliamos la tira "nuevoTexto", concatenándole una "a" al final. Tiene lógica pensar que, en algún momento, llegaré a tener la cantidad de letras "a" que el texto original, y mi repetición terminará.
- Veamos cómo sería el flujo de ejecución del ejemplo anterior:

o Cuando llegamos a la sentencia while, se evalúa la condición. En este caso, esta nos da true, ya que compara la variable texto con la variable nuevoTexto (o sea compara el texto vacío ("") con "aaaaaaaaaaaaa" y son distintos).

Por lo tanto, la evaluación de condición da true: procedemos a ejecutar el bloque del while (**ciclo 1**). En este caso, tenemos únicamente una sentencia de concatenación (+) de Strings.

Procedemos a ejecutarla.

Luego termina el bloque while.

o Ahora volvemos de nuevo a evaluar la condición del while, esta vez, nuevamente me da true, pero ahora la variable nuevoTexto ya no es vacía (""), sino ahora es el texto "a" (por la primera ejecución del bloque while). "a" es distinto de "aaaaaaaaaaaa", entonces entro en **ciclo 2**.

o Ejecutamos nuevamente el bloque while, y por ende le concatenamos una nueva letra "a" a nuestra tira, que ahora queda con el valor "aa". Y volvemos a evaluar la condición del while. Nuevamente nos da true, ya que "aa" es distinto de "aaaaaaaaaaaa") y ejecutamos el bloque while.

o Si repetimos este proceso varias veces, veremos que, el texto de la variable nuevoTexto y el de la variable texto ambos tendrán el valor "aaaaaaaaaaaa". En este momento se termina la ejecución de la sentencia "while" y se continúa con la siguiente a continuación. En este ejemplo, la siguiente sentencias es una sentencia de Print.

Veamos un nuevo ejemplo, mostrando cómo podemos usar un booleano en vez de una expresión como en el ejemplo anterior. En este ejemplo, ya podemos ver también que podemos combinar nuevamente, sentencias "while" y "if" por ejemplo:

Ejemplo 2

```
Ejemplo2.java x
1 package ejs3iteracioneswhile;
2
3 public class Ejemplo2 {
4     public static void main(String[] args) {
5
6         int i = 56;
7         boolean esIgual = false;
8
9         while(!esIgual){
10
11             if(i == 71){
12                 esIgual = true;
13             }
14             i++;
15         }
16
17         if(i == 71){
18             System.out.println("La variable i vale 71");
19         }else{
20             System.out.println("La variable i vale 72");
21         }
22     }
23 }
24 }
```

- En este caso, tenemos un programa, que dado un entero i que comienza con el valor 56, se irá iterando hasta que tenga el valor 71. Cuando tenga dicho valor, la condición del while va a evaluar en false y no se ejecutará más el bloque while. Luego de terminado el while pasaremos a ejecutar lo que sigue, que este caso es una sentencia IF-ELSE.
- ¿Que se va a imprimir en este caso? La respuesta es:

La variable i vale 72.

Ya que cuando i tiene valor 71, se entra al if. Por lo tanto setea "esIgual" a true y le suma 1 a i.

Ahora *i* tiene valor 72, y al ser esIgual true la evaluación de (!esIgual) es false, entonces no se entra al while y continúo con la ejecución del código posterior (print).

1.3.1 sentencia break

Cuando vimos la sentencia switch, vimos que se utilizaba la instrucción break para “salir del switch”, esta se puede utilizar dentro de cualquier bloque de código de repetición también:

Ejemplo 1

```
for (int i = 0; i < 7; i++) {  
    System.out.println(i);  
  
    if(i == 2){  
        break;  
    }  
}
```

Aquí tenemos un for donde la variable *i* toma los valores: 0,1,2,3,4,5,6 pero dentro del bloque for hay un condicional: si *i* es 2 entra al bloque if y se ejecuta la sentencia break que me hace salir del bloque de la repetición.

Por lo tanto, este for se va a ejecutar únicamente para *i*: 0,1,2.

Cuando *i* sea 2, se entra al bloque if, se ejecuta break y salgo del for.

Lo mismo sucede para la repetición con while.

Ejemplo 2

En este ejemplo tenemos un while (true), o sea que la condición siempre va a evaluar como true y quedaría en loop infinito. Pero veamos como al utilizar el break podemos salir de este loop:

```
while(true){  
    i++;  
    if(i == 2){  
        break;  
    }  
}
```

Dentro del bloque while se aumenta en 1 a la variable i, por lo tanto cuando i tome el valor de 2, voy a entrar al bloque de if (ya que la condición `i == 2` es true) y se va a ejecutar el break (que me hace salir del bloque while, sin importar la evaluación de la condición).

La sentencia break se puede utilizar dentro de bloques de repetición (loop) o switch. En general dentro de la repetición no se utilizan ya que puede resultar más complicado seguir el flujo del programa.

1.3.2 sentencia continue

La sentencia continue, permite no ejecutar todas o parte de las sentencias de un bucle (for, while, do while) , en una o varias de sus iteraciones. En este sentido se diferencia de la sentencia break, que en vez de interrumpir completamente la ejecución de un bucle, y saltar a la siguiente sentencia. Vamos a ver un ejemplo de listar todos los números pares del 1 al 20. Al encontrar la sentencia continue se salta a la próxima iteración.

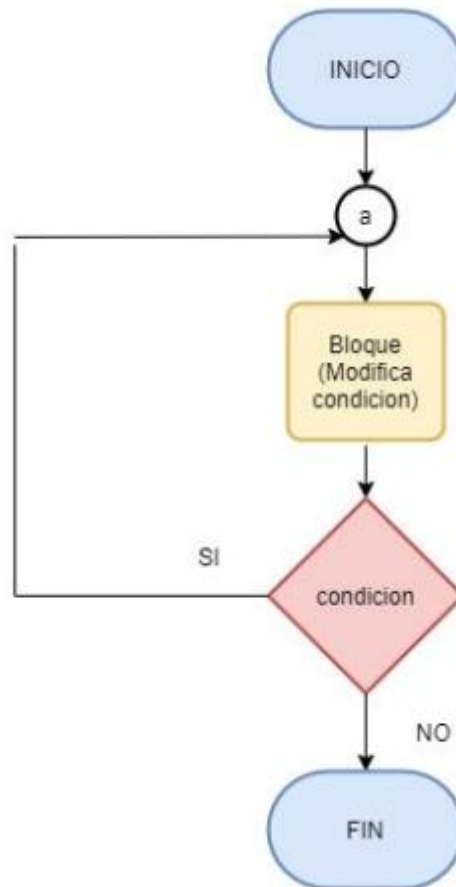
Ejemplo:

```
public class Iteracion {  
    // Ejemplo de sentencia continue  
    public static void main(String[] args) {  
        for (int i=1;i<20;i++) {  
            if(i%2 !=0){  
                continue;  
            }  
            System.out.print(i+" - ");  
        }  
    }  
}
```

Y la salida por consola es :

```
"C:\Program Files\Java\jdk-17.0.2\bin\jav  
2 - 4 - 6 - 8 - 10 - 12 - 14 - 16 - 18 -  
Process finished with exit code 0
```

1.4 Sentencia DO WHILE



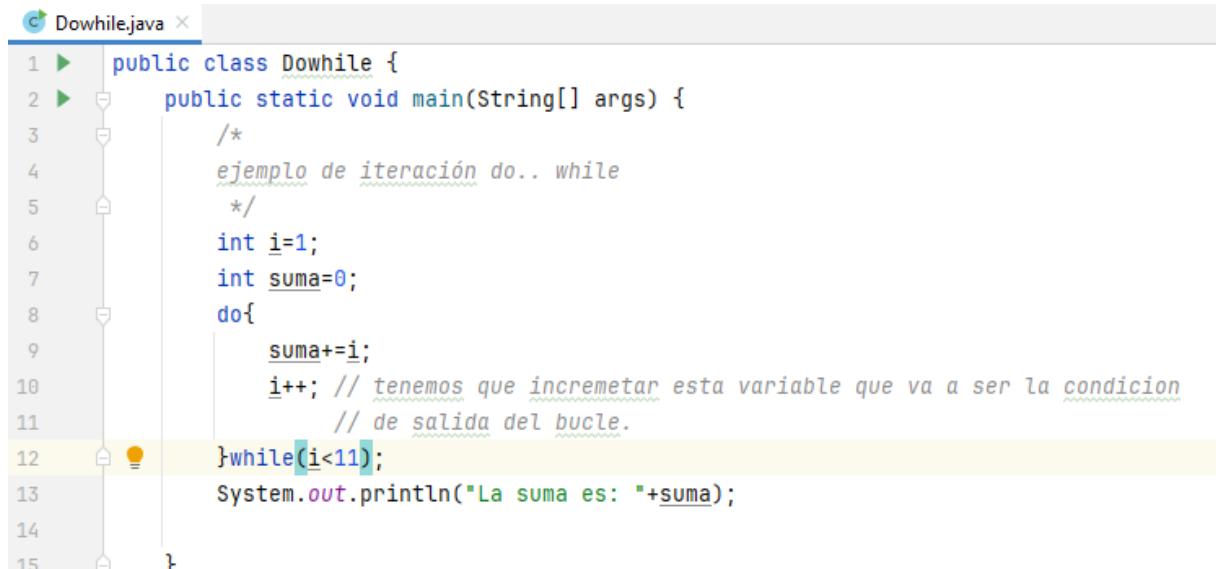
```
do {  
  // en este lugar está el bloque do  
} while(condición)
```

En este caso el bloque se ejecuta por lo menos una vez.

condición: expresión que devuelve un booleano o variable de tipo booleano, que indica si debo salir del bloque dowhile. Se entra al bloque se ejecuta mientras condición sea true, si no, se sale del bucle.

IMPORTANTE: ← el famoso “loop infinito” Para poder parar la repetición del bloque del dowhile, se debe modificar algún componente de la condición para hacer que evalúe en false y poder parar su ejecución.

Ejemplo 1



```
1 public class Dowhile {
2     public static void main(String[] args) {
3         /*
4          * ejemplo de iteración do.. while
5          */
6         int i=1;
7         int suma=0;
8         do{
9             suma+=i;
10            i++; // tenemos que incrementar esta variable que va a ser la condición
11                // de salida del bucle.
12        }while(i<11);
13        System.out.println("La suma es: "+suma);
14    }
15 }
```

El resultado al ejecutar el programa es de 55, es el de sumar los números enteros del 1 al 10, en cada iteración se guardan en una variable del tipo acumulador (suma) , que en cada iteración va guardando la suma de todos los números. Es importante de hacer el incremento en este caso de la variable i, que va a ser la condición de salida del bucle, sino se hace se genera un caso de bucle infinito.

Variable Acumulador

- Se debe arrancar en 0.
- Sirve para sumar determinada información que se nos indique.
- Ejemplos sumar todos los números del 1 al 100.
Sumar todos los números pares del 1 al 100

El anterior ejemplo de do..while es un ejemplo que tiene la variable acumulador en ese caso la variable acumulador se llama suma.

```
AcumuladorPares.java x
1 public class AcumuladorPares {
2     public static void main(String[] args) {
3         // Ejemplo de variable acumulador, donde se suman
4         // solo los números pares entre el 1 y el 100
5
6         int suma=0; // variable acumulador
7         for (int i=1;i<=100;i++){
8             if(i%2==0) // El if con una sola línea como bloque puede no llevar { }
9                 suma+=i;
10        }
11        System.out.println("La suma de números pares entre 1 y 100 es: "+suma);
12    }
13 }
14
```

El resultado de ejecutar este programa es : 2550, son la suma de los números pares entre 1 y 100, para saber si son números pares en el if la condición es que para ser par el módulo de la división por 2 debe ser igual a 0.

Variable Contador

Se debe arrancar en 0. Sirve para contar cada ocurrencia de determinados elementos.

Ejemplos: Cuantos números pares hay entre 0 y 10. Cuantas letras a tiene determinada palabra.

Ejemplo:

```
VariableContador.java x
1 public class VariableContador {
2     public static void main(String[] args) {
3         String palabra="Esto es una prueba";
4         // Quiero saber cuantas letras a hay dentro de la variable palabra.
5         int contador=0; // se inicializa el contador en 0
6         for (int i=0;i<palabra.length();i++){
7             if (palabra.charAt(i)=='a'){
8                 contador++; // Si la letra es una a incremento el contador
9             }
10        }
11        System.out.println("La cantidad de a en palabra es: "+contador);
12    }
13 }
14 }
```

El resultado es 2, ya que tenemos 2 letras a en la variable de tipo String llamada palabra.

Variable Bandera-Centinela-Llave

Es una variable que puede tener 2 valores posibles, true o false, prendido o apagado, 0 o 1.

Se inicializa en determinado valor antes de entrar a una estructura repetitiva (generalmente en while o do while). Cuando pasa determinada condición cambiamos su valor y no necesitamos seguir buscando más. Se usa como la condición de la estructura repetitiva, o parte de ella (condición) Ejemplo: Si tengo que buscar determinada letra en un String, cuando la encuentro cambio el valor de la variable para salir y no seguir buscando más.

```
VariableBandera.java x
1 public class VariableBandera {
2     public static void main(String[] args) {
3         // Variable Bandera
4         // En este caso si encontramos una letra determinada en
5         // un String salimos del bucle
6         String texto="Prueba de la bandera";
7         // si encontramos una b, salimos del bucle
8         boolean encuentre=false;
9         int i=0;
10        while(!encuentre && i<texto.length()){
11            if (texto.charAt(i)=='b'){
12                encuentre=true;
13            }
14            i++; // hay que incrementar el indice para recorrer el String
15        }
16        if (encuentre){
17            System.out.println("Se encontró la letra b en la posicion "+i);
18        } else {
19            System.out.println("No se encontro la b");
20        }
21    }
22 }
23
```

En este caso va a encontrar la b, va a devolver un mensaje que la encontró y la posición que estaba la misma. Una vez que la encuentra sale del loop, es una de las formas de no usar el comando break.