USO DE LOS CICLOS EN JAVA

Los ciclos o también conocidos como bucles, son una estructura de control de total importancia para el proceso de creación de un programa. Java y prácticamente todos los lenguajes más populares de la actualidad nos permiten hacer uso de estas estructuras y muchas más. Un ciclo en Java o bucle en Java (como prefieras llamarlo) permite repetir una o varias instrucciones cuantas veces lo necesitemos o sea necesario, por ejemplo, si quisiéramos escribir los números del uno al cien no tendría sentido escribir cien líneas de código mostrando un número en cada una de estas, para eso y para varias cosas más, es útil un ciclo. Un ciclo nos ayuda a llevar a cabo una tarea repetitiva en una cantidad de líneas muy pequeña y de forma prácticamente automática.

Existen diferentes tipos de ciclos o bucles en Java, cada uno tiene una utilidad para casos específicos y depende de nuestra habilidad y conocimientos poder determinar en qué momento es bueno usar alguno de ellos. Tenemos entonces a nuestra disposición los siguientes tipos de ciclos en Java:

- Ciclo for en Java
- Ciclo while en Java
- Ciclo do-while en Java

Como se mencionó anteriormente, cada uno de estos ciclos en Java tiene ciertas características que lo hacen útil para algunos casos específicos. A lo largo de los contenidos de esta sección de ciclos en Java veremos cada uno de estos al detalle, aprendiendo durante el proceso los componentes, sintaxis y esas características particulares que permiten decidir cuál usar en qué momento, veremos también el concepto de contador y acumulador que parte de la existencia de los ciclos en Java.

Aunque se intentará profundizar bastante en cada concepto, se hará enfocándose hacia el uso de los ciclos en el lenguaje Java y no tanto hacia la parte de la lógica de los ciclos en particular. Si se tiene problemas con la correcta comprensión de la lógica y utilidad de cualquier tipo de ciclo o de los ciclos en general, te invito a que primero leas acerca de la Fundamentación de los Ciclos, en dicha sección podrás comprender correctamente el funcionamiento de un ciclo y posteriormente podrás regresar a esta sección para aprender a implementar los ciclos en Java.

CICLO FOR EN JAVA. ESTRUCTURA, SINTAXIS Y USO DE UN CICLO FOR EN JAVA

Los ciclos for (o ciclos para) son una estructuras de control cíclica, nos permiten ejecutar una o varias líneas de código de forma iterativa (o repetitiva), pero teniendo cierto control y conocimiento sobre las iteraciones. En el ciclo for, es necesario tener un valor inicial y un valor final, y opcionalmente podemos hacer uso del tamaño del "paso" entre cada "giro" o iteración del ciclo.

En resumen, un ciclo for es una estructura iterativa para ejecutar un mismo segmento de código una cantidad de veces deseada; conociendo previamente un valor de inicio, un tamaño de paso y un valor final para el ciclo.

¿Cómo funciona un Ciclo For?

Para comprender mejor el funcionamiento del ciclo for, solucionemos un pequeño ejercicio práctico, supongamos que queremos mostrar los números pares (múltiplos de dos: P) entre el 500 y el 1000. Con esta información inmediatamente podemos determinar que por medio de un ciclo debemos mostrar una serie de números como la siguiente: 500 502 504 506 ... 600 ... 852 ... 906 ... 980 .. 1000. Tenemos entonces todo lo necesario para nuestro ciclo. Tenemos un valor inicial que sería el 500, un valor final (1000) y tenemos un tamaño de paso que es 2 (los números pares). Estamos ahora en capacidad de determinar los componentes esenciales para un ciclo for.

Vamos a ver ahora la sintaxis de un ciclo for en Java, así estaremos listos para usarlos en nuestros programas de ahora en adelante.

Sintaxis del Ciclo For en Java:

La sintaxis de un ciclo for es simple en Java, en realidad en la mayoría de los lenguajes de alto nivel es incluso muy similar, de hecho, con tan solo tener bien claros los 3 componentes del ciclo for (inicio, final y tamaño de paso) tenemos prácticamente todo hecho

```
For ( int i = valor inicial;  i <= valor final;  i = i + paso)
{
    ....
```

....

Bloque de Instrucciones....

....

....

}

Vamos ahora a ver línea por línea el anterior código para comprender todo y quedar claros. Posteriormente veremos un ejemplo con valores reales.

Línea 1:

En esta línea está prácticamente todo lo esencial de un ciclo for. La sintaxis es simple, tenemos una variable de control llamada i que es tipo entero (int), cabe notar que la variable se puede llamar como nosotros lo deseemos y puede ser del tipo de queramos también, sin embargo en la mayoría de los casos se usa la "i" como nombre y el entero como tipo, pero somos libres de modificar esto a nuestro gusto. Esta variable "i" se le asigna un valor inicial que puede ser cualquier número correspondiente al tipo de dato asignado. Posteriormente lo que haremos será especificar hasta donde irá nuestro ciclo por medio del valor final, ten en cuenta que cada uno de estos componentes es separado por un punto y coma ";", también es importante saber que la condición final puede ser cualquier cosa, mayor, menor, mayor o igual, menor o igual, sin embargo no tiene sentido que la condición sea por ejemplo un igual, pues nuestra variable de control siempre va a cambiar entre valores, menores o mayores que el valor final deseado, si fuera un igual no tendríamos un error de sintaxis, pero nuestro for básicamente no haría nada de nada. Finalmente el ultimo componente de esta primer línea es el tamaño del paso, este componente se especifica aumentando en la cantidad deseada la variable de control.

Línea 2:

En la línea 2 tenemos una llave abriendo "{" lo cual como seguramente ya sabrás indica que allí comienza el bloque de instrucciones que se ejecutaran cada vez que el ciclo de un "giro". Esta llave no es del todo obligatoria, sin embargo si no la ponemos solo se ejecutara dentro de nuestro ciclo la primera línea inmediatamente posterior a la declaración del ciclo, de modo que si deseamos que se ejecuten varias líneas dentro de nuestro ciclo, debemos usar las llaves

Línea 3 a 7:

En estas líneas es donde estarán todas las operaciones que queramos llevar a cabo de manera iterativa durante la ejecución del ciclo, este bloque podrá tener la cantidad de líneas necesarias incluso, como veremos más adelante dentro de estas podría haber uno o más ciclos, así que podrías tener todo un programa dentro de un ciclo.

Línea 8:

En esta última línea hacemos uso de la llave cerrando "}", una vez más como seguramente ya sabrás esta nos indica que allí termina el bloque del ciclo for y se dará por terminada la ejecución de este para continuar ejecutando el resto del algoritmo.

No te preocupes si no comprendiste muy bien lo que acabo de escribir, estoy seguro que con un par de ejemplos que veremos a continuación, te va a quedar todo claro.

Ejemplos de Ciclo For en Java

A continuación vamos a ver unos cuantos ejemplos para comprender de manera adecuada el uso de los ciclos for en Java.

Ejemplo 1: Mostrar en pantalla los números pares

Vamos a retomar el ejemplo anterior, donde deseábamos sacar los números pares entre el numero 500 y el 1000, es un ejemplo sencillo con el que nos aseguraremos de haber comprendido bien lo anterior:

Solución Ejemplo 1:

```
for(int i=500;i<=1000;i+=2)

{//Notemos que escribir i+=2 es similar a escribir i = i + 2

   System.out.println(i);

}
```

El código funcional completo sería el siguiente:

```
public class CicloFor
```

```
{
   public static void main(String args[])
   {
      for(int i=500;i<=1000;i+=2)
      {
         System.out.println(i);
      }
   }
}
```

Ejemplo 2: Cuenta regresiva en un ciclo for

Ahora veremos otro ejemplo sencillo en cual haremos que el ciclo for también haga sus iteraciones en sentido inverso, es decir disminuyendo el valor de la variable de control vamos a imprimir por pantalla una cuenta regresiva desde el número 100 hasta el 0, veamos:

Solución Ejemplo 2:

Para este caso, debido a que queremos ir de un número mayor a uno más pequeño, por lo tanto para este ejemplo el valor inicial será 100 y el valor final será 0. Adicional, el tamaño de paso será de 1 negativo, es decir, -1, así:

```
for(int i = 100; i > 0; i--)
{//Notemos que escribir i-- es lo mismo a escribir i = i - 1
   System.out.println(i);
}
```

El código funcional completo sería el siguiente:

```
public class CicloInverso
{
   public static void main(String args[])
   {
      for(int i = 100; i > 0; i--)
      {
         System.out.println(i);
      }
   }
}
```

Ejemplo 3: Contador con un ciclo for

Para este ejemplo haremos algo un poco más complejo. El ejemplo consiste en contar al interior de un ciclo for, cuántos números entre el 0 y el 10.000 son múltiplos del 20. Para ello haremos uso del operador % (módulo) que obtiene el residuo de una división y también usaremos un pequeño condicional para verificar que el módulo sea cero al dividir por 20.

Nota: El operador de módulo (%) obtiene el residuo de una división, por tanto cuando el residuo es cero implica que la division es exacta y el dividendo es un múltiplo del divisor. Por ejemplo 10%3 nos dará el residuto de dividir 10 entre 3, el cual es 1, si calculamos 120%20 nos dará cero, pues 120 es múltiplo de 20 (20 * 6 = 120).

Solución Ejemplo 3:

Para este caso el valor inicial será 0 y el valor final será 10000. Adicional, el tamaño de paso será de 1 (este es el caso más común). Al interior del ciclo, en cada iteración verificaremos si el número en el que estamos es divisible por 20 o no y en caso afirmativo aumentaremos el contador en una unidad así:

Nota: Este problema se puede solucionar de maneras mucho más optimas y adecuadas, pero para fines de comprensión y facilidad lo harémos usando un ciclo for de uno en uno.

```
int contador = 0; //Iniciamos el contador en cero


for(int i = 0; i <= 10000; i++)
{//Notemos que escribir i++ es similar a escribir i = i + 1
   if(i % 20 == 0) //Preguntamos si el residuo es 0 (es múltiplo de 20)
   {
      contador++; //Si es múltiplo aumentamos el contador en 1
   }
   //Si no es múltiplo no hacemos nada
}


//Mostramos el valor del contador
System.out.println(contador);
```

El código funcional completo sería el siguiente:

```
public class Multiplos
{
   public static void main(String args[])
   {
      int contador = 0; //Iniciamos el contador en cero
```

```
        for(int  i = 0; i <= 10000; i++)
        {
           if(i  % 20 == 0) //Preguntamos  si el residuo  es 0 (es múltiplo  de 20)
           {
              contador++;  //Si es múltiplo  aumentamos  el contador en 1
           }
           //Si no es múltiplo  no hacemos  nada
        }
        //Mostramos  el valor del contador
        System.out.println(contador);
    }
}
```

Este ciclo for nos permitirá saber que existen 501 múltiplos del número 20 en los números del 0 al 10000.

En efecto los ciclos for, son bastante útiles, sin embargo desde el punto de vista de la eficiencia, es recomendable evitarlos en la medida de lo posible, siempre que vayas a usar un ciclo for, deberías preguntarte primero si es totalmente necesario o si existe una manera más efectiva de hacerlo. Evidentemente habrá situaciones en las que será casi que imprescindible usar el ciclo for, pues por algo existen. Está entonces en nuestra habilidad determinar cuándo usarlos y mejor aún cómo usarlos de manera efectiva.

CICLO WHILE EN JAVA. ESTRUCTURA Y SINTAXIS. CÓMO Y PARA QUÉ USAR UN CICLO WHILE EN JAVA

Los ciclos while son también una estructura cíclica, que nos permite ejecutar una o varias líneas de código de manera repetitiva sin necesidad de tener un valor inicial e incluso a veces sin siquiera conocer cuándo se va a dar el valor final que

esperamos, los ciclos while, no dependen directamente de valores numéricos, sino de valores booleanos, es decir su ejecución depende del valor de verdad de una condición dada, verdadera o falso, nada más. De este modo los ciclos while, son mucho más efectivos para condiciones indeterminadas, que no conocemos cuando se van a dar a diferencia de los ciclos for, con los cuales se debe tener claro un principio, un final y un tamaño de paso.

¿Cómo funciona un Ciclo While?

Para comprender mejor el funcionamiento del ciclo while, pongamos un buen ejemplo, imaginemos que, por algún motivo, queremos pedirle a un usuario una serie de números cualquiera y que solo dejaremos de hacerlo cuando el usuario ingrese un número mayor a 500. Como imaginarás, aquí no podríamos utilizar un ciclo for, pues no tenemos idea alguna de cuándo al usuario se le va a ocurrir ingresar un número mayor que 500 (si es que lo hace). Es algo indeterminado para nosotros. Sin embargo, el ciclo while nos permite ejecutar una acción de forma "infinita" hasta que se cumpla alguna condición específica, en nuestro caso, sería que el número ingresado sea mayor a 500. De modo que, si el usuario nos ingresa de manera sucesiva los siguientes números 1, 50, 99, 49, 21, 30, 500..., nuestro programa no finalizara, pues ninguno de estos números es mayor que 500. Sin embargo, si nos ingresara el número 800, el programa finalizaría inmediatamente.

Sintaxis del Ciclo While en Java:

La sintaxis de un ciclo while es incluso más simple y "legible" que la del ciclo for en Java, pues simplemente requerimos tener clara una condición de parada. En realidad, en la mayoría de los lenguajes de alto nivel la manera de escribir un ciclo while (la sintaxis) es incluso muy similar, así que con tan solo tener bien clara una condición de finalización para el ciclo tendremos prácticamente todo hecho.

```
while(condición de finalización) //por ejemplo numero == 500
{
    ....
    ....
  Bloque de Instrucciones....
    ....
```

....

}

Vamos entonces a ver línea por línea el anterior código para comprender todo y quedar claros. Posteriormente veremos el ejemplo planteado anteriormente y su solución.

Línea 1:

En esta línea está prácticamente todo lo esencial de un ciclo while. La sintaxis es bastante simple. Tenemos al interior de los paréntesis una condición cualquiera, es decir por ejemplo "==", ">", "<", ">=", "<=", "!=" o algunas más que se nos puedan ocurrir. Esta condición que especifiquemos allí, es la que permitirá que el ciclo se siga ejecutando hasta que en algún momento esta misma condición deje de cumplirse, de esta forma si por ejemplo estamos verificando que un número cualquiera sea igual a 50, el ciclo se ejecutará solo cuando dicho número sea exactamente igual a 50. En cuanto su valor cambie a cualquier otro, el ciclo while finalizará y continuará con el resto de la ejecución del programa. De esta forma, es evidente que la condición que allí ingresemos siempre deberá tomar un valor booleano (true o false).

Línea 2:

En la línea 2 tenemos una llave abriendo "{" lo cual como sabemos indica que allí comienza un bloque de instrucciones que se ejecutaran cada vez que el ciclo de un "giro". Esta llave no es del todo obligatoria, sin embargo, si no la ponemos solo se ejecutara dentro de nuestro ciclo while la primera línea inmediatamente posterior a la declaración del ciclo, de modo que si deseamos que se ejecuten varias líneas dentro de nuestro ciclo, debemos usar las llaves

Línea 3 a 7:

En estas líneas es donde estarán todas las operaciones que queramos llevar a cabo de manera iterativa durante la ejecución del ciclo, este bloque podrá tener la cantidad de líneas necesarias incluso, como veremos más adelante dentro de estas podría haber uno o más ciclos, así que podrías tener todo un programa dentro de un ciclo.

Línea 8:

En esta última línea hacemos uso de la llave cerrando "}", una vez más como seguramente ya debemos saber esta nos indica que allí termina el bloque del ciclo

while y se dará por terminada la ejecución de este para continuar ejecutando el resto del algoritmo.

Ejemplos de Ciclo While en Java

A continuación vamos a ver unos cuantos ejemplos para comprender de manera adecuada el uso de los ciclos while en Java

Ejemplo 1: Pedir números por pantalla hasta que alguno sea mayor a 500

Vamos a retomar el ejemplo anterior, donde queremos hacer que nuestro programa le pida a un usuario una serie de números cualquiera y que solo dejaremos de hacerlo cuando el usuario ingrese un número mayor a 100, una vez más es un ejemplo sencillo con el que nos aseguraremos de haber comprendido bien todos los conceptos anteriores:

Solución Ejemplo 1:

Para solucionar esto, debemos tener clara cuál va a ser la condición que se debe cumplir para que el ciclo este pidiendo el numero contantemente, el ciclo se va a detener solo cuando el número ingresado sea mayor que 500, así que la condición para que se siga ejecutando es que el número sea menor o igual a 500, ¿Comprender la lógica?, si para que se detenga el número debe ser mayor a 500, entonces para seguirse ejecutando el número debe ser menor o igual a 500, veámoslo entonces

```
Scanner sc = new Scanner(System.in);

int numero = sc.nextInt();

while (numero <= 500)

{

   System.out.println("Ingrese un numero ");

   numero = sc.nextInt();

}
```

El código funcional completo y un tanto más amigable para el usuario sería el siguiente:

```
import java.util.Scanner;


public class CicloWhile
{
   public static void main(String[] args)
   {
      Scanner sc = new Scanner(System.in);
      int numero = sc.nextInt();


      while(numero <= 500)
      {
         System.out.println("Ingrese un numero ");
         numero = sc.nextInt();
      }
   }
}
```

En efecto los ciclos while, son bastante útiles, sin embargo, desde el punto de vista de la eficiencia al igual que con los ciclos for, es recomendable evitarlos en la medida de lo posible, siempre que vayas a usar un ciclo while o cualquier tipo de ciclo en realidad, deberías preguntarte primero si es totalmente necesario o si existe una manera más efectiva de hacerlo. Evidentemente habrá situaciones en las que será casi que imprescindible usar el ciclo, pues por algo existen, esta entonces en nuestra habilidad determinar cuándo usarlos y mejor aún cómo usarlos de manera efectiva.

Crea una aplicación que permita adivinar un número. La aplicación genera un número aleatorio del 1 al 100. A continuación va pidiendo números y va respondiendo si el número a adivinar es mayor o menor que el introducido, además de los intentos que te quedan (tienes 10 intentos para acertarlo).

El programa termina cuando se acierta el número (además te dice en cuantos intentos lo has acertado), si se llega al límite de intentos te muestra el número que había generado.

Este es el primer ejercicio de repaso de estructuras repetitivas en JAVA, o lo que es lo mismo, el primero en el que veremos el uso de estructuras como for, while o do-while… ahora iremos analizando cual es, a mi juicio, la opción más adecuada en cada caso.

Para la realización de este primer programa, vemos que podemos dividirlo en tres partes:

Obtención de número aleatorio.

Para obtener un número aleatorio, debemos hacer uso de otra función del método ya bastante frecuentado ***Math*** de Java.

En esta ocasión, en concreto haremos uso de ***Math.random()***, la cual nos devuelve un valor aleatorio entre 0.0 y 1.0. *¿Cómo hacemos para que el valor que nos devuelva esté entre 1 y 100?*

*Math.random()*numerosRango + numeroInicial*

Para conseguir ese valor, debemos multiplicar **Math.random() * (cantidad de números en el rango**) en este caso, 100 (incluye el 1 y el 100).

Para que después empiece a mostrar desde el número 1, debemos sumar 1.

Por ejemplo, si en vez de números del 1 al 100 queremos, 50 números, del 51 al 100 debemos poner ***Math.random() * 50 +51;***

Elección estructura repetitiva (do-while)

Tipos de bucle

Con bucle (for)

Instrucción:

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

for, inicializa la variable y desarrolla una acción un número determinado de veces.

Con bucle (while)

Instrucción:

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

while, realiza la comprobación en primer lugar y después ejecuta las acciones. Puede darse el caso que el valor de la variable no cumpla la condición y no llegue a ejecutarse.

Con bucle (do-while)

Instrucción:

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

do-while, realiza la comprobación al finalizar las instrucciones. Se realiza siempre al menos una vez.

PROGRAMA DE EJEMPLO:

Crea una aplicación que permita adivinar un número. La aplicación genera un número aleatorio del 1 al 100. A continuación va pidiendo números y va respondiendo si el número a adivinar es mayor o menor que el introducido, además de los intentos que te quedan (tienes 10 intentos para acertarlo). El programa termina cuando se acierta el número (además te dice en cuantos intentos lo has acertado), si se llega al límite de intentos te muestra el número que había generado.

```
import java.util.Scanner;

// Inicio del programa y declaración de variables:

public class Ej01AdivinaNumero {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int numeroAleatorio;

int intento;
```

```
int contador = 10;
// Obtención de número aleatorio
numeroAleatorio = (int) (Math.random()*100+1);
System.out.println("Intenta adivinar un número aleatorio entre el 1 y 100. "
+ "Tienes 10 intentos.");
System.out.println(numeroAleatorio);
// Realización del ciclo do-while
do {
System.out.println("Número contador: " + contador);
System.out.print("Introduce el número que creas posible: ");
intento = scanner.nextInt();
if (intento > numeroAleatorio) {
System.out.println("El número que buscas es menor, te quedan "
+ (contador-1) + " intentos: ");
} else if (intento < numeroAleatorio){
System.out.println("El número que buscas es mayor, te quedan "
+ (contador-1) + " intentos: ");
} else {
System.out.print("¡CORRECTO! "+ numeroAleatorio + " era el número que estabas "
+ "buscando, has necesitado " + (10 - (contador-1)) +" intentos.");
}
contador--;
```

```
} while (intento != numeroAleatorio && contador > 0);

if (contador == 0) {

System.out.println("Has perdido. El numero aleatorio era " + numeroAleatorio);

}

} }
```