

Loops in Java

we'll cover the four types of loops in Java: the [for loop](#), [enhanced for loop \(for-each\)](#), [while loop](#) and [do-while loop](#).

Loops Concepts

All loops in Java use the same building blocks that we're going to define here.

- loop initialization – setting the initial value of variable(s) to be used in the loop
- boolean condition – this is a boolean expression that determines whether to continue looping for one more iteration
- step value (update value) – an update to the loop variable(s)
- loop body – execute the main part of the loop that does the processing

For Loop

For loops are best used when you know how many times you have to loop over something. For example, when looping over an array of numbers you will need to loop over as many times as there are elements in the array.

For Loop Structure

Java for loops are structured to follow this order of execution:

- 1) loop initialization
- 2) boolean condition – if true, continue to next step; if false, exit loop
- 3) loop body
- 4) step value
- 5) repeat from step 2 (boolean condition)

Declaring and Initializing loop control variable Checking condition Incrementing loop control variable

```

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

    // Loop statements to be executed

}

```

Let's see below example which prints all the values divisible by 7 in the range of 1 to 100 in reverse order.

```

public class ForLoopDivisibleBy7 {
    public static void main(String[] args) {
        for(int i=100; i>0; --i)
        {
            if(i%7 == 0)
            {
                System.out.print(i);
                System.out.print(", ");
            }
        }
    }
}

```

Output:

```

<terminated> ForLoopDivisibleBy7 [Java Application] C:\Program Files\Ja
98, 91, 84, 77, 70, 63, 56, 49, 42, 35, 28, 21, 14, 7,

```

Enhanced for loop

The enhanced for loop allows you to cycle through an array/ Collection without specifying the starting and ending points for the loop control variable. The general form of the enhanced for loop version is shown here:

```

for(type itr-var : collection) statement-block

```

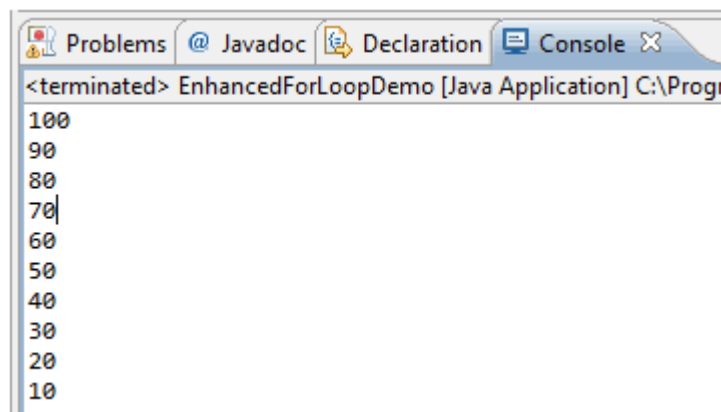
Here, type specifies the type and itr-var specifies the name of an iteration variable that will receive the elements from a collection, one at a time, from beginning to end. Because the iteration variable receives values from the collection, the type must be the same as (or compatible with) the elements

stored in the collection. Thus, when iterating over arrays, the type must be compatible with the base type of the array.

Below example shows the use of enhanced for loop.

```
public class EnhancedForLoopDemo {
    public static void main(String[] args) {
        int [] myArray = new int[10];
        int i=0;
        //Traditional for loop to populate
        for(int k=100; k>0; k=k-10, i++)
        {
            myArray[i]=k;
        }
        //Enhanced for loop to print elements of array
        for(int loopVal: myArray)
        {
            System.out.println(loopVal);
        }
    }
}
```

Output:



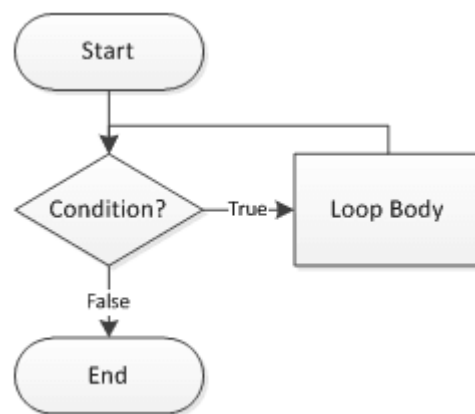
As this output shows, the enhanced for loop style automatically cycles through an array in sequence from the lowest index to the highest.

While Loops in Java

Description

In computer programming, a loop is a sequence of instructions that is continually repeated until a certain condition is reached. Imagine you have to

write a program which performs a repetitive task such as printing 1 to 100. Writing 100 print statements would not be wise thing to do. Loops are specifically designed to perform repetitive tasks with one set of code. Loops save a lot of time. A loop is a structure that allows repeated execution of a block of statements. Within a looping structure, a Boolean expression is evaluated. If it is true, a block of statements called the loop body executes and the Boolean expression is evaluated again. As long as the expression is true, the statements in the loop body continue to execute. When the Boolean evaluation is false, the loop ends.



In Java there are three types of loops:

- A while loop, in which the loop-controlling Boolean expression is the first statement in the loop
- A for loop, which is usually used as a concise format in which to execute loops
- A do...while loop, in which the loop-controlling Boolean expression is the last statement in the loop

While Loops

The while loop is good for scenarios where you don't know how many times a block or statement should repeat, but you want to continue looping as long as some condition is true. A while statement looks like below. In Java, a while loop consists of the keyword `while` followed by a Boolean expression within parentheses, followed by the body of the loop, which can be a single statement or a block of statements surrounded by curly braces.

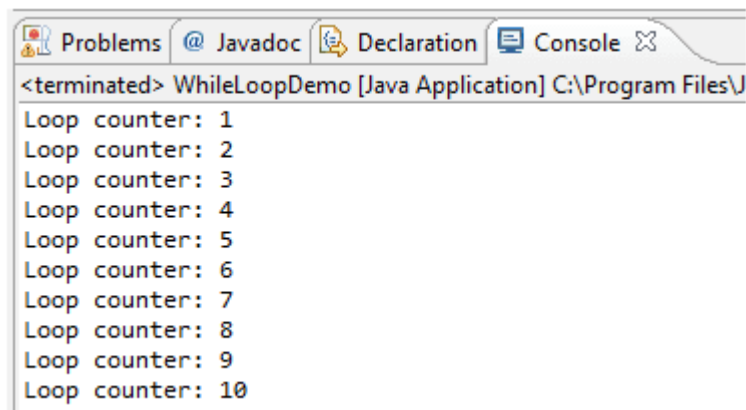
```
while (expression) { // do stuff }
```

You can use a while loop when you need to perform a task a predetermined number of times. A loop that executes a specific number of times is a definite

loop or a counted loop. To write a definite loop, you initialize a loop control variable, a variable whose value determines whether loop execution continues. While the Boolean value that results from comparing the loop control variable and another value is true, the body of the while loop continues to execute. In the body of the loop, you must include a statement that alters the loop control variable.

```
public class WhileLoopDemo {
    public static void main(String[] args) {
        int var = 1;
        int limit=11;
        while (var < limit)
        {
            System.out.println("Loop counter: " + var);
            var++;
        }
    }
}
```

Output:



```
<terminated> WhileLoopDemo [Java Application] C:\Program Files\J
Loop counter: 1
Loop counter: 2
Loop counter: 3
Loop counter: 4
Loop counter: 5
Loop counter: 6
Loop counter: 7
Loop counter: 8
Loop counter: 9
Loop counter: 10
```

The key point to remember about a while loop is that it might not ever run. If the test expression is false the first time the while expression is checked, the loop body will be skipped and the program will begin executing at the first statement after the while loop. Let's take an example to understand this. In below program user enters the value of loop counter variable, If counter variable is less than 5, then the loop will execute and increment the counter variable by one till counter value is equal to 5. If counter variable is more than or equal to 5, the loop will not execute.

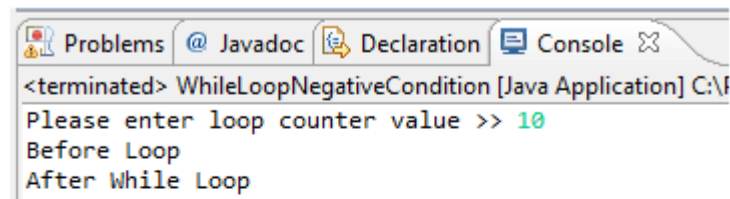
```
import java.util.Scanner;
public class WhileLoopNegativeCondition {
    public static void main(String[] args)
    {
```

```

int counter;
Scanner inputDevice = new Scanner(System.in);
System.out.print("Please enter loop counter value >> ");
counter = inputDevice.nextInt();
System.out.println("Before Loop");
while (counter < 5)
{
    System.out.println("Inside Loop- Counter= "+ counter);
    counter++;
}
System.out.println("After While Loop");
}
}

```

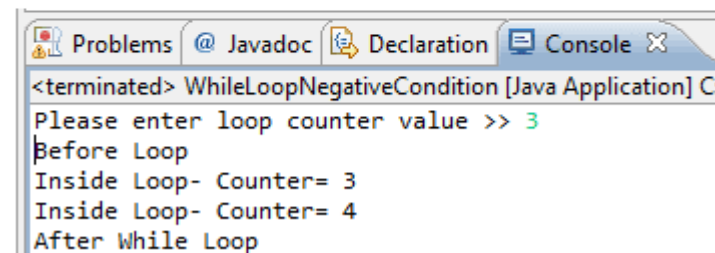
Output:



```

<terminated> WhileLoopNegativeCondition [Java Application] C:\F
Please enter loop counter value >> 10
Before Loop
After While Loop

```



```

<terminated> WhileLoopNegativeCondition [Java Application] C
Please enter loop counter value >> 3
Before Loop
Inside Loop- Counter= 3
Inside Loop- Counter= 4
After While Loop

```

Important Points while using loops

- Loop condition/expression can be true always, which makes our loop infinite. This is bad programming practice as it might result in memory exception. Below statement is valid but not good to have in our program.

```

while (true) { // Some stuff
while (2<4) // Some stuff

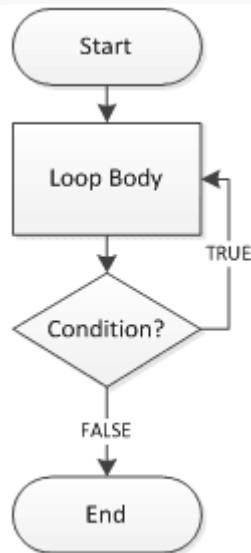
```

- It is very common to alter the value of a loop control variable by adding 1 to it or incrementing the variable. However, not all loops are controlled by adding 1 sometimes we can control by subtracting 1 from a loop control variable or decrementing it.

do ... while Loops

The do loop is similar to the while loop, except that the expression is not evaluated until after the do loop's code is executed. Therefore the code in a do loop is guaranteed to execute at least once. The following shows a do loop syntax:

```
do { //Loop Body } while (Condition);
```



If so, you want to write a loop that checks at the “bottom” of the loop after the first iteration. The do...while loop checks the value of the loop control variable at the bottom of the loop after one repetition has occurred. Below sample code explain do... while loop.

```
public class DoWhileLoopDemo {  
    public static void main(String[] args) {  
        int i = 10;  
        do{  
            i=i+10;  
            System.out.println("Loop Counter Variable=" +i);  
        }  
        while (i< 10);  
    }  
}
```

Output:

