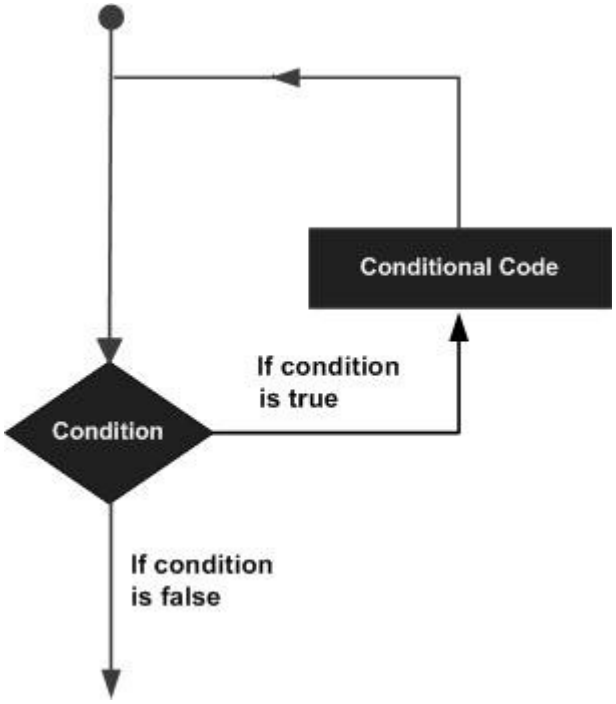
Control statements

Java - Loop Control

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A **loop** statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages −



Java programming language provides the following types of loop to handle looping requirements. Click the following links to check their detail.

A **while** loop statement in Java programming language repeatedly executes a target statement as long as a given condition is true.

Syntax

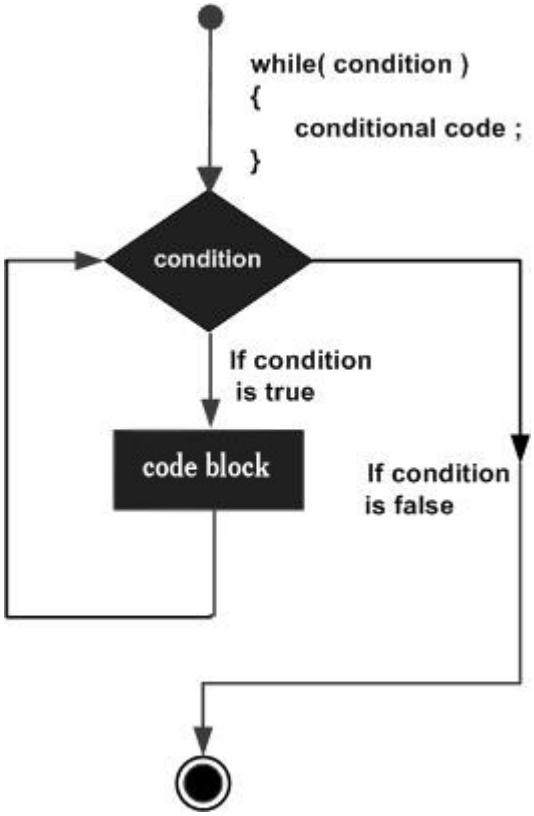
The syntax of a while loop is −

```
while(Boolean_expression) {
  // Statements
}
```

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non zero value.

When executing, if the *boolean_expression* result is true, then the actions inside the loop will be executed. This will continue as long as the expression result is true.
When the condition becomes false, program control passes to the line immediately following the loop.

Flow Diagram



Here, key point of the *while* loop is that the loop might not ever run. When the expression is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Example

```
public class Test {

  public static void main(String args[]) {
    int x = 10;

    while( x < 20 ) {
      System.out.print("value of x : " + x );
      x++;
      System.out.print("\n");
    }
  }
}
```

This will produce the following result −

Output

```
value of x : 10
value of x : 11
```

```
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

for loop

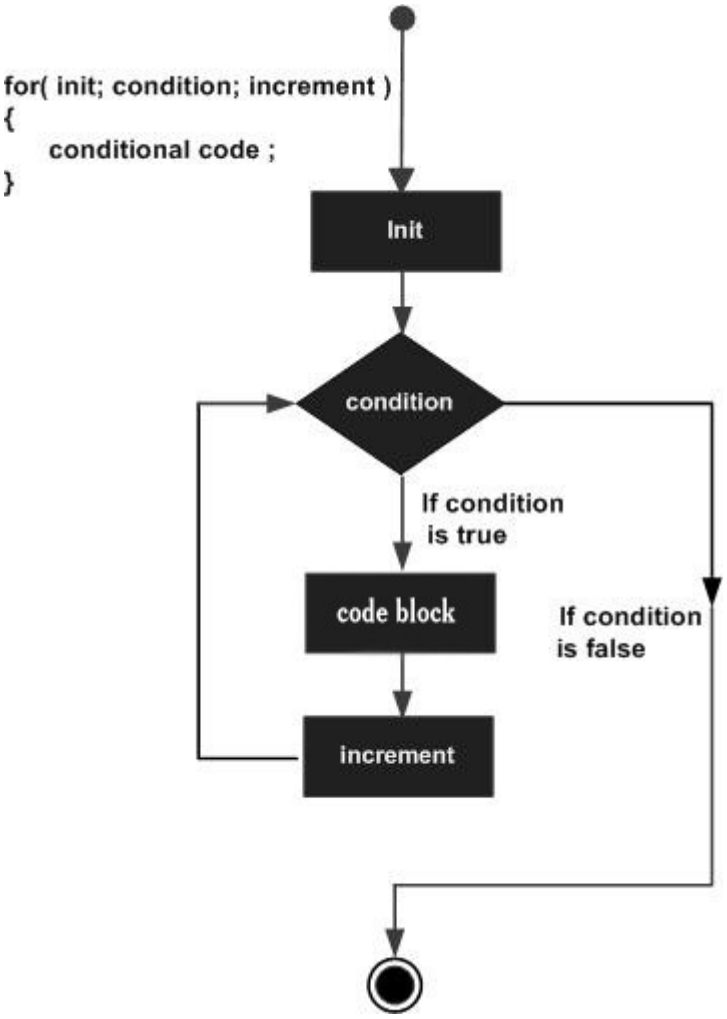
A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to be executed a specific number of times.
A **for** loop is useful when you know how many times a task is to be repeated.
Syntax
The syntax of a for loop is −

```
for(initialization; Boolean_expression; update) {
   // Statements
}
```

Here is the flow of control in a **for** loop −

- The **initialization** step is executed first, and only once. This step allows you to declare and initialize any loop control variables and this step ends with a semi colon (;).
- Next, the **Boolean expression** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop will not be executed and control jumps to the next statement past the for loop.
- After the **body** of the for loop gets executed, the control jumps back up to the update statement. This statement allows you to update any loop control variables. This statement can be left blank with a semicolon at the end.
- The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats (body of loop, then update step, then Boolean expression). After the Boolean expression is false, the for loop terminates.

Flow Diagram



Example

Following is an example code of the for loop in Java.

```
public class Test {

   public static void main(String args[]) {

      for(int x = 10; x < 20; x = x + 1) {
         System.out.print("value of x : " + x );
         System.out.print("\n");
      }
   }
}
```

This will produce the following result −

Output

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
```

value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19

do...while

A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax

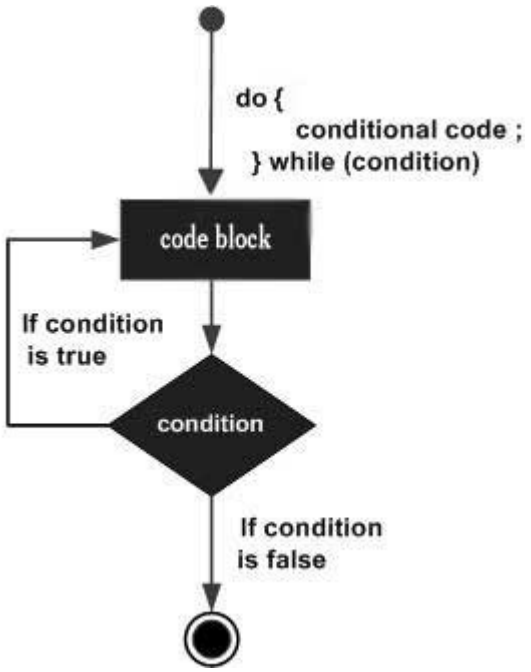
Following is the syntax of a do...while loop −

```
do {
   // Statements
}while(Boolean_expression);
```

Notice that the Boolean expression appears at the end of the loop, so the statements in the loop execute once before the Boolean is tested.

If the Boolean expression is true, the control jumps back up to do statement, and the statements in the loop execute again. This process repeats until the Boolean expression is false.

Flow Diagram



Example

```
public class Test {

  public static void main(String args[]) {
    int x = 10;

    do {
      System.out.print("value of x : " + x );
      x++;
```

```
      System.out.print("\n");
    }while( x < 20 );
  }
}
```

This will produce the following result −

Output

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

Enhanced for loop in Java

As of Java 5, the enhanced for loop was introduced. This is mainly used to traverse collection of elements including arrays.

Syntax

Following is the syntax of enhanced for loop −

```
for(declaration : expression) {
  // Statements
}
```

- **Declaration** − The newly declared block variable, is of a type compatible with the elements of the array you are accessing. The variable will be available within the for block and its value would be the same as the current array element.
- **Expression** − This evaluates to the array you need to loop through. The expression can be an array variable or method call that returns an array.

Example

```
public class Test {

  public static void main(String args[]) {
    int [] numbers = {10, 20, 30, 40, 50};

    for(int x : numbers ) {
      System.out.print( x );
      System.out.print(",");
    }
    System.out.print("\n");
    String [] names = {"James", "Larry", "Tom", "Lacy"};

    for( String name : names ) {
```

```
        System.out.print( name );
        System.out.print(",");
      }
   }
}
```

This will produce the following result −

Output

```
10, 20, 30, 40, 50,
James, Larry, Tom, Lacy,
```