Loops in Java:

Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true.

Java provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.

They are:

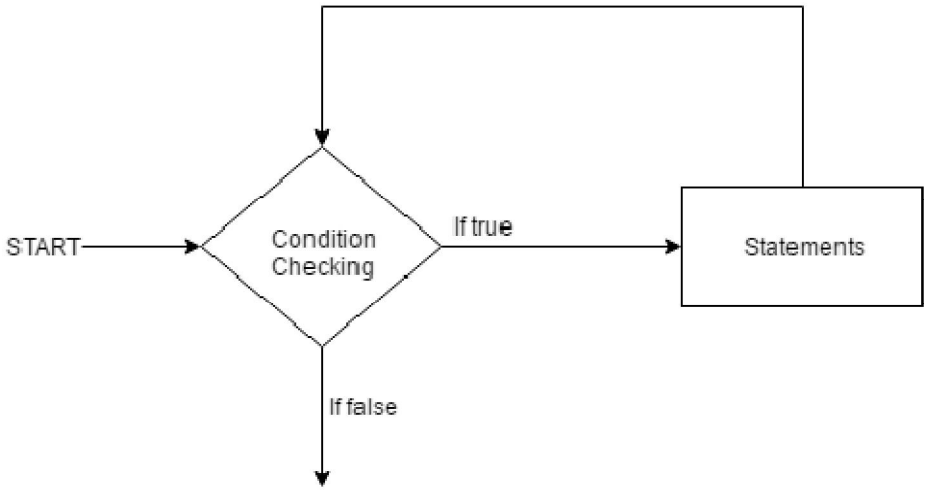
1. while loop
2. for loop
3. do-while loop

while loop: A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

Syntax :

```
while (boolean condition)
{
  loop statements...
}
```

Flowchart:



- While loop starts with the checking of condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason it is also called **Entry control loop.**
- Once the condition is evaluated to true, the statements in the loop body are executed. Normally the statements contain an update value for the variable being processed for the next iteration.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.

Example Program:

```
// Java program to illustrate while loop
class whileLoopDemo
{
   public static void main(String args[])
   {
      int x = 1;

      // Exit when x becomes greater than 4
      while (x <= 4)
      {
         System.out.println("Value of x:" + x);

         //increment the value of x for next iteration
         x++;
      }
   }
}
```

Output:

```
Value of x:1
Value of x:2
Value of x:3
Value of x:4
```

for loop:

for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

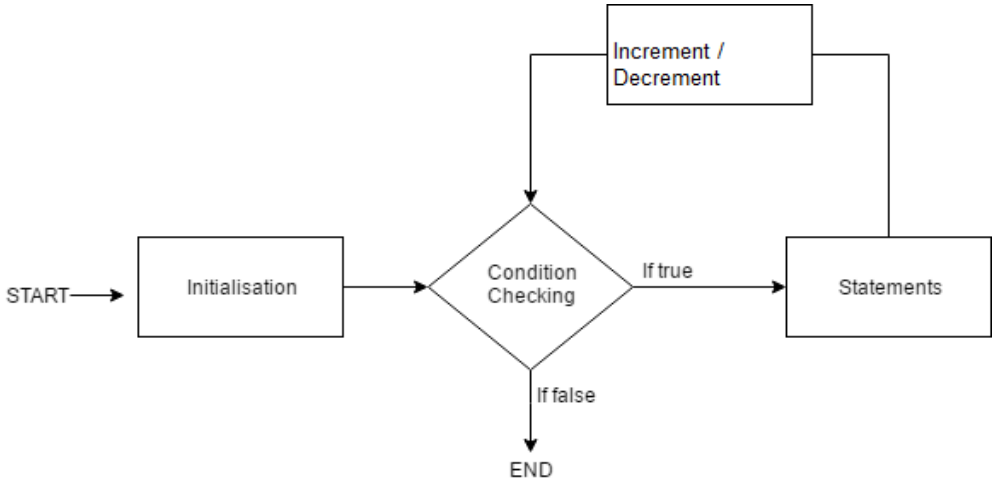
Syntax:

```
for (initialization condition; testing condition; increment/decrement)
{
   statement(s)
}
```

- **Initialization condition:** Here, we initialize the variable in use. It marks the start of a for loop. An already declared variable can be used or a variable can be declared, local to loop only.

- **Testing Condition:** It is used for testing the exit condition for a loop. It must return a Boolean value. It is also an **Entry Control Loop** as the condition is checked prior to the execution of the loop statements.
- **Statement execution:** Once the condition is evaluated to true, the statements in the loop body are executed.
- **Increment/ Decrement:** It is used for updating the variable for next iteration.
- **Loop termination:** When the condition becomes false, the loop terminates marking the end of its life cycle.

Flowchart:



Example:

```
// Java program to illustrate for loop.
class forLoopDemo
{
   public static void main(String args[])
   {
      // for loop begins when x=2
      // and runs till x <=4
      for (int x = 2; x <= 4; x++)
         System.out.println("Value of x:" + x);
   }
}
```

Output:

```
Value of x:2
Value of x:3
Value of x:4
```

Enhanced For loop

Enhanced for loop provides a simpler way to iterate through the elements of a collection or array. It is inflexible and should be used only when there is a need to iterate through the elements in sequential manner without knowing the index of currently processed element.

Syntax:

```
for (T element:Collection obj/array)
{
   statement(s)
}
```

Suppose there is an array of names and we want to print all the names in that array. Let's see the difference with these two examples

Enhanced for loop simplifies the work as follows-

Example: Java program to illustrate enhanced for loop

```
public class enhancedforloop
{
   public static void main(String args[])
   {
      String array[] = {"IIIT", "RGUKT", "Basar"};
      //enhanced for loop
      for (String x:array)
      {
         System.out.println(x);
      }
      /* for loop for same function
      for (int i = 0; i < array.length; i++)
      {
         System.out.println(array[i]);
      }
      */
   }
}
```

Output:

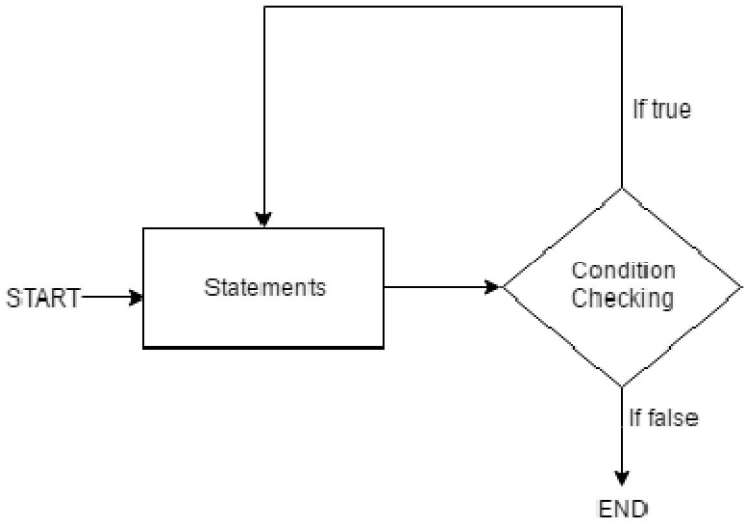
```
IIIT
RGUKT
Basar
```

do while: do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of **Exit Control Loop.**

Syntax:

```
do
{
   statements..
}
while (condition);
```

Flowchart:



- do while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
- After the execution of the statements, and update of the variable value, the condition is checked for true or false value. If it is evaluated to true, next iteration of loop starts.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.
- It is important to note that the do-while loop will execute its statements atleast once before any condition is checked, and therefore is an example of exit control loop.

Example: Java program to illustrate do-while loop

```
class dowhileloopDemo
{
   public static void main(String args[])
   {
      int x = 21;
      do
      {
```

```
            System.out.println("Value of x:" + x);
            x++;
          }
          while (x < 20);
        }
      }
```

Output:

```
Value of x: 21
```

Example Programs:

1. Java Program to Calculate the Sum of Natural Numbers
2. Java Program to Find Factorial of a Number
3. Java Program to Generate Multiplication Table
4. Java Program to Display Fibonacci Series
5. Java Program to Find GCD of two Numbers
6. Java Program to Find LCM of two Numbers
7. Java Program to Display Characters from A to Z using loop
8. Java Program to Count Number of Digits in an Integer
9. Java Program to Reverse a Number
10. Java Program to Calculate the Power of a Number
11. Java Program to Check Whether a Number is Palindrome or Not
12. Java Program to Check Whether a Number is Prime or Not
13. Java Program to Display Prime Numbers Between Two Intervals
14. Java Program to Check Armstrong Number
15. Java Program to Display Factors of a Number