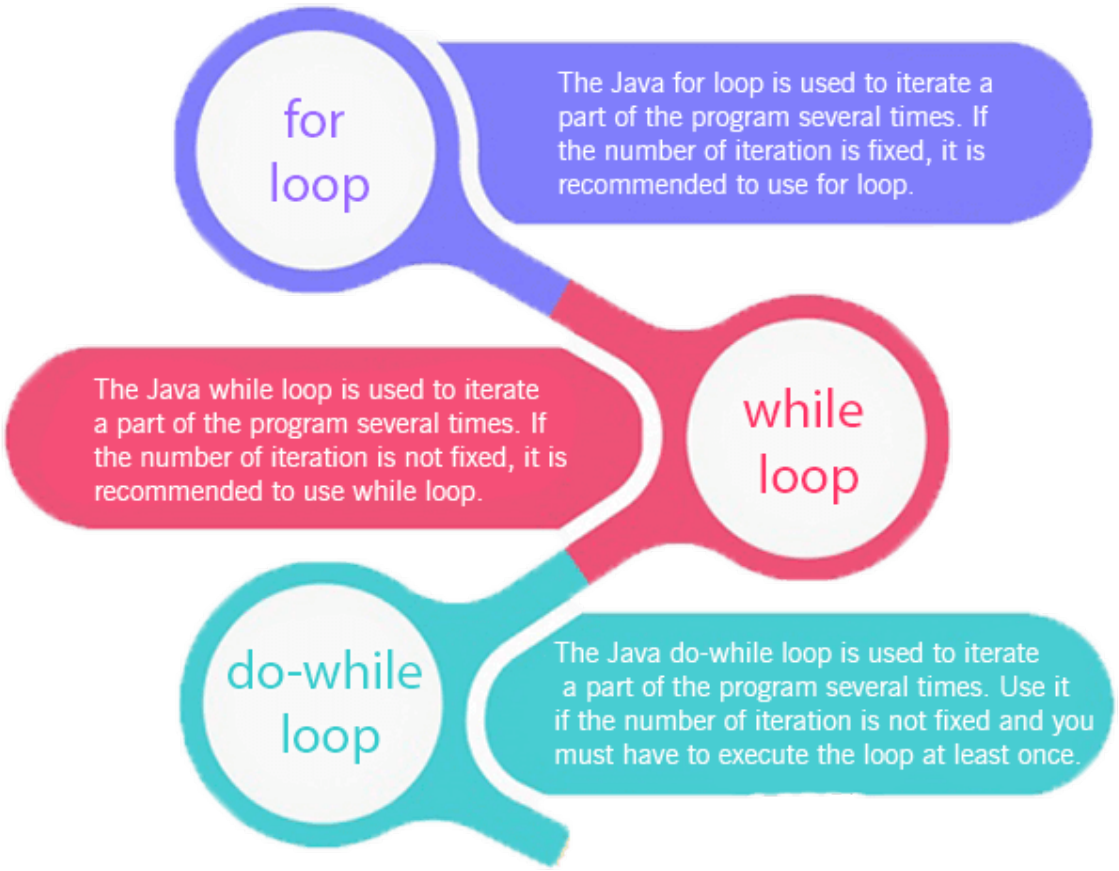
Java Loops

In programming languages, loops are used to execute a set of instructions/functions repeatedly when some conditions become true. There are three types of loops in Java.

- for loop
- while loop
- do-while loop



Java For Loop

The Java *for loop* is used to iterate a part of the program several times. If the number of iterations are fixed, it is recommended to use for loop.
There are two types of for loops in java.

- Simple For Loop
- For-each or Enhanced for loop

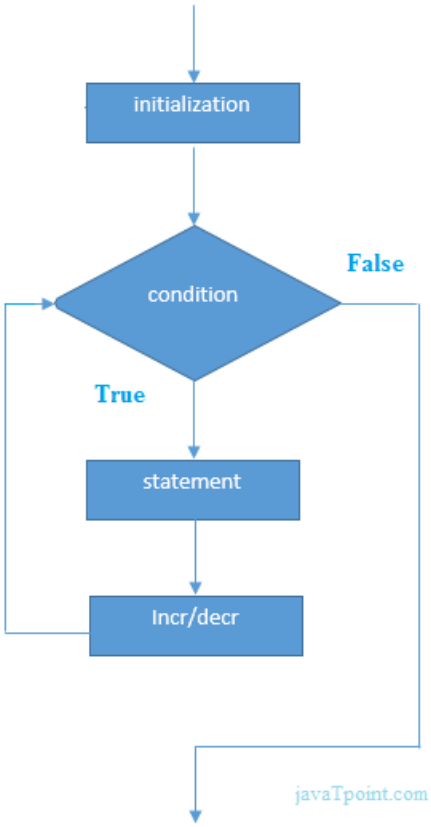
Java Simple For Loop

A simple for loop is the same as C/C++. We can initialize the variable, check condition and increment/decrement value. It consists of four parts:

1. **Initialization**: It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.
2. **Condition**: It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.
3. **Statement**: The statement of the loop is executed each time until the second condition is false.

```
for (initialization; condition; incr/decr){
    //statement or code to be executed
}
```

Flowchart :



Example :

```
for (int i = 0; i < 10; i++) {
  System.out.println(i);
}
```

Example : Odd numbers

```
for (int i = 1; i < 10; i+=2) {
  System.out.println(i);
}
```

//This program will print odd numbers from 1 to 10

Nested For Loop

If we have a for loop inside another loop, it is known as nested for loop. The inner loop executes completely whenever outer loop executes.

Example:

```
for (int i =1;i<4 ;i++ ) {
    for(int j = 3 ; j > 0; j--){
        System.out.print(i*j+ " ");
    }
}
```

Exercise: Lets draw a pyramid with nested for loops.

```
for (int i =1;i<=5 ;i++ ) {
    for(int j = 0 ; j < i; j++){
        System.out.print("*");
    }
    System.out.println();
}
```

Try to draw it upside down as well.

For-Each Loop

• There is also a "**for-each**" loop, which is used exclusively to loop through elements in an **array**:

Syntax :

```
for (type variableName : arrayName) {
    // code block to be executed
    }
```

Java Infinitive For Loop

If you use two semicolons ;; in the for loop, it will be infinitive for loop.

Example:

```
for(;;){
    System.out.println("Infinitive Loop");
}
```

'break' keyword

• You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch statement.

• The break statement can also be used to jump out of a **loop**.

• This example jumps out of the loop when i is equal to 4:

```
for (int i = 0; i < 10; i++) {
    if (i == 4)
    {
        break;
    }
    System.out.print(i+" ");
}

//Output 0 1 2 3
```

‘continue’ keyword

• The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

• This example skips the value of 4:

```
for (int i = 0; i < 10; i++) {
    if (i == 4) {
        continue;
    }
    System.out.println(i);
}
```

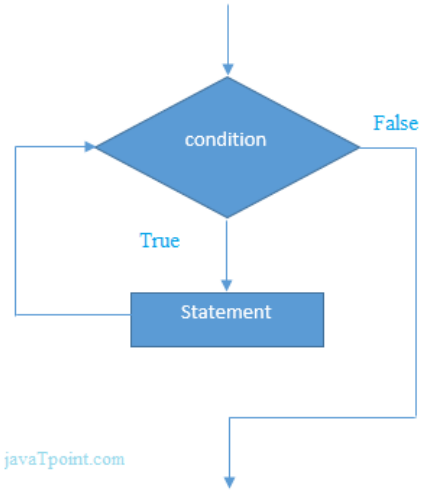
Java While Loop

The Java *while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

• The while loop loops through a block of code as long as a specified condition is true:

Syntax:

```
while (condition) {
// code block to be executed
}
```



In the example below, the code in the loop will run, over and over again, as long as a variable (i) is less than 5:

```
int i = 0;
while (i < 5) {
    System.out.println(i);
    i++;
}
```

• **Note:** Do not forget to increase the variable used in the condition, otherwise the loop will never end!

Java Infinitive While Loop

If you pass **true** in the while loop, it will be infinitive while loop.

Syntax:

```
while(true){
    //code to be executed
    }
```

Example:

```
while(true){
    System.out.println("This is a infinitive while loop");
}
```

'break' and 'continue' in While Loop

'break' example

```
int i = 0;
while (i < 10) {
    System.out.println(i);
    i++;
    if (i == 4) {
        break;
    }
}
```

'continue' example

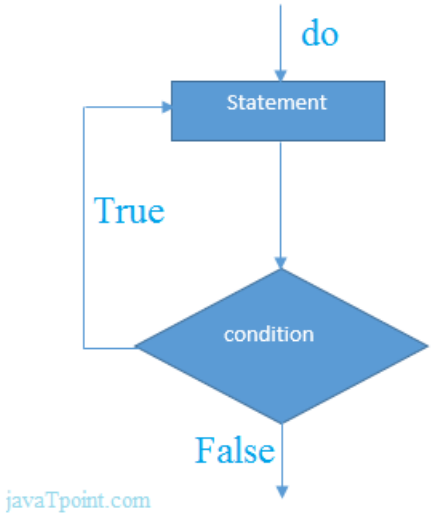
```
int i = 0;
while (i < 10) {
    if (i == 4) {
        i++;
        continue;
    }
    System.out.println(i);
    i++;
}
```

Exercise: Printing the sum all even numbers between 0 and 10.

```
int sum =0;
int i =0;
while (i<=10){
    i++;
    if (i%2 == 1) {
        continue;
    }
    sum += i;
}
System.out.println(sum);
```

The Do/While Loop

• The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.



Syntax:

```
do {
    // code block to be executed
}
while (condition);
```

The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

```
int i = 0;
do {
    System.out.print(i +" ");
    i++;
}
while (i < 5);

//output
// 1 2 3 4
```