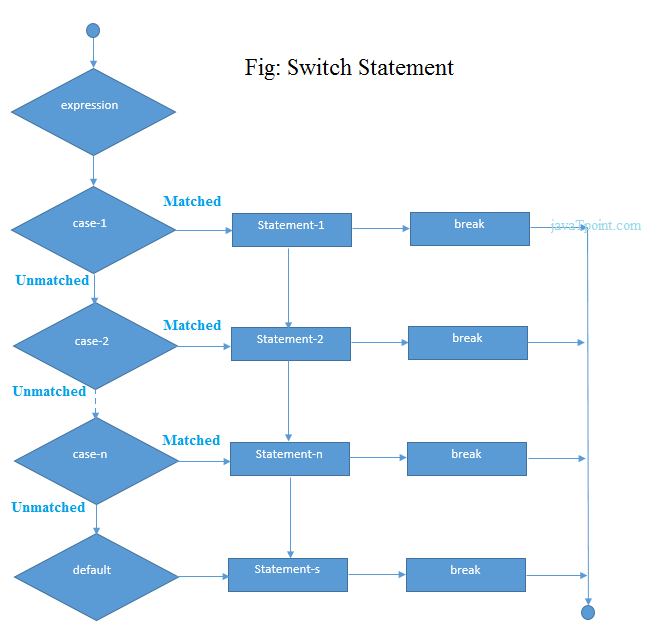
Java Switch Statement

The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement.

**Syntax:**

1. **switch**(expression){
2. **case** value1:
3. //code to be executed;
4. **break**;  //optional
5. **case** value2:
6. //code to be executed;
7. **break**;  //optional
8. ......
10. **default**:
11. code to be executed **if** all cases are not matched;
12. }



**Example:**

1. **public** **class** SwitchExample {
2. **public** **static** **void** main(String[] args) {
3. **int** number=20;
4. **switch**(number){
5. **case** 10: System.out.println("10");
6. **break**;
7. **case** 20: System.out.println("20");**break**;
8. **case** 30: System.out.println("30");**break**;
9. **default**:System.out.println("Not in 10, 20 or 30");
10. }
11. }
12. }

Output:

20

## Java Switch Statement is fall-through

The java switch statement is fall-through. It means it executes all statement after first match if break statement is not used with switch cases.

**Example:**

1. **public** **class** SwitchExample2 {
2. **public** **static** **void** main(String[] args) {
3. **int** number=20;
4. **switch**(number){
5. **case** 10: System.out.println("10");
6. **case** 20: System.out.println("20");
7. **case** 30: System.out.println("30");
8. **default**:System.out.println("Not in 10, 20 or 30");
9. }
10. }
11. }

Output:

20

30

Not in 10, 20 or 30

# Java For Loop

The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

There are three types of for loop in java.

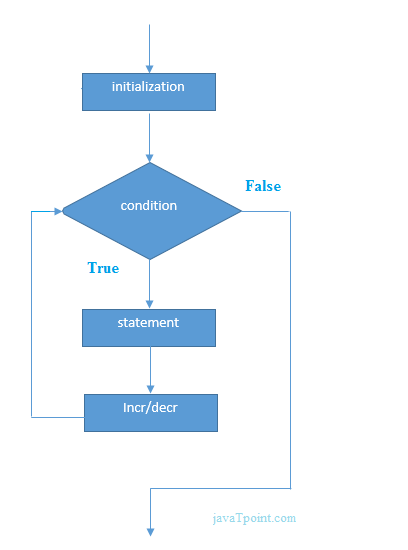
* Simple For Loop
* For-each or Enhanced For Loop
* Labeled For Loop

## Java Simple For Loop

The simple for loop is same as C/C++. We can initialize variable, check condition and increment/decrement value.

**Syntax:**

1. **for**(initialization;condition;incr/decr){
2. //code to be executed
3. }



**Example:**

1. **public** **class** ForExample {
2. **public** **static** **void** main(String[] args) {
3. **for**(**int** i=1;i<=10;i++){
4. System.out.println(i);
5. }
6. }
7. }

Output:

1

2

3

4

5

6

7

8

9

10

## Java For-each Loop

The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.

It works on elements basis not index. It returns element one by one in the defined variable.

**Syntax:**

1. **for**(Type var:array){
2. //code to be executed
3. }

**Example:**

1. **public** **class** ForEachExample {
2. **public** **static** **void** main(String[] args) {
3. **int** arr[]={12,23,44,56,78};
4. **for**(**int** i:arr){
5. System.out.println(i);
6. }
7. }
8. }

Output:

12

23

44

56

78

## Java Infinitive For Loop

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

**Syntax:**

1. **for**(;;){
2. //code to be executed
3. }

**Example:**

1. **public** **class** ForExample {
2. **public** **static** **void** main(String[] args) {
3. **for**(;;){
4. System.out.println("infinitive loop");
5. }
6. }
7. }

Output:

infinitive loop

infinitive loop

infinitive loop

infinitive loop

infinitive loop

ctrl+c

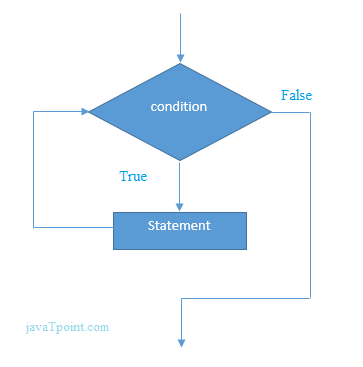
Now, you need to press ctrl+c to exit from the program.

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.

**Syntax:**

1. **while**(condition){
2. //code to be executed
3. }



**Example:**

1. **public** **class** WhileExample {
2. **public** **static** **void** main(String[] args) {
3. **int** i=1;
4. **while**(i<=10){
5. System.out.println(i);
6. i++;
7. }
8. }
9. }

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=WhileExample)

Output:

1

2

3

4

5

6

7

8

9

10

## Java Infinitive While Loop

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

**Syntax:**

1. **while**(**true**){
2. //code to be executed
3. }

**Example:**

1. **public** **class** WhileExample2 {
2. **public** **static** **void** main(String[] args) {
3. **while**(**true**){
4. System.out.println("infinitive while loop");
5. }
6. }
7. }

Output:

infinitive while loop

infinitive while loop

infinitive while loop

infinitive while loop

infinitive while loop

ctrl+c

Now, you need to press ctrl+c to exit from the program.

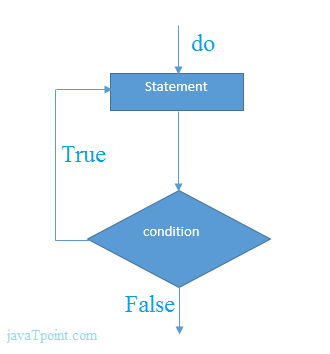
Java do-while Loop

The Java *do-while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.

The Java *do-while loop* is executed at least once because condition is checked after loop body.

**Syntax:**

1. **do**{
2. //code to be executed
3. }**while**(condition);



**Example:**

1. **public** **class** DoWhileExample {
2. **public** **static** **void** main(String[] args) {
3. **int** i=1;
4. **do**{
5. System.out.println(i);
6. i++;
7. }**while**(i<=10);
8. }
9. }

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=DoWhileExample)

Output:

1

2

3

4

5

6

7

8

9

10

## Java Infinitive do-while Loop

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

**Syntax:**

1. **do**{
2. //code to be executed
3. }**while**(**true**);

**Example:**

1. **public** **class** DoWhileExample2 {
2. **public** **static** **void** main(String[] args) {
3. **do**{
4. System.out.println("infinitive do while loop");
5. }**while**(**true**);
6. }
7. }

Output:

infinitive do while loop

infinitive do while loop

infinitive do while loop

ctrl+c

Now, you need to press ctrl+c to exit from the program.