**Java for Loop**

In computer programming, loops are used to repeat a block of code. For example, if you want to show a message 100 times, then rather than typing the same code 100 times, you can use a loop.

In Java, there are three types of loops.

* **for loop**
* [**while loop**](https://www.programiz.com/java-programming/do-while-loop#syntax-while)
* [**do...while loop**](https://www.programiz.com/java-programming/do-while-loop#do-while-loop)

**Java for Loop**

Java for loop is used to run a block of code for a certain number of times. The syntax of for loop is:

for (initialExpression; testExpression; updateExpression)

{

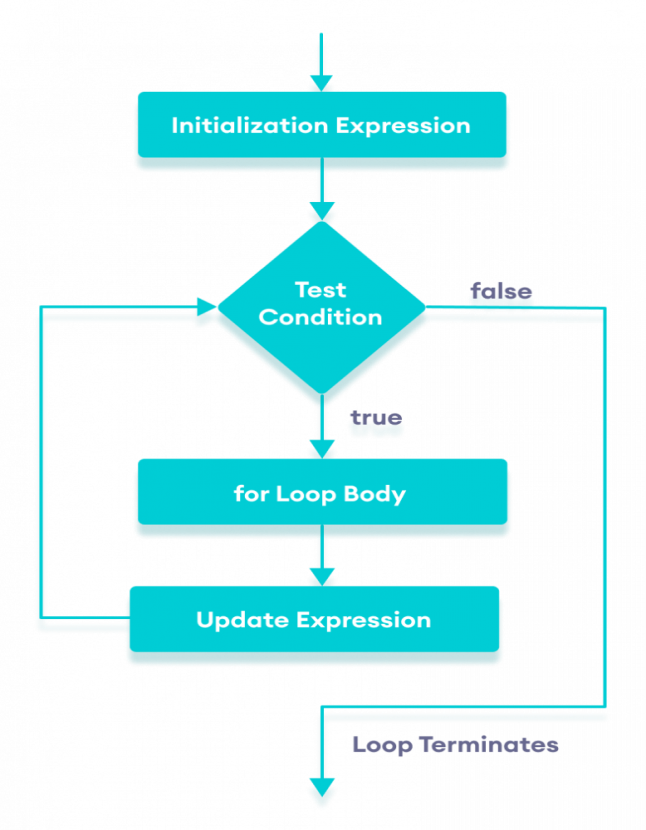
// body of the loop

}

Here,

1. The **initialExpression** initializes and/or declares variables and executes only once.
2. The **condition** is evaluated. If the **condition** is true, the body of the for loop is executed.
3. The **updateExpression** updates the value of **initialExpression**.
4. The **condition** is evaluated again. The process continues until the **condition** is false.

.



Flowchart of Java for loop

**Example 1: Display a Text Five Times**

// Program to print a text 5 times

class Main {

public static void main(String[] args) {

int n = 5;

// for loop

for (int i = 1; i <= n; ++i) {

System.out.println("Java is fun");

}

}

}

**Output**

Java is fun

Java is fun

Java is fun

Java is fun

Java is fun

Here is how this program works.

|  |  |  |  |
| --- | --- | --- | --- |
| Iteration | Variable | Condition: i <= n | Action |
| 1st | i = 1 n = 5 | true | Java is fun is printed. i is increased to **2**. |
| 2nd | i = 2 n = 5 | true | Java is fun is printed. i is increased to **3**. |
| 3rd | i = 3 n = 5 | true | Java is fun is printed. i is increased to **4**. |
| 4th | i = 4 n = 5 | true | Java is fun is printed. i is increased to **5**. |
| 5th | i = 5 n = 5 | true | Java is fun is printed. i is increased to **6**. |
| 6th | i = 6 n = 5 | false | The loop is terminated. |

**Example 2: Display numbers from 1 to 5**

// Program to print numbers from 1 to 5

class Main {

public static void main(String[] args) {

int n = 5;

// for loop

for (int i = 1; i <= n; ++i) {

System.out.println(i);

}

}

}

**Output**

1

2

3

4

5

Here is how the program works.

|  |  |  |  |
| --- | --- | --- | --- |
| Iteration | Variable | Condition: i <= n | Action |
| 1st | i = 1 n = 5 | true | 1 is printed. i is increased to **2**. |
| 2nd | i = 2 n = 5 | true | 2 is printed. i is increased to **3**. |
| 3rd | i = 3 n = 5 | true | 3 is printed. i is increased to **4**. |
| 4th | i = 4 n = 5 | true | 4 is printed. i is increased to **5**. |
| 5th | i = 5 n = 5 | true | 5 is printed. i is increased to **6**. |
| 6th | i = 6 n = 5 | false | The loop is terminated. |

**Example 3: Display Sum of n Natural Numbers**

class Main {

public static void main(String[] args) {

int sum = 0;

int n = 1000;

for (int i = 1; i <= n; ++i) {

// body inside for loop

sum += i; // sum = sum + i

}

System.out.println("Sum = " + sum);

}

}

**Output**:

Sum = 500500

Here, the value of sum is **0** initially. Then, the for loop is iterated from i = 1 to 1000. In each iteration, iis added to sum and its value is increased by **1**.

When i becomes **1001**, the test condition is false and sum will be equal to 0 + 1 + 2 + .... + 1000.

The above program to add the sum of natural numbers can also be written as

// Program to find the sum of natural numbers from 1 to 1000.

class Main {

public static void main(String[] args) {

int sum = 0;

int n = 1000;

// for loop

for (int i = n; i >= 1; --i) {

// body inside for loop

sum += i; // sum = sum + i

}

System.out.println("Sum = " + sum);

}

}

The output of this program is the same as the **Example 3**.

**Java for-each Loop**

The Java for loop has an alternative syntax that makes it easy to iterate through [arrays](https://www.programiz.com/java-programming/arrays) and collections. For example,

// print array elements

class Main {

public static void main(String[] args) {

// create an array

int[] numbers = {3, 7, 5, -5};

// iterating through the array

for (int number: numbers) {

System.out.println(number);

}

}

}

**Output**

3

7

5

-5

Here, we have used the **for-each loop** to print each element of the numbers array one by one.

In the first iteration of the loop, number will be 3, number will be 7 in second iteration and so on.

**Java Infinite for Loop**

If we set the **test expression** in such a way that it never evaluates to false, the for loop will run forever. This is called infinite for loop. For example,

class Infinite {

public static void main(String[] args) {

int sum = 0;

for (int i = 1; i <= 10; --i) {

System.out.println("Hello");

}

}

}

Here, the test expression ,i <= 10, is never false and Hello is printed repeatedly until the memory runs out.

**Nested For Loop**

**public** **class** PyramidExample

 {

**public** **static** **void** main(String[] args)

 {

**for**(**int** i=1;i<=5;i++)

{

**for**(**int** j=1;j<=i;j++)

         System.out.print(" \*  ");

System.out.println();//new line

}

}

}

**Output:**

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

**Pyramid Example 2:**

**PyramidExample2.java**

**public** **class** PyramidExample2

{

**public** **static** **void** main(String[] args)

 {

**int** term=6;

**for**(**int** i=1;i<=term;i++)

{

**for**(**int** j=term;j>=i;j--){

         System.out.print("\* ");

System.out.println();//new line

}

}

}

**Output:**

\* \* \* \* \* \*

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

## 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 the basis of elements and not the index. It returns element one by one in the defined variable.

**Syntax:**

**for**(data\_type variable : array\_name){

//code to be executed

}

**ForEachExample.java**

**public** **class** ForEachExample {

**public** **static** **void** main(String[] args) {

    //Declaring an array

**int** arr[]={12,23,44,56,78};

    //Printing array using for-each loop

**for**(**int** i:arr){

        System.out.println(i);

    }

}

}

**Output:**

12

23

44

56

78

## Java Labeled For Loop

We can have a name of each Java for loop. To do so, we use label before the for loop. It is useful while using the nested for loop as we can break/continue specific for loop.

#### Note: The break and continue keywords breaks or continues the innermost for loop respectively.

**Syntax:**

labelname:

**for**(initialization; condition; increment/decrement){

//code to be executed

}

**LabeledForExample.java**

**public** **class** LabeledForExample {

**public** **static** **void** main(String[] args) {

    //Using Label for outer and for loop

    aa:

**for**(**int** i=1;i<=3;i++){

            bb:

**for**(**int** j=1;j<=3;j++){

**if**(i==2&&j==2){

**break** aa;

                    }

                    System.out.println(i+" "+j);

                }

        }

}

}

**Output:**

1 1

1 2

1 3

2 1

If you use **break bb;**, it will break inner loop only which is the default behaviour of any loop.

**LabeledForExample2.java**

**public** **class** LabeledForExample2 {

**public** **static** **void** main(String[] args) {

    aa:

**for**(**int** i=1;i<=3;i++){

            bb:

**for**(**int** j=1;j<=3;j++){

**if**(i==2&&j==2){

**break** bb;

                    }

                    System.out.println(i+" "+j);

                }

        }

}

}

**Output:**

1 1

1 2

1 3

2 1

3 1

3 2

3 3

## Java Infinitive for Loop

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

**Syntax:**

**for**(;;){

//code to be executed

}

**Example:**

**public** **class** ForExample {

**public** **static** **void** main(String[] args) {

    //Using no condition in for loop

**for**(;;){

        System.out.println("infinitive loop");

    }

}

}

**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 for Loop vs while Loop vs do-while Loop

|  |  |  |  |
| --- | --- | --- | --- |
| **Comparison** | **for loop** | **while loop** | **do-while loop** |
| Introduction | The Java for loop is a control flow statement that iterates a part of the [programs](https://www.javatpoint.com/java-programs) multiple times. | The Java while loop is a control flow statement that executes a part of the programs repeatedly on the basis of given boolean condition. | The Java do while loop is a control flow statement that executes a part of the programs at least once and the further execution depends upon the given boolean condition. |
| When to use | If the number of iteration is fixed, it is recommended to use for loop. | If the number of iteration is not fixed, it is recommended to use while loop. | If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop. |
| Syntax | for(init;condition;incr/decr){ // code to be executed } | while(condition){ //code to be executed } | do{ //code to be executed }while(condition); |
| Example | //for loop for(int i=1;i<=10;i++){ System.out.println(i); } | //while loop int i=1; while(i<=10){ System.out.println(i); i++; } | //do-while loop int i=1; do{ System.out.println(i); i++; }while(i<=10); |
| Syntax for infinitive loop | for(;;){ //code to be executed } | while(true){ //code to be executed } | do{ //code to be executed }while(true); |