**Example: Java switch Statement**

// Java Program to check the size

// using the switch...case statement

class Main {

public static void main(String[] args) {

int number = 44;

String size;

// switch statement to check size

switch (number) {

case 29:

size = "Small";

break;

case 42:

size = "Medium";

break;

// match the value of week

case 44:

size = "Large";

break;

case 48:

size = "Extra Large";

break;

default:

size = "Unknown";

break;

}

System.out.println("Size: " + size);

}

}

### **Nested if Syntax**

if(condition\_1) {

Statement1(s);

if(condition\_2) {

Statement2(s);

}

}

In this program, we will see the implementation of nested if statements in java.

### **Algorithm:**

1. Start
2. Create an instance of the Scanner class.
3. Declare a variable to store the number.
4. Ask the user to initialize the number.
5. Use the first if statement to check if the number is lesser than 100.
6. Use the inner if statement to check if the number is greater than 50.
7. If the number is greater than 50 and lesser than 100, then print the message that the entered number is greater than 50 and lesser than 100.
8. If the number is not greater than 50 but is lesser than 100, then print the message that the entered number is lesser than 100.
9. If the number is greater than 100 then print the message that the entered number is greater than 100.
10. Stop.

Below is the Java code example for nested if-else.

//Java nested if Program

import java.util.Scanner;

public class Main

{

public static void main(String []args)

{

//Take input from the user

//Create an instance of the Scanner class

Scanner sc=new Scanner(System.in);

System.out.println("Enter a number: ");

int num=sc.nextInt();

if( num < 100 )

{

System.out.println("The entered number is less than 100");

if(num > 50)

{

System.out.println("The entered number is greater than 50");

}

}

else

{

System.out.println("The entered number is greater than 100");

}

}

}

## Program 2: Java Nested if-else Program

In this program, we will see the implementation of the nested if-else statements in a java program.

### **Algorithm:**

1. Start
2. Create an instance of the Scanner class.
3. Declare two variables to store the age and weight of a person.
4. Ask the user to initialize the age and weight.
5. Use the first if statement to check if the person is above 18 years of age.
6. If the person is above 18 years of age then use another if statement to check if the weight of the person is above 50 or not.
7. If the person's age is above 18 and weight is also above 50 then, print the message that the person is eligible to donate blood.
8. If the person is above 18 years of age but his/her weight is below 50 then print the message that the person is not eligible to donate blood.
9. If the person is below 18 years of age, then print the message that the age must be greater than 18.
10. Display the result.
11. Stop

//Java nested if-else Program

import java.util.Scanner;

public class Main

{

public static void main(String []args)

{

//Take input from the user

//Create an instance of the Scanner class

Scanner sc=new Scanner(System.in);

System.out.println("Enter the age: ");

int age=sc.nextInt();

System.out.println("Enter the weight: ");

int weight=sc.nextInt();

if(age>=18)

{

if(weight>50)

{

System.out.println("The person is eligible to donate blood");

}

else

{

System.out.println("The person is not eligible to donate blood");

}

}

else

{

System.out.println("Age must be greater than 18");

}

}

}

# **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:

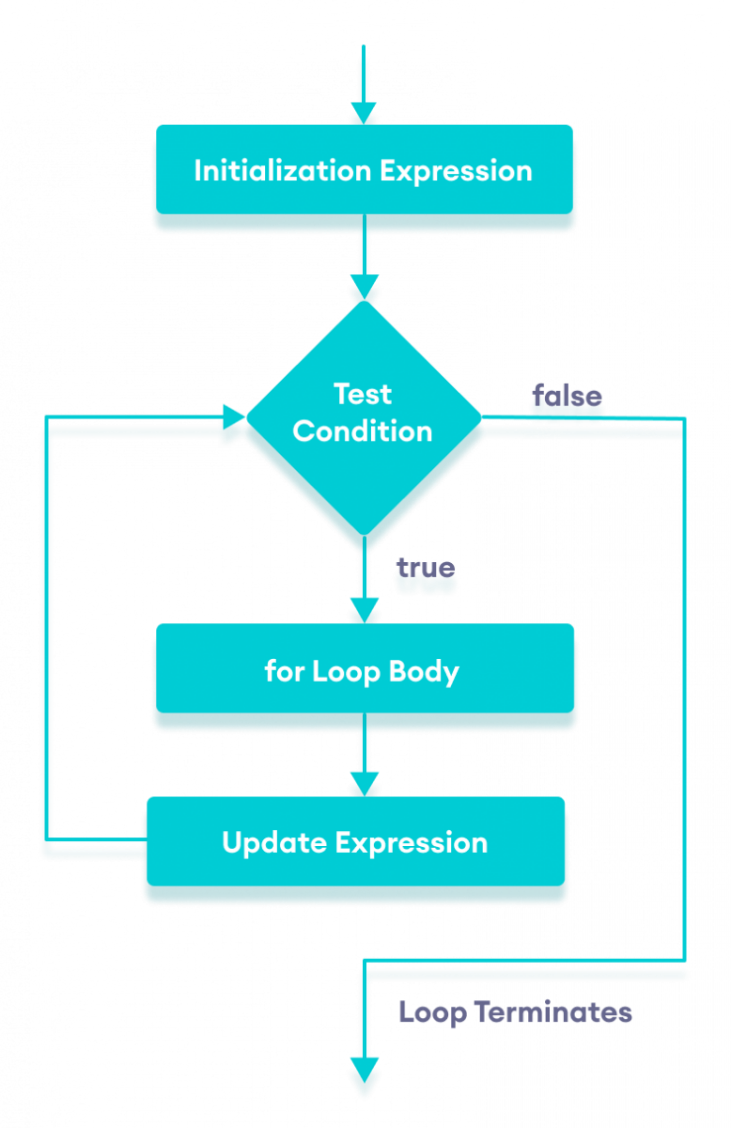
for (initialExpression; testExpression; updateExpression) {

// 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

### 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);

}

}

}

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

// 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 = 1; i <= n; ++i) {

// body inside for loop

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

}

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

}

}

### Example 2: Java for loop inside the while loop

class Main {

public static void main(String[] args) {

int weeks = 3;

int days = 7;

int i = 1;

// outer loop

while (i <= weeks) {

System.out.println("Week: " + i);

// inner loop

for (int j = 1; j <= days; ++j) {

System.out.println(" Days: " + j);

}

++i;

}

}

}

**Output**:

Week: 1

Day: 1

Day: 2

Day: 3

.... .. ....

Week: 2

Day: 1

Day: 2

Day: 3

.... .. ....

.... .. ....

Here you can see that the output of both **Example 1** and **Example 2** is the same.

### **Example 3: Java nested loops to create a pattern**

We can use the nested loop in Java to create patterns like full pyramid, half pyramid, inverted pyramid, and so on.

Here is a program to create a half pyramid pattern using nested loops.

class Main {

public static void main(String[] args) {

int rows = 5;

// outer loop

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

// inner loop to print the numbers

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

System.out.print(j + " ");

}

System.out.println("");

}

}

}

**Output**

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

## break and continue Inside Nested Loops

When we use a break statement inside the inner loop, it terminates the inner loop but not the outer loop. For example,

class Main {

public static void main(String[] args) {

int weeks = 3;

int days = 7;

// outer loop

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

System.out.println("Week: " + i);

// inner loop

for(int j = 1; j <= days; ++j) {

// break inside the inner loop

if(i == 2) {

break;

}

System.out.println(" Days: " + j);

}

}

}

}

**Output**

Week: 1

Day: 1

Day: 2

.... .. ....

Week: 2

Week: 3

Day: 1

Day: 2

.... .. ....

.... .. ....

## Java Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with **square brackets**:

String[] cars;

We have now declared a variable that holds an array of strings. To insert values to it, we can use an array literal - place the values in a comma-separated list, inside curly braces:

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

To create an array of integers, you could write:

int[] myNum = {10, 20, 30, 40};

## Access the Elements of an Array

You access an array element by referring to the index number.

This statement accesses the value of the first element in cars:

### **Example**

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

System.out.println(cars[0]);

// Outputs Volvo

## Change an Array Element

To change the value of a specific element, refer to the index number:

### **Example**

cars[0] = "Opel";

### **Example**

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

cars[0] = "Opel";

System.out.println(cars[0]);

// Now outputs Opel instead of Volvo

## Array Length

To find out how many elements an array has, use the length property:

### **Example**

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

System.out.println(cars.length);

// Outputs 4

## Loop Through an Array

You can loop through the array elements with the for loop, and use the length property to specify how many times the loop should run.

The following example outputs all elements in the **cars** array:

### **Example**

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

for (int i = 0; i < cars.length; i++) {

System.out.println(cars[i]);

}

## Loop Through an Array with For-Each

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

### **Syntax**

for (type variable : arrayname) {

...

}

The following example outputs all elements in the **cars** array, using a "**for-each**" loop:

### **Example**

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

for (String i : cars) {

System.out.println(i);

}

The example above can be read like this: **for each** String element (called **i** - as in **i**ndex) in **cars**, print out the value of**i**.

If you compare the for loop and **for-each** loop, you will see that the **for-each** method is easier to write, it does not require a counter (using the length property), and it is more readable.