**Table of Contents**

[Java Program to Check Leap Year 2](#_Toc92044969)

[Example: Java Program to Check a Leap Year 2](#_Toc92044970)

[Java Program to Check Whether a Number is Positive or Negative 4](#_Toc92044971)

[Example: Check if a Number is Positive or Negative using if else 4](#_Toc92044972)

[Java Program to Calculate the Sum of Natural Numbers 6](#_Toc92044973)

[Example 1: Sum of Natural Numbers using for loop 6](#_Toc92044974)

[Example 2: Sum of Natural Numbers using while loop 7](#_Toc92044975)

[Java Program to Generate Multiplication Table 8](#_Toc92044976)

[Example 1: Generate Multiplication Table using for loop 8](#_Toc92044977)

[Example 2: Generate Multiplication Table using while loop 9](#_Toc92044978)

[Java Program to Find GCD of two Numbers 10](#_Toc92044979)

[Example 1: Find GCD of two numbers using for loop and if statement 10](#_Toc92044980)

[Example 2: Find GCD of two numbers using while loop and if else statement 11](#_Toc92044981)

[Example 3: GCD for both positive and negative numbers 12](#_Toc92044982)

[Java Program to Find Factorial of a Number 13](#_Toc92044983)

[Example 1: Find Factorial of a number using for loop 13](#_Toc92044984)

[Example 2: Find Factorial of a number using BigInteger 14](#_Toc92044985)

[Example 3: Find Factorial of a number using while loop 15](#_Toc92044986)

[Java Program to Display Fibonacci Series 16](#_Toc92044987)

[Display Fibonacci Series 16](#_Toc92044988)

[Example: Display Fibonacci Series Using for Loop 17](#_Toc92044989)

[Example 2: Display Fibonacci series using while loop 17](#_Toc92044990)

[Example 3: Display Fibonacci series up to a given number 18](#_Toc92044991)

**Java Program to Check Leap Year**

In this program, you'll learn to check if the given year is a leap year or not. This is checked using a if else statement.

To understand this example, you should have the knowledge of the following [Java programming](https://www.programiz.com/java-programming) topics:

* [Java Operators](https://www.programiz.com/java-programming/operators)
* [Java if...else Statement](https://www.programiz.com/java-programming/if-else-statement)

A leap year is exactly divisible by 4 except for century years (years ending with 00). The century year is a leap year only if it is perfectly divisible by 400.

**Example: Java Program to Check a Leap Year**

public class Main {

public static void main(String[] args) {

// year to be checked

int year = 1996;

boolean leap = false;

// if the year is divided by 4

if (year % 4 == 0) {

// if the year is century

if (year % 100 == 0) {

// if year is divided by 400

// then it is a leap year

if (year % 400 == 0)

leap = true;

else

leap = false;

}

// if the year is not century

else

leap = true;

}

else

leap = false;

if (leap)

System.out.println(year + " is a leap year.");

else

System.out.println(year + " is not a leap year.");

}

}

**Output**

1900 is not a leap year.

In the above example, we are checking if the year 1900 is a leap year or not. Since 1900 is a century year (ending with 00), it should be divisible by both **4** and **400** to be a leap year.

However, 1900 is not divisible by 400. Hence, it is not a leap year.

Now, let's change the year to 2012. The output will be

2012 is a leap year.

Here, 2012 is not a century year. Hence, to be a leap year, it needs to be divisible by only **4**.

Since 2012 is divisible by 4, it is a leap year.

**Java Program to Check Whether a Number is Positive or Negative**

In this program, you'll learn to check whether a given number is positive or negative. This is done by using a if else statement in Java.

To understand this example, you should have the knowledge of the following [Java programming](https://www.programiz.com/java-programming) topics:

* [Java if...else Statement](https://www.programiz.com/java-programming/if-else-statement)
* [Java Operators](https://www.programiz.com/java-programming/operators)

**Example: Check if a Number is Positive or Negative using if else**

public class PositiveNegative {

public static void main(String[] args) {

double number = 12.3;

// true if number is less than 0

if (number < 0.0)

System.out.println(number + " is a negative number.");

// true if number is greater than 0

else if ( number > 0.0)

System.out.println(number + " is a positive number.");

// if both test expression is evaluated to false

else

System.out.println(number + " is 0.");

}

}

**Output**

12.3 is a positive number.

If you change the value of number to a negative number (say -12.3), the output will be:

-12.3 is a negative number.

In the above program, it is quite clear how the variable number is checked to be positive or negative, by comparing it to 0.

If you're not sure, here is the breakdown:

* If a number is greater than zero, it is a positive number.
* If a number is less than zero, it is a negative number.
* If a number equals to zero, it is zero.

**Java Program to Calculate the Sum of Natural Numbers**

In this program, you'll learn to calculate the sum of natural numbers using for loop and while loop in Java.

To understand this example, you should have the knowledge of the following [Java programming](https://www.programiz.com/java-programming) topics:

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

The positive numbers 1, 2, 3... are known as natural numbers and its sum is the result of all numbers starting from 1 to the given number.

For n, the sum of natural numbers is:

1 + 2 + 3 + ... + n

**Example 1: Sum of Natural Numbers using for loop**

public class SumNatural {

public static void main(String[] args) {

int num = 100, sum = 0;

for(int i = 1; i <= num; ++i)

{

// sum = sum + i;

sum += i;

}

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

}

}

**Output**

Sum = 5050

The above program loops from 1 to the given num(100) and adds all numbers to the variable sum.

You can solve this problem using a while loop as follows:

**Example 2: Sum of Natural Numbers using while loop**

public class SumNatural {

public static void main(String[] args) {

int num = 50, i = 1, sum = 0;

while(i <= num)

{

sum += i;

i++;

}

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

}

}

**Output**

Sum = 1275

In the above program, unlike a for loop, we have to increment the value of i inside the body of the loop.

Though both programs are technically correct, it is better to use for loop in this case. It's because the number of iteration (up to num) is known.

Visit this page to learn *how to find the sum of natural numbers using recursion*.

**Java Program to Generate Multiplication Table**

In this program, you'll learn to generate multiplication table of a given number. This is done by using a for and a while loop in Java.

To understand this example, you should have the knowledge of the following [Java programming](https://www.programiz.com/java-programming) topics:

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

**Example 1: Generate Multiplication Table using for loop**

public class MultiplicationTable {

public static void main(String[] args) {

int num = 5;

for(int i = 1; i <= 10; ++i)

{

System.out.printf("%d \* %d = %d \n", num, i, num \* i);

}

}

}

**Output**

5 \* 1 = 5

5 \* 2 = 10

5 \* 3 = 15

5 \* 4 = 20

5 \* 5 = 25

5 \* 6 = 30

5 \* 7 = 35

5 \* 8 = 40

5 \* 9 = 45

5 \* 10 = 50

The same multiplication table can also be generated using a while loop in Java.

**Example 2: Generate Multiplication Table using while loop**

public class MultiplicationTable {

public static void main(String[] args) {

int num = 9, i = 1;

while(i <= 10)

{

System.out.printf("%d \* %d = %d \n", num, i, num \* i);

i++;

}

}

}

**Output**

9 \* 1 = 9

9 \* 2 = 18

9 \* 3 = 27

9 \* 4 = 36

9 \* 5 = 45

9 \* 6 = 54

9 \* 7 = 63

9 \* 8 = 72

9 \* 9 = 81

9 \* 10 = 90

In the above program, unlike a for loop, we have to increment the value of i inside the body of the loop.

Though both programs are technically correct, it is better to use for loop in this case. It's because the number of iteration (from 1 to 10) is known.

**Java Program to Find GCD of two Numbers**

In this program, you'll learn to find GCD of two numbers in Java. This is done by using for and while loops with the help of if else statements.

To understand this example, you should have the knowledge of the following [Java programming](https://www.programiz.com/java-programming) topics:

* [Java if...else Statement](https://www.programiz.com/java-programming/if-else-statement)
* [Java for Loop](https://www.programiz.com/java-programming/for-loop)
* [Java while Loop](https://www.programiz.com/java-programming/do-while-loop)

The HCF or GCD of two integers is the largest integer that can exactly divide both numbers (without a remainder).

**Example 1: Find GCD of two numbers using for loop and if statement**

class Main {

public static void main(String[] args) {

// find GCD between n1 and n2

int n1 = 81, n2 = 153;

// initially set to gcd

int gcd = 1;

for (int i = 1; i <= n1 && i <= n2; ++i) {

// check if i perfectly divides both n1 and n2

if (n1 % i == 0 && n2 % i == 0)

gcd = i;

}

System.out.println("GCD of " + n1 +" and " + n2 + " is " + gcd);

}

}

**Output**

GCD of 81 and 153 is 9

Here, two numbers whose GCD are to be found are stored in n1 and n2 respectively.

Then, a for loop is executed until i is less than both n1 and n2. This way, all numbers between 1 and smallest of the two numbers are iterated to find the GCD.

If both n1 and n2 are divisble by i, gcd is set to the number. This goes on until it finds the largest number (GCD) which divides both n1 and n2 without remainder.

We can also solve this problem using a while loop as follows:

**Example 2: Find GCD of two numbers using while loop and if else statement**

class Main {

public static void main(String[] args) {

// find GCD between n1 and n2

int n1 = 81, n2 = 153;

while(n1 != n2) {

if(n1 > n2) {

n1 -= n2;

}

else {

n2 -= n1;

}

}

System.out.println("GCD: " + n1);

}

}

**Output**

GCD: 9

This is a better way to find the GCD. In this method, smaller integer is subtracted from the larger integer, and the result is assigned to the variable holding larger integer. This process is continued until n1 and n2 are equal.

The above two programs works as intended only if the user enters positive integers. Here's a little modification of the second example to find the GCD for both positive and negative integers.

**Example 3: GCD for both positive and negative numbers**

class GCD {

public static void main(String[] args) {

int n1 = 81, n2 = -153;

// Always set to positive

n1 = ( n1 > 0) ? n1 : -n1;

n2 = ( n2 > 0) ? n2 : -n2;

while(n1 != n2) {

if(n1 > n2) {

n1 -= n2;

}

else {

n2 -= n1;

}

}

System.out.println("GCD: " + n1);

}

}

**Output**

GCD: 9

**Java Program to Find Factorial of a Number**

In this program, you'll learn to find the factorial of a number using for and while loop in Java.

To understand this example, you should have the knowledge of the following [Java programming](https://www.programiz.com/java-programming) topics:

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

The factorial of a positive number n is given by:

factorial of n (n!) = 1 \* 2 \* 3 \* 4 \* ... \* n

**Example 1: Find Factorial of a number using for loop**

public class Factorial {

public static void main(String[] args) {

int num = 10;

long factorial = 1;

for(int i = 1; i <= num; ++i)

{

// factorial = factorial \* i;

factorial \*= i;

}

System.out.printf("Factorial of %d = %d", num, factorial);

}

}

**Output**

Factorial of 10 = 3628800

In this program, we've used for loop to loop through all numbers between 1 and the given number num (10), and the product of each number till num is stored in a variable factorial.

We've used long instead of int to store large results of factorial. However, it's still not big enough to store the value of bigger numbers (say 100).

For results that cannot be stored in a long variable, we use BigInteger variable declared in java.math library.

**Example 2: Find Factorial of a number using BigInteger**

import java.math.BigInteger;

public class Factorial {

public static void main(String[] args) {

int num = 30;

BigInteger factorial = BigInteger.ONE;

for(int i = 1; i <= num; ++i)

{

// factorial = factorial \* i;

factorial = factorial.multiply(BigInteger.valueOf(i));

}

System.out.printf("Factorial of %d = %d", num, factorial);

}

}

**Output**

Factorial of 30 = 265252859812191058636308480000000

Here, instead of long, we use BigInteger variable factorial.

Since, \* cannot be used with BigInteger, we instead use multiply() for the product. Also, num should be casted to BigInteger for multiplication.

Likewise, we can also use a while loop to solve this problem.

**Example 3: Find Factorial of a number using while loop**

public class Factorial {

public static void main(String[] args) {

int num = 5, i = 1;

long factorial = 1;

while(i <= num)

{

factorial \*= i;

i++;

}

System.out.printf("Factorial of %d = %d", num, factorial);

}

}

**Output**

Factorial of 5 = 120

In the above program, unlike a for loop, we have to increment the value of i inside the body of the loop.

Though both programs are technically correct, it is better to use for loop in this case. It's because the number of iteration (upto num) is known.

Visit this page to learn to *find factorial of a number using recursion*.

# Java Program to Display Fibonacci Series

In this program, you'll learn to display the Fibonacci series in Java using for and while loops.

To understand this example, you should have the knowledge of the following [Java programming](https://www.programiz.com/java-programming) topics:

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

## Display Fibonacci Series

The Fibonacci series is a series where the next term is the sum of the previous two terms. The first two terms of the Fibonacci sequence are **0** followed by **1**.

**Fibonacci Series**: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34

Suppose, our first two terms are:

firstTerm = 0

secondTerm = 1

The next terms in the Fibonacci series would be calculated as:

nextTerm = firstTerm + secondTerm; (0 + 1)

firstTerm = secondTerm; (1)

secondTerm = nextTerm; (1)

nextTerm = firstTerm + secondTerm; (1 + 1)

....

Let's now apply this logic in our program.

### Example: Display Fibonacci Series Using for Loop

class Main {

public static void main(String[] args) {

int n = 10, firstTerm = 0, secondTerm = 1;

System.out.println("Fibonacci Series till " + n + " terms:");

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

System.out.print(firstTerm + ", ");

// compute the next term

int nextTerm = firstTerm + secondTerm;

firstTerm = secondTerm;

secondTerm = nextTerm;

}

}

}

**Output**

Fibonacci Series till 10 terms:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34,

In the above program, firstTerm and secondTerm are initialized with **0** and **1** respectively (first two digits of Fibonacci series).

Here, we have used the for loop to

* print the firstTerm of the series
* compute nextTerm by adding firstTerm and secondTerm
* assign value of secondTerm to firstTerm and nextTerm to secondTerm

We can also use a while loop to generate the Fibonacci series in Java.

### Example 2: Display Fibonacci series using while loop

class Main {

public static void main(String[] args) {

int i = 1, n = 10, firstTerm = 0, secondTerm = 1;

System.out.println("Fibonacci Series till " + n + " terms:");

while (i <= n) {

System.out.print(firstTerm + ", ");

int nextTerm = firstTerm + secondTerm;

firstTerm = secondTerm;

secondTerm = nextTerm;

i++;

}

}

}

**Output**

Fibonacci Series till 10 terms:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34,

The working of this program is the same as the previous program.

And, though both programs are technically correct, it is better to use a for loop in this case. It's because the number of iterations (from **1** to n) is known.

### Example 3: Display Fibonacci series up to a given number

class Fibonacci {

public static void main(String[] args) {

int n = 100, firstTerm = 0, secondTerm = 1;

System.out.println("Fibonacci Series Upto " + n + ": ");

while (firstTerm <= n) {

System.out.print(firstTerm + ", ");

int nextTerm = firstTerm + secondTerm;

firstTerm = secondTerm;

secondTerm = nextTerm;

}

}

}

**Output**

Fibonacci Series Upto 100:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,

In this example, instead of displaying the Fibonacci series of a certain number, we are displaying the series up to the given number (**100**).

For this, we just need to compare the firstTerm with n. And, if firstTerm is less than n, it is printed in the series. Else, the series is completed.