**Java Program Flow and Considerations: EEL 2880: Fall 2018 Subbarao Wunnava 12 05 2019**

[**www.careerbless.com/samplecodes/java/**](http://www.careerbless.com/samplecodes/java/) **Modified Fall 2019 12 05 2019**

[**www.vogella.com/tutorials/JavaIntroduction**](http://www.vogella.com/tutorials/JavaIntroduction)

## 1. Introduction to Java

### 1.1. History

Java is a programming language created by James Gosling from Sun Microsystems (Sun) in 1991. The first publicly available version of Java (Java 1.0) was released in 1995.

Sun Microsystems was acquired by the Oracle Corporation in 2010. Oracle has now the seamanship for Java.

Over time new enhanced versions of Java have been released. The current version of Java is Java 1.7 which is also known as Java 7.

From the Java programming language the Java platform evolved. The Java platform allows software developers to write program code in other languages than the Java programming language which still runs on the Java virtual machine. The Java platform is usually associated with the Java virtual machine and the Java core libraries.

### 1.2. Java and Open Source

In 2006 Sun started to make Java available under the GNU General Public License (GPL). Oracle continues this project called OpenJDK. As of 2018, Java virtual machine and the Java core libraries. Are available for public.

### 1.3. Java Virtual machine

The Java virtual machine (JVM) is a software implementation of a computer that executes programs like a real machine. The Java virtual machine is written specifically for a specific operating system, e.g., for Linux a special implementation is required as well as for Windows.

### Java Program: Print to screen

|  |  |
| --- | --- |
| 12345678 | public class FirstJavaProgram{    public static void main(String[] args)    {        //Displaying a message as output        System.out.println("Output of my first Java Program");    }} |

****

1. **Java functional Program for basic arithmetic operations:**

0 import java.io.InputStreamReader;

|  |  |
| --- | --- |
| 123456789101112131415161718192021222324252627282930313233343536 | /\*This Java program performs  basic arithmetic operations \*addition, subtraction, multiplication and division \*/public class BasicArithmeticDemo{     public static void main(String[] args)    {        int number1 = 10;        int number2 = 5;         //calculating number1 + number2;        int sum = number1 + number2;         //calculating number1 - number2;        int difference = number1 - number2;         //calculating number1 \* number2;        int product = number1 \* number2;         //calculating number1 / number2;        int quot = number1 / number2;         //calculating number1 % number2;        int rem = number1 % number2;         //Displaying the values        System.out.println("number1 : "+number1);        System.out.println("number2 : "+number2);        System.out.println("sum : "+sum);        System.out.println("difference : "+difference);        System.out.println("product : "+product);        System.out.println("quot : "+quot);        System.out.println("rem : "+rem);    }} |



## Java program which converts a decimal number to binary number

|  |  |
| --- | --- |
| 123456789101112131415161718192021222324 | import java.io.BufferedReader;import java.io.IOException;import java.io.InputStreamReader;/\*This program converts a decimal number to binary \*/public class DecimalToBinaryConvertor{    public static void main(String[] args)  throws IOException    {         //Get the decimal number which needs to be converted to binary         BufferedReader br = new BufferedReader(new . . …………InputStreamReader(System.in));        System.out.println("Enter the decimal number");        String numberString = br.readLine();        int number = Integer.parseInt(numberString);         //Converts the decimal number to binary        String binaryNumber = Integer.toBinaryString(number);         //Display the values        System.out.println("Entered decimal number : "+number);        System.out.println("Corresponding binary number : "+binaryNumber);    }// end main }// end public class |
|  |  |



1. **Java program which generates a random number**

|  |  |
| --- | --- |
| 123456789101112131415 | /\* \* This Java program generates a random number \*/public class RandomNumberGenerator{     public static void main(String args[])    {        //Generates a random number        double randomNumber = Math.random();         //Displays the random number        System.out.println(randomNumber);    }// end main()}// end public class |



**Java bytecode**

**Java bytecode** is the instruction set of the **Java** virtual machine. Each **bytecode** is composed by one, or in some cases two, bytes that represent the instruction (opcode), along with zero or more bytes for passing parameters.

A [Java](http://en.wikipedia.org/wiki/Java_%28programming_language%29) programmer does not need to be aware of or understand Java bytecode at all. However, as suggested in the [IBM](http://en.wikipedia.org/wiki/IBM) developerWorks journal, "Understanding bytecode and what bytecode is likely to be generated by a [Java compiler](http://en.wikipedia.org/wiki/Java_compiler) helps the Java programmer in the same way that knowledge of [assembly](http://en.wikipedia.org/wiki/Assembly_Language) helps the [C](http://en.wikipedia.org/wiki/C_%28programming_language%29) or [C++](http://en.wikipedia.org/wiki/C%2B%2B) programmer."[[2]](http://en.wikipedia.org/wiki/Java_bytecode#cite_note-2)

Instructions fall into a number of broad groups:

* Load and store (e.g. aload\_0, istore)
* Arithmetic and logic (e.g. ladd, fcmpl)
* Type conversion (e.g. i2b, d2i)
* Object creation and manipulation (new, putfield)
* Operand stack management (e.g. swap, dup2)
* Control transfer (e.g. ifeq, goto)
* Method invocation and return (e.g. invokespecial, areturn)

There are also a few instructions for a number of more specialized tasks such as exception throwing, synchronization, etc.

For example, "iadd" will add two integers, while "dadd" will add two doubles. The "const", "load", and "store" instructions may also take a suffix of the form "\_*n*", where *n* is a number from 0–3 for "load" and "store". The maximum *n* for "const" differs by type.

The "const" instructions push a value of the specified type onto the stack. For example "iconst\_5" will push an integer 5, while "dconst\_1" will push a double 1. There is also an "aconst\_null", which pushes "null". The *n* for the "load" and "store" instructions specifies the location in the variable table[*[clarification needed](http://en.wikipedia.org/wiki/Wikipedia%3APlease_clarify%22%20%5Co%20%22Wikipedia%3APlease%20clarify)*] to load from or store to. The "aload\_0" instruction pushes the object in variable 0 onto the stack (this is usually the "this" object). "istore\_1" stores the integer on the top of the stack into variable 1. For variables with higher numbers the suffix is dropped and operands must be used.

## Example

Consider the following Java code:

outer:

for (int i = 2; i < 1000; i++) {

 for (int j = 2; j < i; j++) {

 if (i % j == 0)

 continue outer;

 }

 System.out.println (i);

}