

Unit I Part A Basics of Java Programming

What is Java

Java was originally developed by James Gosling at Sun Microsystems (which has since been acquired by Oracle) and released in 1995 as a core component of Sun Microsystems' Java platform.

Java is an object-oriented, class-based, concurrent, secured and general-purpose computer-programming language. It is a widely used robust technology.

Java is a high level, robust, object-oriented and secure programming language.

Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

Application: According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.
2. Web Applications such as irctc.co.in etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games, etc.

Types of Java Applications

There are mainly 4 types of applications that can be created using Java programming:

1) Standalone Application

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates a dynamic page is called a web application. Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

3) Enterprise Application

An application that is distributed in nature, such as banking applications, etc. is called enterprise application. It has advantages of the high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

4) Mobile Application

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

Java Platforms / Editions

There are 4 platforms or editions of Java:

1) Java SE (Java Standard Edition)

It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

2) Java EE (Java Enterprise Edition)

It is an enterprise platform which is mainly used to develop web and enterprise applications. It is built on the top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA, etc.

3) Java ME (Java Micro Edition)

It is a micro platform which is mainly used to develop mobile applications.

4) JavaFX

It is used to develop rich internet applications. It uses a light-weight user interface API.

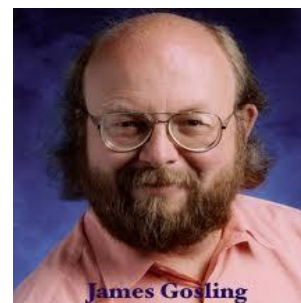
History of Java

The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of java starts with Green Team. Java team members (also known as **Green Team**), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was suited for internet programming. Later, Java technology was incorporated by Netscape.

The principles for creating Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted and Dynamic".

Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc. There are given the significant points that describe the history of Java.

- 1) **James Gosling, Mike Sheridan, and Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.
- 2) Originally designed for small, embedded systems in electronic appliances like set-top boxes.
- 3) Firstly, it was called "**Greentalk**" by James Gosling, and file extension was .gt.
- 4) After that, it was called **Oak** and was developed as a part of the Green project.



Why Java named "Oak"?

- 5) **Why Oak?** Oak is a symbol of strength and chosen as a national tree of many countries like U.S.A., France, Germany, Romania, etc.
- 6) In 1995, Oak was renamed as "**Java**" because it was already a trademark by Oak Technologies.



Why Java Programming named "Java"?

7) **Why had they chosen java name for java language?** The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA", etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.

According to James Gosling, "Java was one of the top choices along with **Silk**". Since Java was so unique, most of the team members preferred Java than other names.

8) Java is an island of Indonesia where first coffee was produced (called java coffee).

9) Notice that Java is just a name, not an acronym.

10) Initially developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.

11) In 1995, Time magazine called **Java one of the Ten Best Products of 1995**.

12) JDK 1.0 released in (January 23, 1996).

Java Version History

Many java versions have been released till now. The current stable release of Java is Java SE 12.

1. JDK Alpha and Beta (1995)

Version Name	Code Name	Release Date
JDK 1.0	Oak	January 1996
JDK 1.1	(none)	February 1997
J2SE 1.2	Playground	December 1998
J2SE 1.3	Kestrel	May 2000
J2SE 1.4	Merlin	February 2002
J2SE 5.0	Tiger	September 2004
JDK 6	Mustang	December 2006
JDK 7	Dolphin	July 2011
JDK 8		March 2014
JDK 9		September, 21st 2017
JDK 10		March, 20th 2018
JDK 11		September, 25th 2018
JDK 12		March, 19th 2019

Features of Java

The primary objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as java **buzzwords**.

A list of most important features of Java language is given below.

1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded



- 11. Distributed
- 12. Dynamic

Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun, Java language is a simple programming language because:

- Java syntax is based on C++ (so easier for programmers to learn it after C++).
- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Object-oriented

Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

Platform Independent

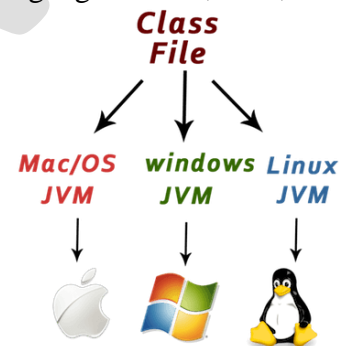
Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides a software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

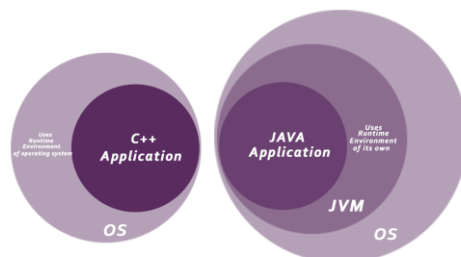
Java code can be run on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere(WORA).



Secured

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- No explicit pointer



- **Java Programs run inside a virtual machine sandbox**

Java language provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.

Robust

Robust simply means strong. Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java. All these points make Java robust.

Architecture-neutral

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

Portable

Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

High-performance

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

Distributed

Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

Dynamic

Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

Java supports dynamic compilation and automatic memory management (garbage collection).

First Java Program

In this page, we will learn how to write the simple program of java. We can write a simple hello java program easily after installing the JDK.

To create a simple java program, you need to create a class that contains the main method. Let's understand the requirement first.

```
class FirstJavaProgram
{
    public static void main(String args[])
    {
        System.out.println("This is my First Java Program");
    }
}
```

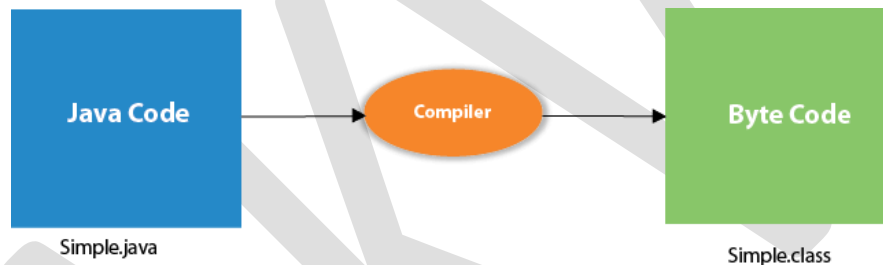
save this file as FirstJavaProgram.java

To compile: javac FirstJavaProgram.java

To execute: java FirstJavaProgram

Compilation Flow:

When we compile Java program using javac tool, java compiler converts the source code into byte code.



Parameters used in First Java Program

Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility. It means it is visible to all.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create an object to invoke the main method. So it saves memory.
- **void** is the return type of the method. It means it doesn't return any value.
- **main** represents the starting point of the program.
- **String[] args** is used for command line argument. We will learn it later.
- **System.out.println()** is used to print statement. Here, System is a class, out is the object of PrintStream class, println() is the method of PrintStream class. We will learn about the internal working of System.out.println statement later.

How to set path in Java

The path is required to be set for using tools such as javac, java, etc.

If you are saving the Java source file inside the JDK/bin directory, the path is not required to be set because all the tools will be available in the current directory.

However, if you have your Java file outside the JDK/bin folder, it is necessary to set the path of JDK.

There are two ways to set the path in Java:

1. Temporary
2. Permanent

1) How to set the Temporary Path of JDK in Windows

To set the temporary path of JDK, you need to follow the following steps:

- Open the command prompt
- Copy the path of the JDK/bin directory
- Write in command prompt: set path=copied_path

For Example:

```
set path=C:\Program Files\Java\jdk1.6.0_23\bin
```

2) How to set Permanent Path of JDK in Windows

For setting the permanent path of JDK, you need to follow these steps:

- Go to MyComputer properties -> advanced tab -> environment variables -> new tab of user variable -> write path in variable name -> write path of bin folder in variable value -> ok -> ok -> ok

Difference between JDK, JRE, and JVM

We must understand the differences between JDK, JRE, and JVM before proceeding further to Java. See the brief overview of JVM here.

If you want to get the detailed knowledge of Java Virtual Machine, move to the next page. Firstly, let's see the differences between the JDK, JRE, and JVM.

JVM

JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.

JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each OS is different from each other. However, Java is platform independent. There are three notions of the JVM: *specification*, *implementation*, and *instance*.

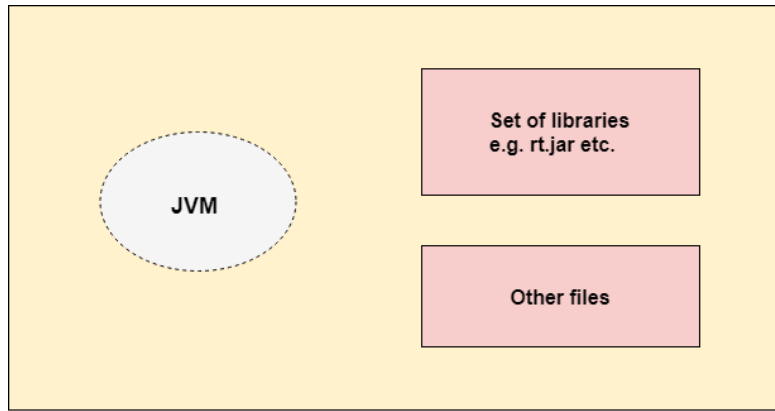
The JVM performs the following main tasks:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JRE

JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

The implementation of JVM is also actively released by other companies besides Sun Micro Systems.



JRE

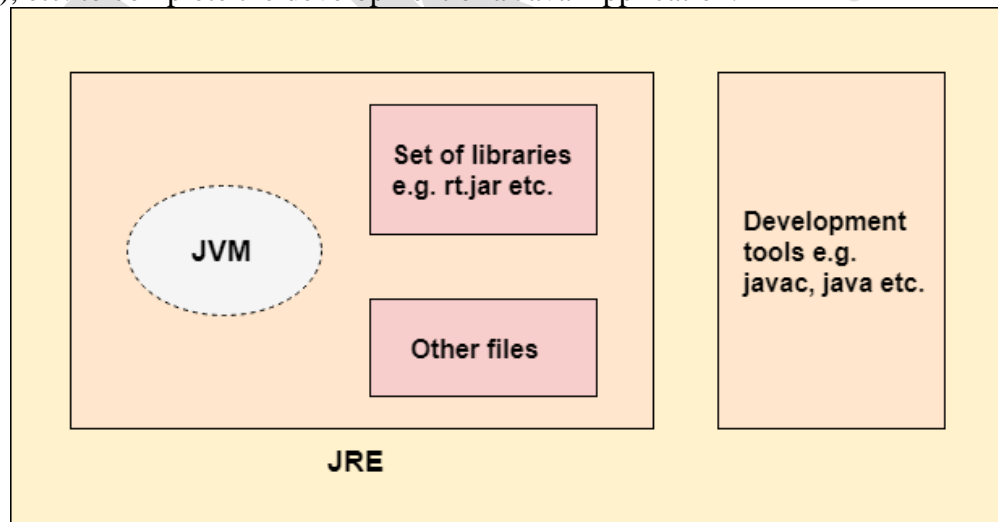
JDK

JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.

JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

- Standard Edition Java Platform
- Enterprise Edition Java Platform
- Micro Edition Java Platform

The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.



JDK

JVM (Java Virtual Machine) Architecture

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

What is JVM

It is:

1. **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Oracle and other companies.
2. **An implementation** Its implementation is known as JRE (Java Runtime Environment).
3. **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

What it does

The JVM performs following operation:

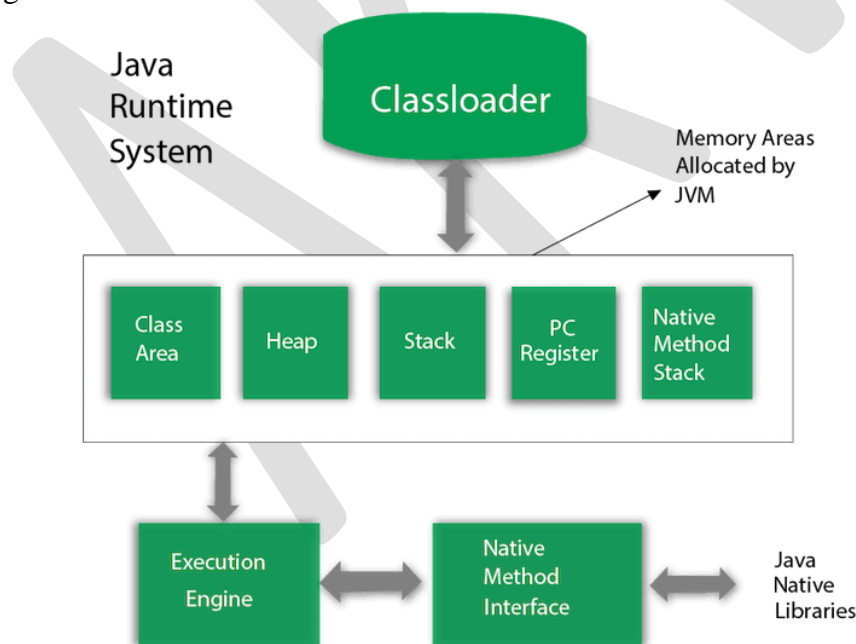
- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.

JVM Architecture

Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc.



1) Classloader

Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader.

2) Class(Method) Area

Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

3) Heap

It is the runtime data area in which objects are allocated.

4) Stack

Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

5) Program Counter Register

PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.

6) Native Method Stack

It contains all the native methods used in the application.

7) Execution Engine

It contains:

1. **A virtual processor**
2. **Interpreter:** Read bytecode stream then execute the instructions.
3. **Just-In-Time(JIT) compiler:** It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

8) Java Native Interface

Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, Assembly etc. Java uses JNI framework to send output to the Console or interact with OS libraries.

Data Types in Java

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

Data Type	Default Value	Default size
boolean	false	1 bit
Char	'\u0000'	2 byte
Byte	0	1 byte
short	0	2 byte
Int	0	4 byte
Long	0L	8 byte

Float	0.0f	4 byte
double	0.0d	8 byte

Java Variables

A variable is a container which holds the value while the java program is executed. A variable is assigned with a datatype. Variable is a name of memory location. There are three types of variables in java: local, instance and static. There are two types of data types in java: primitive and non-primitive.

Types of Variables

There are three types of variables in java:

- local variable
- instance variable
- static variable

1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.

It is called instance variable because its value is instance specific and is not shared among instances.

3) Static variable

A variable which is declared as static is called static variable. It cannot be local. You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

Example:

1. class A{
2. int data=50;//instance variable
3. static int m=100;//static variable
4. void method(){
5. int n=90;//local variable

6. }
7. }//end of class

Operators in java

Operator in java is a symbol that is used to perform operations.

Java Operator Precedence

Operator Type	Category	Precedence
Unary	Postfix	<i>expr</i> ++ <i>expr</i> --
	Prefix	++ <i>expr</i> -- <i>expr</i> + <i>expr</i> - <i>expr</i> ~ !
Arithmetic	Multiplicative	* / %
	Additive	+ -
Shift	Shift	<< >> >>>
Relational	Comparison	< > <= >= instanceof
	Equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	
Logical	logical AND	&&
	logical OR	
Ternary	Ternary	? :
Assignment	Assignment	= += -= *= /= %= &= ^= = <<= >>= >>>=

Java Keywords

Java keywords are also known as **reserved words**. Keywords are particular words which acts as a key to a code. These are predefined words by Java so it cannot be used as a variable or object name.

List of Java Keywords

A list of Java keywords or reserved words are given below:

1. **abstract:** Java abstract keyword is used to declare abstract class. Abstract class can provide the implementation of interface. It can have abstract and non-abstract methods.
2. **boolean:** Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.
3. **break:** Java break keyword is used to break loop or switch statement. It breaks the current flow of the program at specified condition.

4. **byte:** Java byte keyword is used to declare a variable that can hold an 8-bit data values.
5. **case:** Java case keyword is used to with the switch statements to mark blocks of text.
6. **catch:** Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.
7. **char:** Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters
8. **class:** Java class keyword is used to declare a class.
9. **continue:** Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.
10. **default:** Java default keyword is used to specify the default block of code in a switch statement.
11. **do:** Java do keyword is used in control statement to declare a loop. It can iterate a part of the program several times.
12. **double:** Java double keyword is used to declare a variable that can hold a 64-bit floating-point numbers.
13. **else:** Java else keyword is used to indicate the alternative branches in an if statement.
14. **enum:** Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.
15. **extends:** Java extends keyword is used to indicate that a class is derived from another class or interface.
16. **final:** Java final keyword is used to indicate that a variable holds a constant value. It is applied with a variable. It is used to restrict the user.
17. **finally:** Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether exception is handled or not.
18. **float:** Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.
19. **for:** Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some conditions become true. If the number of iteration is fixed, it is recommended to use for loop.
20. **if:** Java if keyword tests the condition. It executes the if block if condition is true.
21. **implements:** Java implements keyword is used to implement an interface.
22. **import:** Java import keyword makes classes and interfaces available and accessible to the current source code.
23. **instanceof:** Java instanceof keyword is used to test whether the object is an instance of the specified class or implements an interface.
24. **int:** Java int keyword is used to declare a variable that can hold a 32-bit signed integer.
25. **interface:** Java interface keyword is used to declare an interface. It can have only abstract methods.
26. **long:** Java long keyword is used to declare a variable that can hold a 64-bit integer.
27. **native:** Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).
28. **new:** Java new keyword is used to create new objects.
29. **null:** Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.
30. **package:** Java package keyword is used to declare a Java package that includes the classes.
31. **private:** Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.

32. **protected:** Java protected keyword is an access modifier. It can be accessible within package and outside the package but through inheritance only. It can't be applied on the class.
33. **public:** Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.
34. **return:** Java return keyword is used to return from a method when its execution is complete.
35. **short:** Java short keyword is used to declare a variable that can hold a 16-bit integer.
36. **static:** Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is used for memory management mainly.
37. **strictfp:** Java strictfp is used to restrict the floating-point calculations to ensure portability.
38. **super:** Java super keyword is a reference variable that is used to refer parent class object. It can be used to invoke immediate parent class method.
39. **switch:** The Java switch keyword contains a switch statement that executes code based on test value. The switch statement tests the equality of a variable against multiple values.
40. **synchronized:** Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.
41. **this:** Java this keyword can be used to refer the current object in a method or constructor.
42. **throw:** The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exception. It is followed by an instance.
43. **throws:** The Java throws keyword is used to declare an exception. Checked exception can be propagated with throws.
44. **transient:** Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.
45. **try:** Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.
46. **void:** Java void keyword is used to specify that a method does not have a return value.
47. **volatile:** Java volatile keyword is used to indicate that a variable may change asynchronously.
48. **while:** Java while keyword is used to start a while loop. This loop iterates a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

Java Control Flow

- Java If-else
- Java Switch
- Java For Loop
- Java While Loop
- Java Do While Loop
- Java Break
- Java Continue

Java Arrays

An array is a collection of similar type of elements which have a contiguous memory location.

Example 1

1. //Java Program to illustrate how to declare, instantiate, initialize
2. //and traverse the Java array.
3. class Testarray
4. {
5. public static void main(String args[])
6. {
7. int a[]=new int[5];//declaration and instantiation
8. a[0]=10;//initialization
9. a[1]=20;
10. a[2]=70;
11. a[3]=40;
12. a[4]=50;
13. //traversing array
14. for(int i=0;i<a.length;i++)//length is the property of array
15. System.out.println(a[i]);
16. }
17. }

Example 2

1. //Java Program to illustrate the use of declaration, instantiation
2. //and initialization of Java array in a single line
3. class Testarray1
4. {
5. public static void main(String args[])
6. {
7. int a[]={33,3,4,5};//declaration, instantiation and initialization
8. //printing array
9. for(int i=0;i<a.length;i++)//length is the property of array
10. System.out.println(a[i]);
11. }
12. }

Example 3

1. class LargestElement_array
2. {
3. public static void main(String[] args)
4. {
5. //Initialize array
6. int [] arr = new int [] {25, 11, 7, 75, 56};
7. //Initialize max with first element of array.
8. int max = arr[0];
9. //Loop through the array
10. for (int i = 0; i < arr.length; i++)
11. {
12. //Compare elements of array with max
13. if(arr[i] > max)
14. max = arr[i];
15. }


```

16.     System.out.println("Largest element present in given array: " + max);
17. }
18. }

```

Using Array input:-

```

class LargestElement_array
{
public static void main(String[] args)
{
    int n;
        int arr[] = new int[10];
        Scanner ip= new Scanner(System.in);
        System.out.print("Enter Length of array : ");
        n=ip.nextInt();
        System.out.println("Enter element of Array : ");
        for (int i = 0; i < n; i++)
        {
            arr[i]=ip.nextInt();
        }
        //Initialize max with first element of array.
        int max = arr[0];
        //Loop through the array
        for (int i = 0; i < arr.length; i++)
        {
            //Compare elements of array with max
            if(arr[i] > max)
                max = arr[i];
        }
        System.out.println("Largest element present in given array: " + max);
    }
}

```

BigInteger Class in Java

BigInteger class is used for mathematical operation which involves very big integer calculations that are outside the limit of all available primitive data types.

For example factorial of 100 contains 158 digits in it so we can't store it in any primitive data type available. We can store as large Integer as we want in it. There is no theoretical limit on the upper bound of the range because memory is allocated dynamically but practically as memory is limited you can store a number which has Integer.MAX_VALUE number of bits in it which should be sufficient to store mostly all large values.

Below is an example Java program that uses BigInteger to compute Factorial.

```

// Java program to find large factorials using BigInteger
import java.math.BigInteger;

```

```

import java.util.Scanner;
public class BigIntegerExample
{
    // Returns Factorial of N
    static BigInteger factorial(int N)
    {
        // Initialize result
        BigInteger f = new BigInteger("1"); // Or BigInteger.ONE
        // Multiply f with 2, 3, ...N
        for (int i = 2; i <= N; i++)
            f = f.multiply(BigInteger.valueOf(i));
        return f;
    }
    // Driver method
    public static void main(String args[]) //throws Exception
    {
        int N = 20;
        System.out.println(factorial(N));
    }
}

```

Output:

```
2432902008176640000
```

If we have to write above program in C++, that would be too large and complex, we can look at Factorial of Large Number.

In this way BigInteger class is very handy to use because of its large method library and it is also used a lot in competitive programming.

Now below is given a list of simple statements in primitive arithmetic and its analogous statement in terms of BigInteger objects.

Declaration

```

int a, b;
BigInteger A, B;

```

Initialization:

```

a = 54;
b = 23;
A = BigInteger.valueOf(54);
B = BigInteger.valueOf(37);

```

And for Integers available as string you can initialize them as:

```
A = new BigInteger("54");
```

```
B = new BigInteger("123456789123456789");
```

Some constant are also defined in BigInteger class for ease of initialization :

```
A = BigInteger.ONE;  
// Other than this, available constant are BigInteger.ZERO  
// and BigInteger.TEN
```

Mathematical operations:

```
int c = a + b;  
BigInteger C = A.add(B);
```

Other similar function are subtract(), multiply(), divide(), remainder()

But all these function take BigInteger as their argument so if we want these operation with integers or string convert them to BigInteger before passing them to functions as shown below:

```
String str = "123456789";  
BigInteger C = A.add(new BigInteger(str));  
int val = 123456789;  
BigInteger C = A.add(BigInteger.valueOf(val));
```

Extraction of value from BigInteger:

```
int x = A.intValue(); // value should be in limit of int x  
long y = A.longValue(); // value should be in limit of long y  
String z = A.toString();
```

Comparison:

```
if (a < b) {} // For primitive int  
if (A.compareTo(B) < 0) {} // For BigInteger
```

Actually compareTo returns -1(less than), 0(Equal), 1(greater than) according to values.

For equality we can also use:

```
if (A.equals(B)) {} // A is equal to B
```