In **Java 1.2** Swing and Collection framework was added and suspend (), resume() and stop()methods were deprecated from **Thread** class.

**Java 1.4** contained several important changes. Keyword assert, chained exceptions and channel based I/O System was introduced.

**Java 1**.**5** was called **J2SE** **5**, it added following major new features:

1. Generics
2. Annotations
3. Autoboxing and auto-unboxing
4. Enumerations
5. For-each Loop
6. Varargs
7. Static Import
8. Formatted I/O
9. Concurrency utilities

**Java SE 7** which included many new changes, like:

1. Now String can be used to control Switch statement.
2. Multi Catch Exception
3. try-with-resource statement
4. Binary Integer Literals
5. Underscore in numeric literals, etc.

**Java SE 8,** it was released on March 18, 2014. Some of the major new features introduced in JAVA 8 are,

1. Lambda Expressions
2. New Collection Package java.util.stream to provide Stream API.
3. Enhanced Security
4. Nashorn Javascript Engine included
5. Parallel Array Sorting
6. The JDBC-ODBC Bridge has been removed etc.

Key Features of Java:

1. **Object Oriented**

In java everything is Object which has some data and behavior. Java can be easily extended as it is based on Object Model.

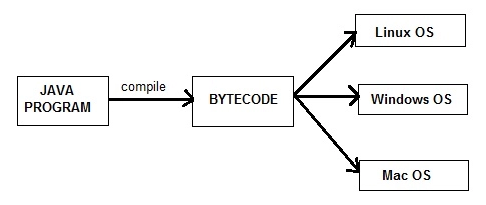
1. **Robust**

Java tries to eliminate error prone codes by emphasizing mainly on compile time error checking and runtime checking. Main areas which Java improved were Memory Management and mishandled Exceptions by introducing automatic Garbage Collector and Exception Handling

1. **Platform Independent**

Unlike other programming languages such as C, C++ etc which are compiled into platform specific machines. Java is guaranteed to be write-once, run-anywhere language.

On compilation Java program is compiled into bytecode. This bytecode is platform independent and can be run on any machine, plus this bytecode format also provide security. Any machine with Java Runtime Environment can run Java Programs.



#### Secure

Java program always runs in Java runtime environment with almost null interaction with system OS, hence it is more secure.

#### Multi-Threading

Java multithreading feature makes it possible to write program that can do many tasks simultaneously. Benefit of multithreading is that it utilizes same memory and other resources to execute multiple threads at the same time, like While typing, grammatical errors are checked along.

#### Portable

Java Byte code can be carried to any platform. No implementation dependent features. Everything related to storage is predefined, example: size of primitive data types

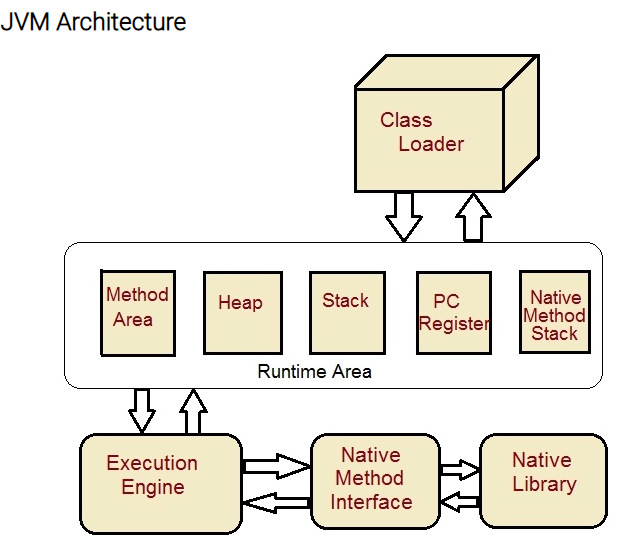
#### High Performance

Java is an interpreted language, so it will never be as fast as a compiled language like C or C++. But, Java enables high performance with the use of just-in-time compiler.

### **What is JVM?**

Java virtual Machine(JVM) is a virtual Machine that provides runtime environment to execute java byte code. The JVM doesn't understand Java typo, that's why you compile your \*.java files to obtain \*.class files that contain the bytecodes understandable by the JVM

JVM control execution of every Java program. It enables features such as automated exception handling, Garbage-collected heap.



**Class Loader:** Class loader loads the Class for execution.

**Method area:** Stores pre-class structure as constant pool.

**Heap:** Heap is in which objects are allocated.

**Stack:** Local variables and partial results are store here. Each thread has a private JVM stack created when the thread is created.

**Program register:** Program register holds the address of JVM instruction currently being executed.

**Native method stack:** It contains all native used in application.

**Executive Engine:** Execution engine controls the execute of instructions contained in the methods of the classes.

**Native Method Interface:** Native method interface gives an interface between java code and native code during execution.

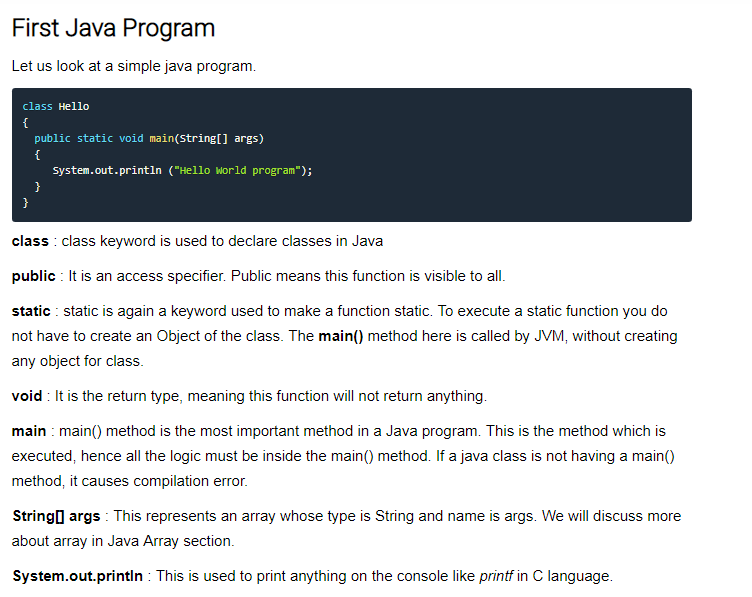
**Native Method Libraries:** Native Libraries consist of files required for the execution of native code.

**Difference between JDK and JRE:**

**JRE**: The Java Runtime Environment (JRE) provides the libraries, the Java Virtual Machine, and other components to run applets and applications written in the Java programming language. JRE does not contain tools and utilities such as compilers or debuggers for developing applets and applications.

**JDK**: The JDK also called Java Development Kit is a superset of the JRE, and contains everything that is in the JRE, plus tools such as the compilers and debuggers necessary for developing applets and applications.



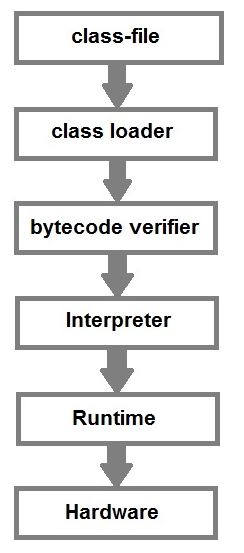


#### What happens at Runtime

1. Class loader loads the java class

2. Byte Code verifier checks the code fragments for illegal codes that can violate access right to the object.

3. Interpreter reads the byte code stream and then executes the instructions, step by step.



### **Data Types in Java**

1. **Primitive Data type**

**Integer**

1. Byte: 1-byte Value range from -128 to 127
2. Short: 2-byte Value range from -32768 to 32767
3. Int: 4-bytes Value range from -2147483648 to 2147483647
4. Long:8-byte

**Floating-Point Number**

1. **float:** It is 4 bytes(32-bits) float data type. Default value 0.0f. example: float ff=10.3f;
2. **double:** It is 8 bytes(64-bits) float data type. Default value 0.0d. example: double db=11.123;

**Characters**

1. **char**: It is 2 bytes(16-bits) unsigned unicode character. Range 0 to 65,535. example: char c='a';

**Boolean**

boolean

**Type Casting:**

**Widening:**

int i = 100;

long l = i; //no explicit type casting required

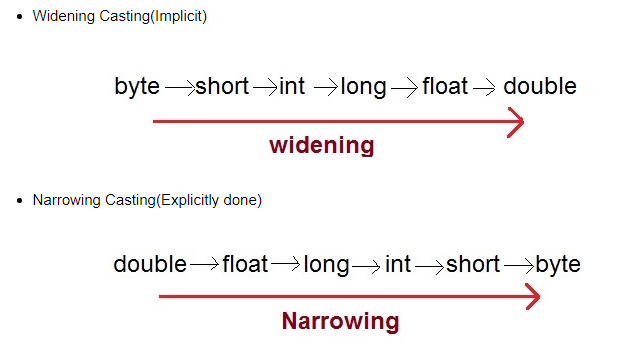
float f = l;

**Narrowing:**

double d = 100.04;

long l = (long)d; //explicit type casting required

int i = (int)l; //explicit type casting required



**What is a variable?**

The naming of an address is known as variable. Variable is the name of memory location.

Java Programming language defines mainly three kinds of variables.

1. **Instance variables**

It is declared inside a class but outside any method, constructor or block. They are referred as object variable. Each object has its own copy of each variable.

1. **Static Variables**

Static are class variables declared with static keyword. Static variables are initialized only once. Static variables are also used in declaring constant along with final keyword.

Additional points on static variable:

* static variable is also known as class variable.
* static means to remain constant.
* In Java, it means that it will be constant for all the instances created for that class.
* static variable need not be called from object.
* It is called by classname.static variable name.

1. **Local Variables**

Local variables are initialized when method, constructor or block start and will be **destroyed** once its end. Local variable resides in **stack**.

**Array:**

An array is a collection of similar data types.  It gets memory in heap area.

Bitwise Operator:

&, |, ^, >> (left shift), << (Right Shift)

#### instanceOf operator

This operator is used for object reference variables. The operator checks whether the object is of type (class type or interface type).

#### Main Features of OOPS

1. Inheritence
2. Polymorphism
3. Encapsulation
4. Abstraction

#### Rules for Java Class

1. A class can have **only public or default** (no modifier) access specifier.
2. It can be either **abstract**, **final** or **concrete** (normal class).
3. It must have the class keyword, and class must be followed by a legal identifier.
4. It may optionally extend one parent class. By default, it will extend java.lang.Object.
5. It may optionally implement any number of comma-separated interfaces.
6. Each **.java** source file may contain only one public class. A source file may contain any number of default visible classes.
7. Finally, the source file name must match the public class name and it must have a .java suffix.

class Student.

{

String name;

int rollno;

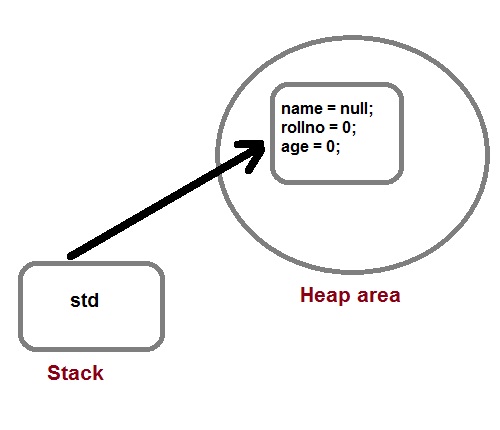
int age;

}

When a reference is made to a student with its property then it becomes an object, physical existence of Student class.

Student std=new Student ();

After the above statement **std** is instance/object of Student class. Here the **new** keyword creates an actual physical copy of the object and assign it to the **std** variable. It will have physical existence and get memory in heap area. **The new operator dynamically allocates memory for an object**



**Q How a class is initialized in java?**

A Class is initialized in Java when an instance of class is created using either **new** operator or using reflection using class.forName(). A class is also said to be initialized when a static method of **Class** is invoked, or a static field of **Class** is assigned.

**Q. How would you make a copy of an entire Java object with its state?**

Make that class implement **Cloneable** interface and call **clone ()** method on its object. **Clone ()** method is defined in **Object** class which is parent of all java class by default.

#### Q. Does constructors return any value?

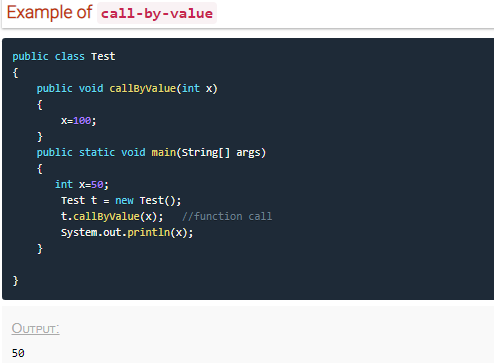
Yes, constructors return current instant of a class. But constructor signature cannot have any return type.

#### call-by-value and call-by-reference

There are two ways to pass an argument to a method

1. **call-by-value:** In this approach copy of an argument value is pass to a method. Changes made to the argument value inside the method will have no effect on the arguments.
2. **call-by-reference:** In this reference of an argument is pass to a method. Any changes made inside the method will affect the argument value.

**NOTE:** There is only call by value in java, not call by reference.



**If two or more method have same name and same parameter list but differs in return type are not said to be overloaded method.**

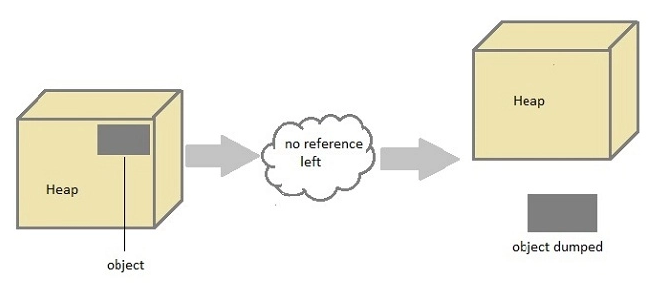
### **this keyword**

* **this** keyword is used to refer to current object.
* **this** is always a reference to the object on which method was invoked.
* **this** can be used to invoke current class constructor.
* **this** can be passed as an argument to another method.

### **Garbage Collection**

In Java destruction of object from memory is done automatically by the JVM. When there is no reference to an object, then that object is assumed to be no longer needed and the memory occupied by the object are released. This technique is called Garbage Collection. This is accomplished by the JVM.

Unlike C++ there is no explicit need to destroy object.



#### Can the Garbage Collection be forced explicitly?

No, the Garbage Collection cannot be forced explicitly. We may request JVM for **garbage collection** by calling **System.gc()** method. But This does not guarantee that JVM will perform the garbage collection.

#### Advantages of Garbage Collection

1. Programmer doesn't need to worry about dereferencing an object.
2. It is done automatically by JVM.
3. Increases memory efficiency and decreases the chances for memory leak.

#### finalize () method

Sometime an object will need to perform some specific task before it is destroyed such as closing an open connection or releasing any resources held. To handle such situation, finalize () method is used. Finalize () method is called by garbage collection thread before collecting object. It’s the last chance for any object to perform cleanup utility.

#### Some Important Points to Remember

1. finalize() method is defined in **java.lang.Object** class, therefore it is available to all the classes.
2. finalize() method is declare as **proctected** inside Object class.
3. finalize() method gets called only once by a Daemon thread named GC (Garbage Collector)thread.

#### gc() Method

**gc()** method is used to call garbage collector explicitly. However **gc ()** method does not guarantee that JVM will perform the garbage collection. It only requests the JVM for garbage collection. This method is present in **System** and **Runtime** class.

#### Example for gc() method

public class Test

{

public static void main(String[] args)

{

Test t = new Test ();

t=null;

System.gc();

}

public void finalize ()

{

System.out.println("Garbage Collected");

}

}

Output: Garbage Collected