Java

Overview of Java

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

Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995. *James Gosling* is known as the father of Java. Before Java, its name was *Oak*. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

History

Java was developed by James Ghosling, Patrick Naughton, Mike Sheridan at Sun Microsystems Inc. in 1991. It took 18 months to develop the first working version.

The initial name was **Oak** but it was renamed to **Java** in 1995 as OAK was a registered trademark of another Tech company.

Features of Java

Simple

Java is easy to learn and its syntax is quite simple, clean and easy to understand.The confusing and ambiguous concepts of C++ are either left out in Java or they have been re-implemented in a cleaner way.

Eg : Pointers and Operator Overloading are not there in java but were an important part of C++.

Object Oriented

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

Robust

Java makes an effort to eliminate error prone codes by emphasizing mainly on compile time error checking and runtime checking. But the main areas which Java improved were Memory Management and mishandled Exceptions by introducing automatic **Garbage Collector** and **Exception Handling**.

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

When it comes to security, Java is always the first choice. With java secure features it enable us to develop virus free, temper free system. 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.

Architectural Neutral

Compiler generates bytecodes, which have nothing to do with a particular computer architecture, hence a Java program is easy to intrepret on any machine.

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

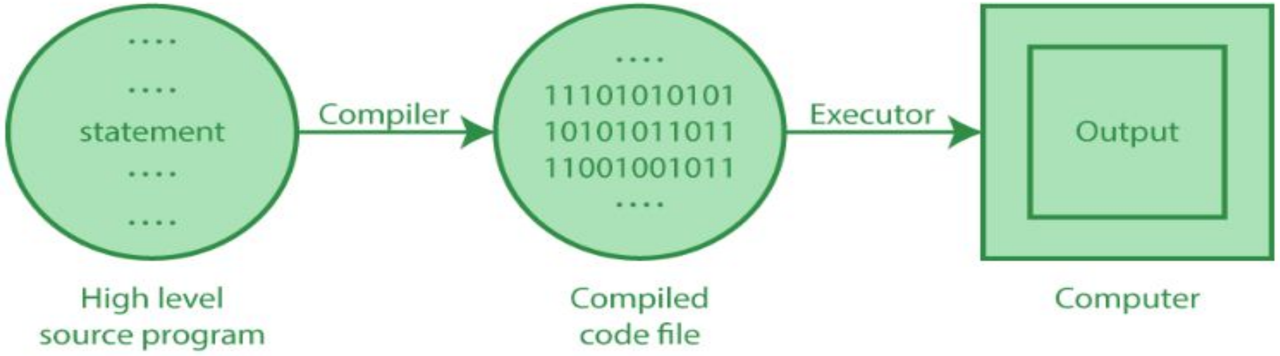
Java is an interpreted language, so it will never be as fast as a compiled language like C or C++. But, Java enables high performance with the use of just-in-time compiler.

Java Compiler

A compiler in Java translates the entire source code into a machine-code file or any intermediate code, and that file is then executed. It is platform-independent. A bytecode is basically an intermediate code generated by the compiler after the compilation of its source code.

Roles of Java Compiler

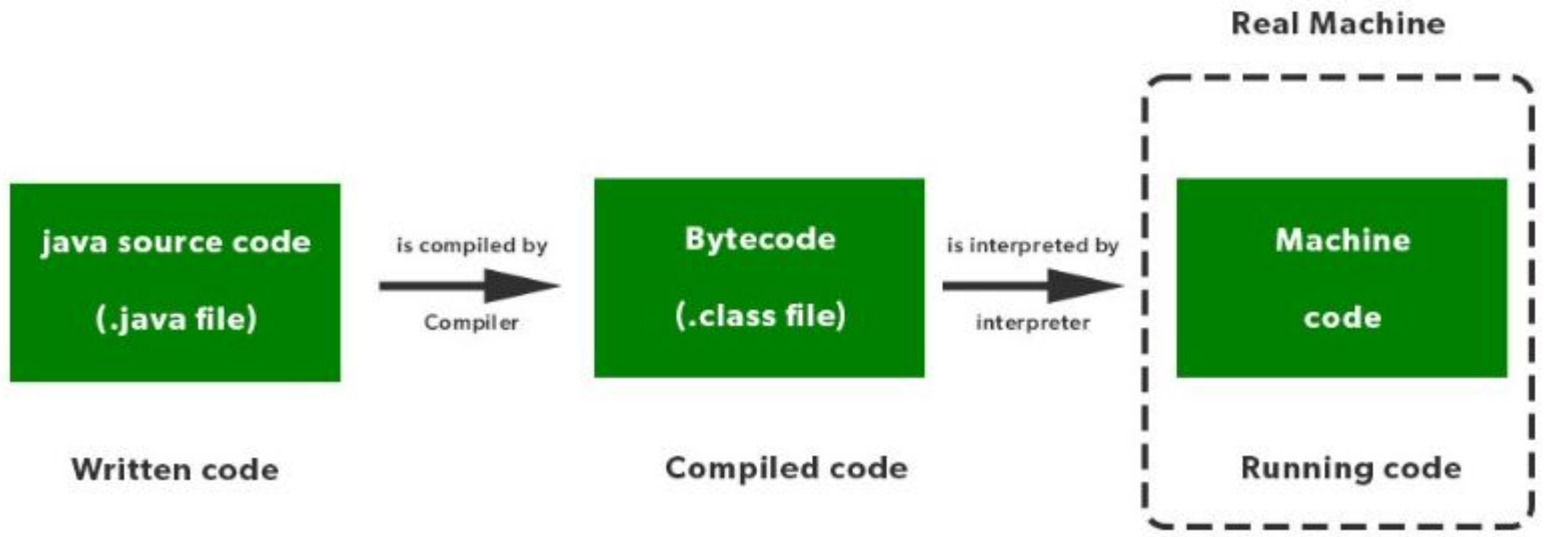
- It scans the complete source code in one go and then highlights the error.
- It requires more memory to produce the bytecode.
- It checks the correctness of the program by checking the type errors, syntax, etc.
- It also adds some additional code to our program if required.



Java Interpreter

an Interpreter is a computer program that converts the high-level program statement into Assembly-level language. It converts the code into machine code when the program is run.

- To convert the bytecode into the native code of the machine.
- This process is done line by line.
- If the error comes on any line, the process is stopped over there.

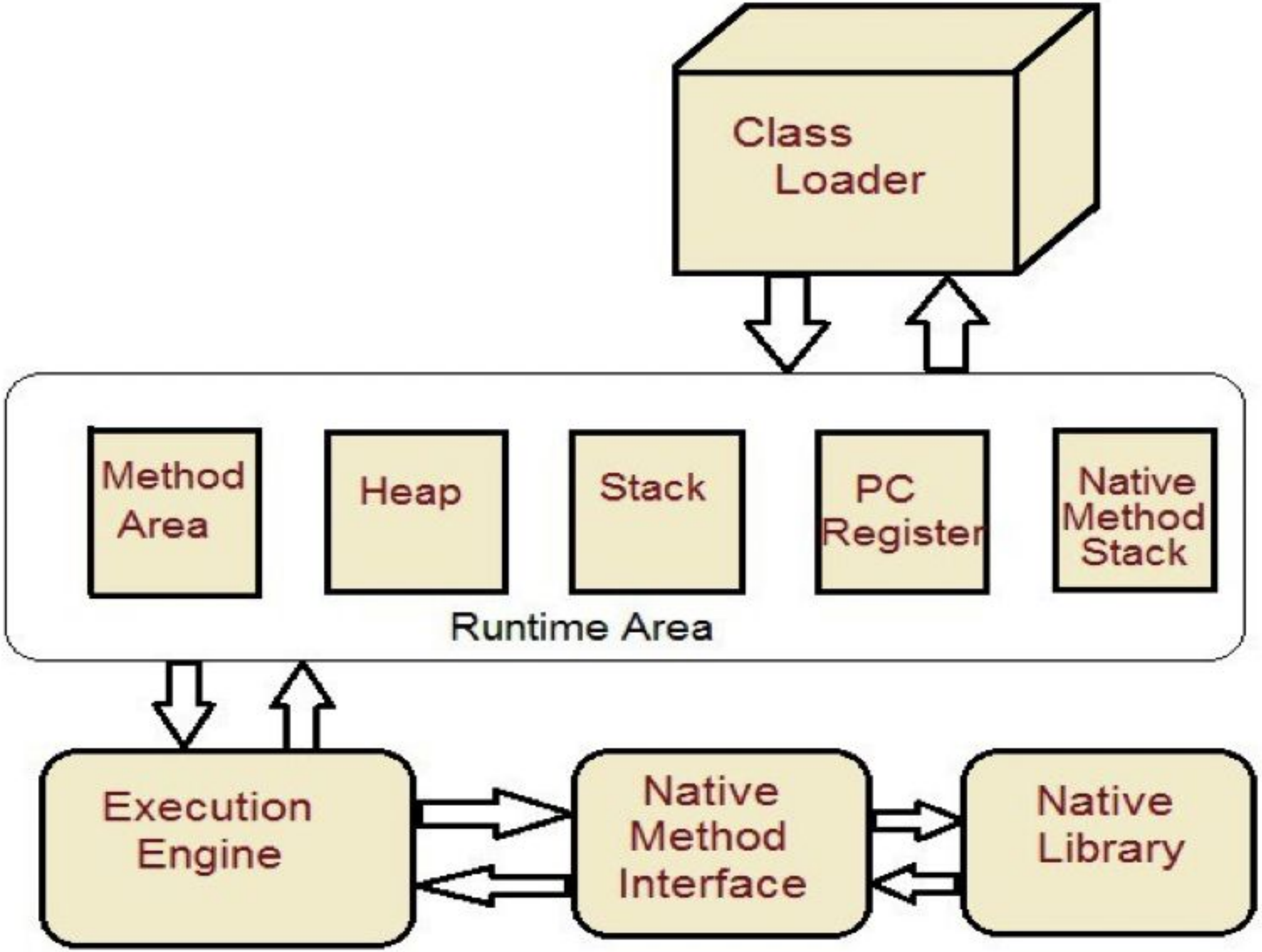


What is JVM?

Java virtual Machine(JVM) is a virtual Machine that provides runtime environment to execute java byte code. The JVM doesn't understand Java Code, 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.

JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, **Java is platform independent.**


Class Loader
Method Area
Heap
Stack
PC Register
Native Method Stack
Runtime Area
Execution Engine
Native Method Interface
Native Library

JVM Architecture

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

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

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

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

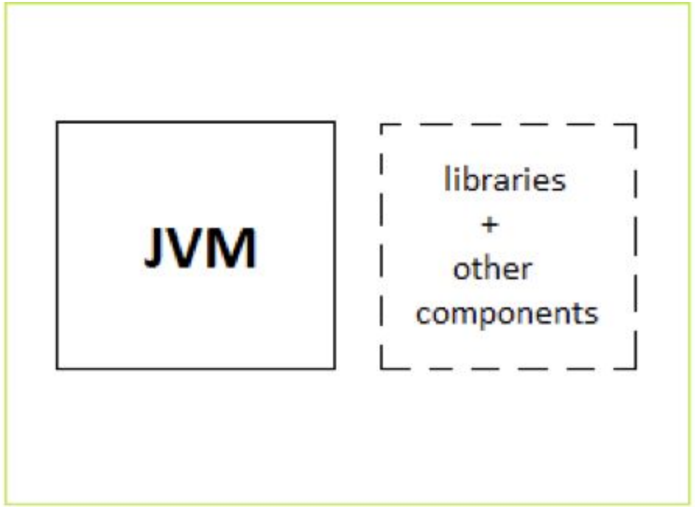
Executive Engine : Execution engine controls the execute of instructions contained in the methods of the classes.

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.

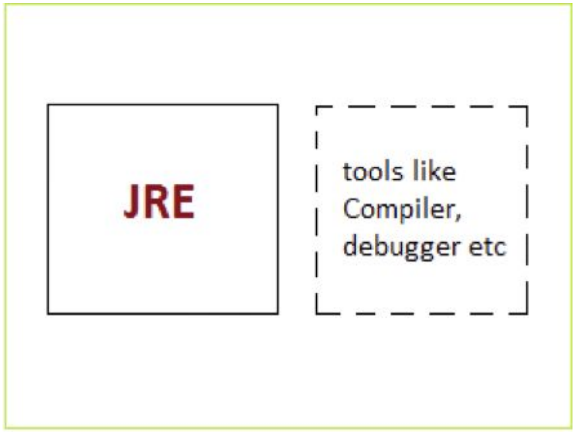
Difference between JDK and JRE

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 app



JRE - Java Runtime Environment

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.



JDK - Java Development Kit

First Java Program

```
class Hello
{
 public static void main(String[] args)
 {
   System.out.println (" Hello World program");
 }
}
```

class : class keyword is used to declare classes in Java

public : It is an access specifier. Public means this function is visible to all.

static : static is again a keyword used to make a function static. To execute a static function you do not have to create an Object of the class. The **main()** method here is called by JVM, without creating any object for class.

void : It is the return type, meaning this function will not return anything.

main : main() method is the most important method in a Java program. This is the method which is executed, hence all the logic must be inside the main() method. If a java class is not having a main() method, it causes compilation error.

System.out.println : This is used to print anything on the console like *printf* in C language.

Data Types in Java

Java language has a rich implementation of data types. Data types specify size and the type of values that can be stored in an identifier.

In java, data types are classified into two categories :

1. Primitive Data type
2. Non-Primitive Data type

Primitive Data type

A primitive data type can be of eight types :

`char boolean  byte short int long float`

Once a primitive data type has been declared its type can never change, although in most cases its value can change. These eight primitive type can be put into four groups

Integer

This group includes `byte`, `short`, `int`, `long`

byte : It is 8 bit integer data type. Value range from -128 to 127. Default value zero. example:
`byte b=10;`

short : It is 16 bit integer data type. Value range from -32768 to 32767. Default value zero. example: `short s=11;`

int : It is 32 bit integer data type. Value range from -2147483648 to 2147483647. Default value zero. example: `int i=10;`

long : It is 64 bit integer data type. Value range from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. Default value zero. example: `long l=100012;`

Floating-Point Number

This group includes `float`, `double`

float : It is 32 bit float data type. Default value 0.0f. example: `float ff=10.3f;`

double : It is 64 bit float data type. Default value 0.0d. example: `double db=11.123;`

Characters

This group represent `char`, which represent symbols in a character set, like letters and numbers.

char : It is 16 bit unsigned unicode character. Range 0 to 65,535. example: `char c='a';`

Boolean

This group represent `boolean`, which is a special type for representing true/false values. They are defined constant of the language. example: `boolean b=true;`

Identifiers in Java

All Java components require names. Name used for classes, methods, interfaces and variables are called **Identifier**. Identifier must follow some rules. Here are the rules:

- All identifiers must start with either a letter(a to z or A to Z) or currency character($) or an underscore.
- After the first character, an identifier can have any combination of characters.
- A Java **keyword** cannot be used as an identifier.
- Identifiers in Java are case sensitive, foo and Foo are two different identifiers

Concept of Array in Java

An array is a collection of similar data types. Array is a container object that hold values of homogeneous type. It is also known as static data structure because size of an array must be specified at the time of its declaration.

An array can be either primitive or reference type. It gets memory in heap area. Index of array starts from zero to size-1.

Array Declaration Syntax :

```
datatype[] identifier; or
datatype identifier[];
```

Both are valid syntax for array declaration. But the former is more readable.

Example :

```
int[] arr; char[] arr; short[] arr; long[] arr;
int[][] arr; //two dimensional array.
```

Initialization of Array

`new` operator is used to initialize an array.

Example :

```
int[] arr = new int[10]; //10 is the size of array. or
int[] arr = {10,20,30,40,50};
```

Accessing array element

As mention ealier array index starts from 0. To access nth element of an array. Syntax

```
arrayname[n-1];
```

Example : To access 4th element of a given array

```
int[] arr={10,20,30,40};
System.out.println("Element at 4th place"+arr[3]);
```

The above code will print the 4th element of array arr on console.

Java Command Line Arguments

The java command-line argument is an argument i.e. passed at the time of running the java program.

The arguments passed from the console can be received in the java program and it can be used as an input.

So, it provides a convenient way to check the behavior of the program for the different values. You can pass **N** (1,2,3 and so on) numbers of arguments from the command prompt.

```
Class CommandLine

{

        public static void main(String args[]){

        System.out.println("Hello "+args[0]);

        }

        }
```

Types of Java Comments

There are three types of comments in Java.

1. **Single Line Comment**

The single-line comment is used to comment only one line of the code. It is the widely used and easiest way of commenting the statements.

Single line comments starts with two forward slashes **(//)**. Any text in front of // is not executed by Java.

2. **Multi Line Comment**

The multi-line comment is used to comment multiple lines of code. It can be used to explain a complex code snippet or to comment multiple lines of code at a time (as it will be difficult to use single-line comments there).

Multi-line comments are placed between /* and */. Any text between /* and */ is not executed by Java.

3. **Documentation Comment**

Documentation comments are usually used to write large programs for a project or software application as it helps to create documentation API. These APIs are needed for reference, i.e., which classes, methods, arguments, etc., are used in the code.

To create documentation API, we need to use the **javadoc tool**. The documentation comments are placed between /** and */.

Accepting input from console

1. **Using BufferedReader class:**

```
BufferedReader br = new BufferedReader(
  new InputStreamReader(System.in));
```

- `String s = br.readLine();`
- `int a = Integer.parseInt(br.readLine());`
- `float b=Float.parseFloat(br.readLine());`
- `double c=Double.parseDouble(br.readLine());`

2. Using Scanner Class:

Scanner sc = **new** Scanner(System.in);

- int a = sc.nextInt();
- String b = sc.nextLine();
- float c=sc.nextFloat();
- double d=sc.nextDouble();

Class

In Java everything is encapsulated under classes. Class is the core of Java language. Class can be defined as a template/ blueprint that describe the behaviors /states of a particular entity. A class defines new data type. Once defined this new type can be used to create object of that type.

Object is an instance of class. You may also call it as physical existence of a logical template class.

A class is declared using **class** keyword. A class contain both data and code that operate on that data. The data or variables defined within a **class** are called **instance variables** and the code that operates on this data is known as **methods**.

Declaring class

Syntax:

```
Class class_name

{

Variables;

methods();

}
```

Student is a **class** and student's name, roll number, age will be its property. Lets see this in Java syntax

```
class Student.
{
 String name; int rollno; int age;
}
```

When a reference is made to a particular 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**

Access Modifiers

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Array of Objects

Syntax:

ClassName obj[]=**new** ClassName[array_length];

Example:

Student s[]=new Students[5];

In above example, we can store details of 5 students.

```
Class Student

{

Int rno;

String name;

Void accept(){}

Void display(){}

Public static void main(String args[])

{

Student s=new Student();

s.accept();

s.display();


})

}
```

Constructors in Java

- A constructor is a special method that is used to initialize an object.Every class has a constructor,if we don't explicitly declare a constructor for any java class the compiler builds a default constructor for that class.
- A constructor does not have any return type.
- A constructor has same name as the class in which it resides.
- Constructor in Java can not be abstract, static, final or synchronized. These modifiers are not allowed for constructor.
- Each time a new object is created at least one constructor will be invoked.

Syntax:

```
class Car
{
 String name ;
 String model;
 Car( )    //Constructor
 {
  name ="";
  model="";
 }
}
```

There are two types of Constructor

- Default Constructor

  A constructor without any parameter is called as default constructor.

- Parameterized constructor

  A constructor with parameter is called as Parameterized constructor.

Usage of Java this keyword

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

Constructor Overloading

Like methods, a constructor can also be overloaded. Overloaded constructors are differentiated on the basis of their type of parameters or number of parameters. Constructor overloading is not much different than method overloading. In case of method overloading you have multiple methods with same name but different signature, whereas in Constructor overloading you have multiple constructor with different signature but only difference is that Constructor doesn't have return type in Java.

```
Class Add
{
Int a, b,c;
Add()
{
a=10;
b=20;
}
Add(int a, int b)
{
this.a=a;
this.b=b;
}
Static Void sum()
{
c=a+b;
System.out. println(“Addition”:+c);
}
public static void main(String args[])
{
Add a1=new Add();
Add a2=new Add(11,22);
a1.sum();
a2.sum();
}
}
```

Java static keyword

Whenever we use static keyword then single copy of variable or method is shared by all the objects of class.

Static members belongs to class, not to the object.

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block

Java static variable

If you declare any variable as static, it is known as a static variable.

- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.

Java static method

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

Why is the Java main method static?

It is because the object is not required to call a static method.

If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation.

Object Class in Java

- **Object** class is present in **java.lang** package.
- **Object** class is **Super class** of all classes in java
- Every class in Java is directly or indirectly derived from the **Object** class.
- If a class does not extend any other class then it is a direct child class of **Object** and if extends another class then it is indirectly derived.
- Therefore the Object class methods are available to all Java classes. Hence Object class acts as a root of the inheritance hierarchy in any Java Program.

Methods of Object class:

- toString() method
- hashCode() method
- equals(Object obj) method
- finalize() method
- getClass() method
- clone() method
- wait(), notify() notifyAll() methods

Java Garbage Collection

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

Advantage of Garbage Collection:

- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
- It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

finalize() method

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

```
protected void finalize(){}
```

gc() method

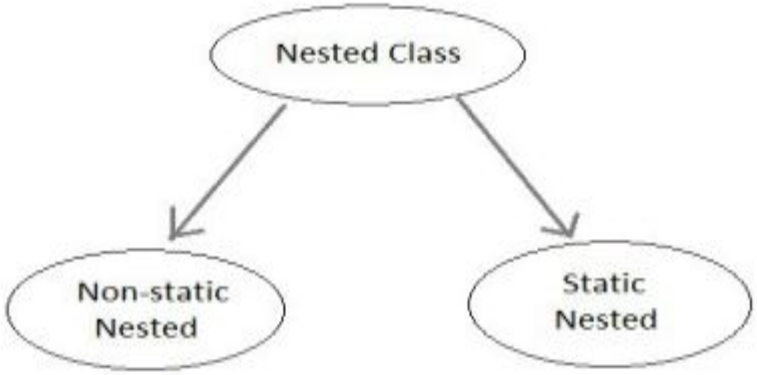
The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

```
public static void gc(){}
public class Garbage{


 public void finalize()
{
System.out.println("object is garbage collected");
}
 public static void main(String args[]){
  Garbage s1=new Garbage();
  Garbage s2=new Garbage();
  s1=null;
  s2=null;
  System.gc();
 }
}
```

Nested Class

- A class within another class is known as Nested class.
- The scope of the nested is bounded by the scope of its enclosing class.
- A nested class has access to the members, including private members, of the class in which it is nested. But the enclosing class does not have access to the member of the nested class.
- A nested class is also a member of its enclosing class.
- As a member of its enclosing class, a nested class can be declared *private*, *public*, *protected*, or *package-private*(default).

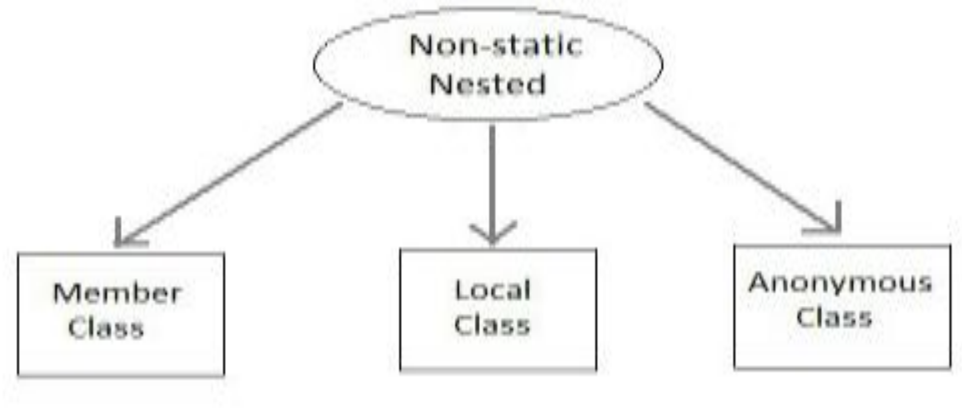


Static Nested Class

A static nested class is the one that has static modifier applied. Because it is static it cannot refer to non-static members of its enclosing class directly.

Non-static Nested class

Non-static Nested class is most important type of nested class. It is also known as Inner class. It has access to all variables and methods of Outer class and may refer to them directly. But the reverse is not true, that is, Outer class cannot directly access members of Inner class. One more important thing to notice about an Inner class is that it can be created only within the scope of Outer class. Java compiler generates an error if any code outside Outer class attempts to instantiate Inner class.



Example of Inner class

```
class Outer{
 private int data=30;
 class Inner{
  void msg(){System.out.println("data is "+data);
}
 }
 public static void main(String args[])
{
  Outer obj=new Outer();
  Outer.Inner in=obj.new Inner();
  in.msg();
 }
}
```

Annonymous inner class

- It is an inner class without a name and for which only a single object is created. An anonymous inner class can be useful when making an instance of an object with certain “extras” such as overloading methods of a class or interface, without having to actually subclass a class.
- Anonymous inner classes are useful in writing implementation classes for listener interfaces in graphics programming.
- Anonymous inner class are mainly created in two ways:
- Class (may be abstract or concrete)
- Interface

Syntax:

```
//Test can be interface,abstract/concrete
class Test t = new Test()
{ // data members and methods public void test_method()
{
........ ........
}
};
```

Java static nested class

- A static class is a class that is created inside a class, is called a static nested class in Java.
- It cannot access non-static data members and methods.
- It can be accessed by outer class name.
- It can access static data members of the outer class, including private.
- The static nested class cannot access non-static (instance) data members

```
class A
{
  static int data=30;
  static class B
{
   void msg(){System.out.println("data is "+data);}
  }
  public static void main(String args[]){
  A.B obj=new A.B();
  obj.msg();
  }
}
```

Inner Class	Static nested Class
The inner class object is always associated with the outer class object.	Static nested class object is not associated with the outer class object.
Inside normal/regular inner class, static members can't be declared.	Inside static nested class, static members can be declared.
As main() method can't be declared, regular inner class can't be invoked directly from the command prompt.	As main() method can be declared, the static nested class can be invoked directly from the command prompt.
Both static and non static members of outer class can be accessed directly.	Only a static member of outer class can be accessed directly.