**Java Introduction**

Java is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

Java is an Object-Oriented Language that has the Object-Oriented feature. It supports the following fundamental concepts (Polymorphism, Inheritance, Encapsulation, Abstraction, Classes, Objects, Instance, Method, Message Parsing).

**Features of Java**

* **Object Oriented** − In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
* **Platform Independent** − Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
* **Simple** − Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
* **Secure** − With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
* **Architecture-neutral** − Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
* **Portable** − Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.
* **Robust** − Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
* **Multithreaded** − With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
* **Interpreted** − Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
* **High Performance** − With the use of Just-In-Time compilers, Java enables high performance.
* **Distributed** − Java is designed for the distributed environment of the internet.
* **Dynamic** − Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

**Java Basic**

**Case Sensitivity** − Java is case sensitive, which means identifier **Hello** and **hello** would have different meaning in Java.

**Program File Name** − Name of the program file should exactly match the class name.

**Example:** Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as *'MyFirstJavaProgram.java'*

**public static void main(String args[])** − Java program processing starts from the main() method which is a mandatory part of every Java program.

It can be defined as a collection of objects that communicate via invoking each other's methods

* **Object** − Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behavior such as wagging their tail, barking, eating. An object is an instance of a class. The new keyword is used to create new objects.
* **Class** − A class can be defined as a template/blueprint that describes the behavior/state that the object of its type supports. Objects are created from class.
* **Methods** − A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
* **Instance Variables** − Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

## **Java Identifiers**

All Java components require names. Names used for classes, variables, and methods are called **identifiers**.

* All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (\_).
* After the first character, identifiers can have any combination of characters.
* A key word cannot be used as an identifier.
* Most importantly, identifiers are case sensitive.
* Examples of legal identifiers: age, $salary, \_value, \_\_1\_value.
* Examples of illegal identifiers: 123abc, -salary.

**Variables**

The variable is the basic unit of storage in a java program. A variable is defined by the combination of an identifier, a type, and an optional initializer. All variables have a scope, which defines their visibility and a lifetime.

Declaration: type identifier =value , identifier=value….

Int a, b , c; Int d=3, e, f=7;

**Method**

A Java method is a collection of statements that are grouped together to perform an operation.

modifier returnType nameOfMethod (Parameter List) {

// method body

}

public class ExampleTest {

public static void main(String[] args) {

string fullName = GetFullName (“firstname”, “lastname”);

System.out.println("Full Name: " + fullName);

}

public static string GetFullName(string fname, string lname) {

return fname+lname;

}

}

**Class and Object in Java**

**The Constructors**

A constructor initializes an object when it is created. It has the same name as its class and is syntactically similar to a method. However, constructors have no explicit return type.

import java.io.\*;

public class Employee {

String name;

double salary;

public Employee(String name) {

this.name = name;

}

public void empSalary(double empSalary) {

salary = empSalary;

}

public void printEmployee() {

System.out.println("Name:"+ name );

System.out.println("Salary:" + salary);

}

}

import java.io.\*;

public class EmployeeTest {

public static void main(String args[]) {

Employee empOne = new Employee("James Smith");

Employee empTwo = new Employee("Mary Anne");

empOne.empSalary(1000);

empOne.printEmployee();

empTwo.empSalary(500);

empTwo.printEmployee();

}

}

**The this keyword**

The **this** keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity. Using this we can refer the members of a class such as constructors, variables and methods.

**Arrays**

An array is a group of like-typed variables that are referred to by a common name. Arrays of any type can be created and may have one or more dimensions. A specific element in an array is accessed by its index. Arrays offer a convenient means of grouping related information.

type var-name[];

int month\_days[];

month\_days=new int[12];

declaration of the array variable with the allocation of the array itself

type var-name[]=new type[size]; **or** int month\_days[]=new int[12];

**Two dimension Array**

int[][] multi = new int[5][10];

which is a short hand for something like this:

int[][] multi = new int[5][];

multi[0] = new int[10];