**VARIABLES AND OOP CONCEPTS**

Local Variable: declared inside method (& is limited to that local scope only).

Class Test {

int a; // Instance variable

int b; // Instance variable

static int c; //static variable

Public void m1 (int a, int b) {

int d = 20; // local variable.

}

}

**BASIC OOP CONCEPTS:**

* Object: Instance of a class.
* Class: class is blueprint or virtual entity (or group of objects).
* Encapsulation: binding the data members & member functions as single unit. We’ll achieve by declaring properties or data members of a class as private.
* Abstraction: Hiding the internal logic (the code what we write inside a method) to the end user enables the services what can avail.

By using keyword ‘abstract’.

If I declare the method as abstract method, **then we don’t write body inside it**.

* **Abstract method won’t be having body**. Because we don’t have to show the functions to the end user.
* **Instance method has body**.
* Abstract class can have both **instance method** and **abstract method**.
* By using abstract class, we won't be able to achieve 100% abstraction because sometimes we can also have instance methods.
* All abstraction methods from abstract will be implemented in its child class.

**INHERITANCE:**

* Getting **properties (variables)**, **behaviors (methods)** of superclass to child class is called inheritance.
* And the keyword which we must use is **‘extends’**.

To call any method we can do is by creating object of that method.

* All **static methods and variables** belong to **classes**.
* All **instance members** **and methods** belong to **objects**.

**CONSTRUCTOR:**

Java constructors or constructors in Java is a terminology used to construct something in our programs. A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created.

* The purpose of constructor is to assign the values to instance variables.
* ‘new’ is for creating an object (allocating memory for objects).

public int m1 (int a, int b) {

return a+b;

}

Test t = new Test (10, 20);

t.m1 (30, 40) // **Calling of the method**.

**The purpose of the object is to call the constructor (Allocating memory for instance variables)**.

**KEY POINTS ABOUT ABSTRACT CLASSES**:

* We will achieve partial implementation by using abstract classes.
* We can have constructor inside abstract classes.
* Abstract classes will be having both abstract and instance methods.
* It is not possible to create objects for abstract classes.

**MULTIPLE INHERITANCE:**

* Multiple inheritance is the case when one child class tries to extend with multiple superclasses or trying to inherit from multiple superclasses or parent classes.
* Java doesn’t support multiple inheritance.

Class A {

public void m1 () {

SOP (“m1 method”);

}

}

Class B {

Public void m1 () {

SOP (“m1 method”);

}

}

Class C extends A, B {

public void m2 () {

SOP (“m2 method of C class”)

}

}

C c = new c ();

c.m1 (); // which class m1() method has to call that is the confusion or ambiguity case.

This is also called as **diamond problem**: In above example we have seen that “C” class is extending two classes “A” and “B” and its calling function “m1()” which is defined in both parent classes so now here it got confused which definition it should inherit.

Although Diamond Problem is a serious issue, we can create a solution for it which is Interface. Interface are created by using interface keyword. It contains all methods by default as abstract we don’t need to declare as abstract; compiler will do it implicitly. We can’t instantiate interface for this we have to use a class which will implement the interface and will write the definitions of its all functions.

**POLYMORPHISM:**

Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word “poly” means many and “morphs” means forms, so it means many forms.

**Method Overloading:**

Method Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters, or a mixture of both.

**Method Overriding:**

Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, the same parameters or signature, and the same return type (or sub-type) as a method in its super-class, then the method in the subclass is said to *override* the method in the super-class.