UNIT-03. Object-Oriented Programming Overview

What is Object Oriented Programming (OOP)?

- Object oriented programming (OOP) is a programming paradigm (Model) that depends on the concept of **classes and objects**.
- It is used to develop a software program which is simple, reusable using classes, which are used to create individual instances i.e. objects.
- There are many object-oriented programming languages like JavaScript, C++, C#, Java, PHP, Python etc.

Principles of OOP (OOP's Concepts):

- OOP totally depends on several OOP principles like class, object, data encapsulation, data abstraction, polymorphism, inheritance, message passing, Persistence etc.
- Let's see these principles in details-

1) Class:

- 'Class' is user defined data type which is collection of variables and methods.
- That is, these variables are called 'properties or attributes' and methods are called 'actions'
- Also, 'Class' is blue print for object because everything in class can also hold by object. i.e. class decides how the object was ?
- Since, all variables and methods are also available in objects, because they are created from class therefore they are called 'instance variable' and 'instance method'.

Consider, following defined class for 'person'.

```
class           person
{
         int      age;
        String   name;       //Properties-  i.e. instance variables of class

       void    talk( )    // action- i.e. instance method
      {
           System.out.println("Hi, My name is "+name);
           System.out.println("My age is "+age);
       }
}
```

In above, example *person* is class that consists of instance variables (properties) *age, name* and having instance method (action) *talk( )*.

2) Object:

- 'Object' is basic run time entity that holds entire data (variables and methods) of class.
- Object is also called as 'instance' of class.
- When object is created then that created object is stored in 'Heap area' by JVM.
- Also, whenever an object is created then JVM allocates separate memory reference (address) for created object called 'hash code'
- This 'hash code' is unique hexadecimal number created by JVM for object.
- We can also see *hash code* of every created object using *hashCode()* method of *Object* class of *java.lang* package.

Syntax to create object:

```
class_name      obj= new     class_name( );
```

Here, *class_name* is name of the class which is an identifier
obj is created object of class.

Purpose to create object:

There are following purpose to create the object:

1) To store entire data (variables and methods) of class.
2) To access variable or to make call for methods of class from outside class.

Following program demonstrate the use of *hashCode()* method.

```
Class       Demo
{
        public    static   void   main(String [ ] args)
        {
                Demo   a=new   Demo();
                Demo   b=new   Demo();
                System.out.println("First Object's Hash Code="+a.hashCode( ));
                System.out.println("Second Object's Hash Code="+b.hashCode( ));
        }
}

OUT PUT:
        First Object's Hash Code=53ab83406
        Second Object's Hash Code=1b6164678
```

3) Data Encapsulation:

- The binding of data and functions into single unit is called "Data encapsulation". And that keeps both data and functions safe from outside the world and misuse.
- Data encapsulation lead to the important OOP concept of data hiding and is accomplished by class. Because, by default all the data of class is 'default' therefore this data is only accessed within package and it is not accessed by outside the world (Other package).
- Java supports the properties of encapsulation and data hiding through the creation of user-defined types, called 'classes'
- A benefit of encapsulation is that it can reduce system complexity, and thus increases robustness, by allowing the developer to limit the interdependencies between software components.
- This feature supports the data security in OOP.
- Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding.
- To achieve encapsulation in Java −
  - ➢ Declare the variables of a class as private.
  - ➢ Provide public setter and getter methods to modify and view the variables values.

4) Data Abstraction:

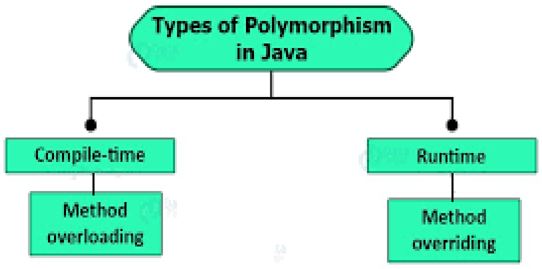
- Data abstraction refers to, providing only essential information to the outside word and hiding their background details.
- This is closely related to encapsulation because abstraction is implemented by encapsulation.
- In Java, classes provide great level of data abstraction. They provide sufficient public methods to the outside world to play with the functionality of the object and to manipulate object data. Without actually knowing how class has been implemented internally.
- If we have to give accessibility to some data of the class then it has to be placed in public section and those we have to make hide from outside world then it is placed in private section of class. Thus, Classes makes data abstraction with the help of access specifier or visibility mode.
- E.g.
- Let's take one real life example of a TV. Which you can turn on and off, change the channel, adjust the volume, and add external components such as speakers, VCRs, and DVD players.
- But you do not know its internal details such as,
  - ▪ you do not know how it receives signals over the air or through a cable
  - ▪ How it translates them, and finally displays them on the screen.
- Thus we can say, a television clearly separates its internal implementation from its external interface and you can play with its interfaces like the power button, channel changer, and volume control without having zero knowledge of its internals

5) Polymorphism:

- The word polymorphism is made up of two Greek words "poly" and "morphism".
- "Poly" means many and "morphism" means forms, so polymorphism means many-forms.
- In OOP, we implement many functions with same name but these functions are differ from each other according to their signature. (**Signature** refers to type of argument and number of argument accepts by the functions)
- In OOP, polymorphism means "one name" for multiple functions that have different behaviors.
- E.g. Consider following prototype of functions:

```
int     add( );
void    add(int);
int     add(int, int);
```

- In above example, many functions having same name but all these differ from each other by signatures therefore here polymorphism occurs.
- Following figure shows polymorphism with its types:



Basically polymorphism having two types:

1) Compile Time polymorphism (Static linkage/Static Binding/Early Binding):

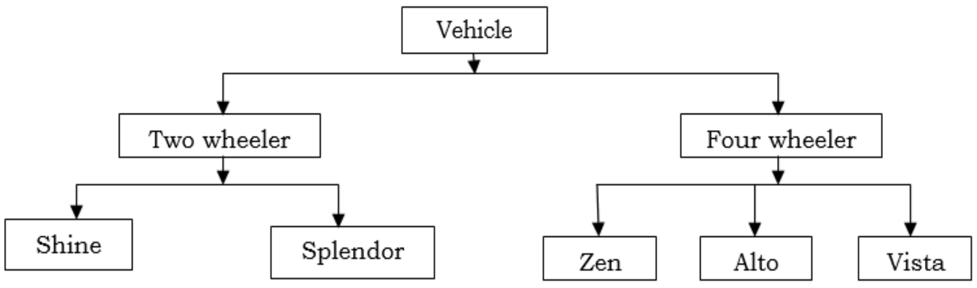
- Definition: Choosing the functions at compile time of program is called as "Compile time polymorphism or Static linkage/Static Binding/Early Binding"
- Compile time polymorphism is achieved by 'Method overloading'
- Note that: Java **does not** supports for 'operator overloading'

2) Run Time polymorphism (Dynamic linkage/Dynamic Binding/Late Binding):

- Definition: Choosing the functions at run time of program is called as "run time polymorphism or dynamic linkage/dynamic Binding/late binding"
- Runtime polymorphism is achieved by 'Method overriding"

6) Inheritance:

- Inheritance is a one of the most important feature of OOP, in which a class can inherits data members and member function (methods) of existing class.
- Inheritance is nothing but the process of deriving (Creating) new classes from existing class. The new class is called as 'Derived class or sub class or child class' where as existing class is called as 'Base class or Super class or parent class'.
- In inheritance, any number of classes can be linked with one another.
- Inheritance supports for "Is-A" relationship.
- While creating new class from existing one, the derived class accesses the data from base class and also add its own features into existing class without modifying base class. Thus, existing class is reused in derived class.
- The main concept behind inheritance is "Reusability" of code is possible. i.e. we use existing program again and again with addition of extra features.
- Inheritance reduces software developing time.
- Inheritance reduces software developing cost also.
- Following figure shows concept of inheritance:

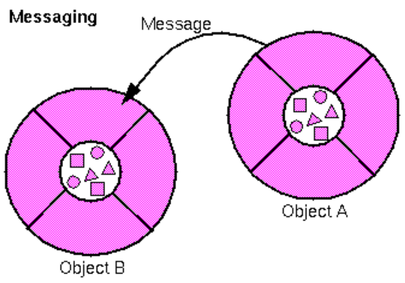


In above figure, 'Vehicle' is super base class from which two classes are derived 'Two wheeler' and 'Four wheeler'. Also, 'Shine' and 'Splendor' are derived classes from 'Two wheeler'. And 'Zen', 'Alto' and 'Vista' are derived classes from 'Four wheeler' base class.

7) Message Passing:

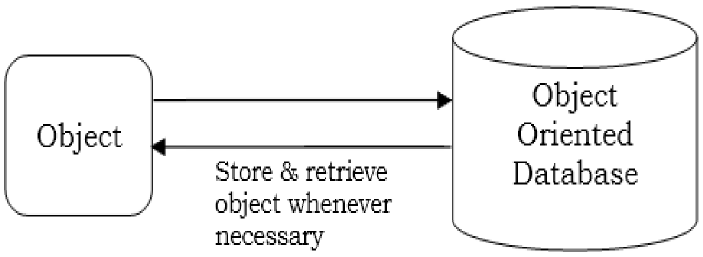
- The act of communicating with an object to get something done is called as "Message Passing".
- In message passing sending and receiving of information is done by the objects. Therefore it helps in building real life systems.
- Following are the basic steps in message passing.
    - ➢ Creating classes that define attributes and its behavior.
    - ➢ Creating objects from class definitions.
    - ➢ Establishing communication among objects.
- In OOPs, Message Passing involves the name of objects, the name of the function, and the information to be sent.
- Objects, which are usually instances of classes, are used to interact with one another to design strong applications.
- In OOP, each object is capable of **receiving messages**, processing data, and **sending messages** to other objects and can be viewed as an independent 'unit' with a distinct role or responsibility.
- Objects react when they receive messages by applying methods on themselves.
- A message is a request to an object to invoke one of its methods; in other words, a message for an object is a simply a call to one of its methods through the object.
- When objects communicate with one another, we say that they send and receive messages.
- E.g.   X.getdata(a,b);
- Here, we are passing a message getdata() to the objects 'X' with parameters 'a' and 'b'.

Following figure shows messaging between object A and Object B:



8) Persistence:

- In OOP, Persistence is simply related with the 'objects' that "Stick around" between the programs.
- This is just serialization (It is the process of translating an 'object' into a format that can be stored and reconstructed or retrieve later whenever required) an object from Object Oriented database (OO-database).
- In short, due to persistence concept, OOP language is capable to deal with the object oriented database.



Variable:

- Variable is the name given to the memory location where the value or data will be stored.
- Variable is such thing that has ability to store the value or data.

Types of Variable in Java:

There are three kinds of variable found in Java language:

1) Local Variable
2) Instance Variable (Non-static Variable)
3) Class Variable (Static Variable)

Let's see all variables in details:

1) Local Variable:

- The variable is declared within body any method is called 'Local Variable'.
- The scope of Local variable is limited to that method in which it is declared i.e. it is not accessible inside other method.

2) Instance Variable (Non-static Variable):

- The variable is declared within class body without using static keyword is called 'Instance Variable'.
- These variables are called instance variable because their content is in object of class.
- Such, instance variable can easily be accessed by class methods of same class in which it is declared.
- Instance variable's Separate copy is given all objects of class.
- Instance variables are stored at Heap area by JVM.
- Following table shows default values allocated by Java compiler to Instance variable.

Data Type	Default Value
byte	0
short	0
int	0
long	0
float	0.0
double	0.0
char	space
String	null
any class type (i.e. Object)	null
boolean	false

3) Class Variable (Static Variable):

- The variable is declared within class body with using static keyword is called 'Class Variable' or 'Static variable'.
- Static variables are common to all objects of class i.e. Common copy of static variable is shared among all objects of class.
- These variables are called class variable because they are in class scope.
- Such, class variable can easily be accessed by class methods of same class in which it is declared.
- Class variables are stored at method area by JVM.
- Following program shows different java varialbes-

```
class          person
{
        static    int      age;     //class variable
        String   name;       //instance variable
       void    talk( )    // action- i.e. instance method
      {
           int    p;     // Local variable
           System.out.println("Hi, My name is "+name);
           System.out.println("My age is "+age);
       }
}
```

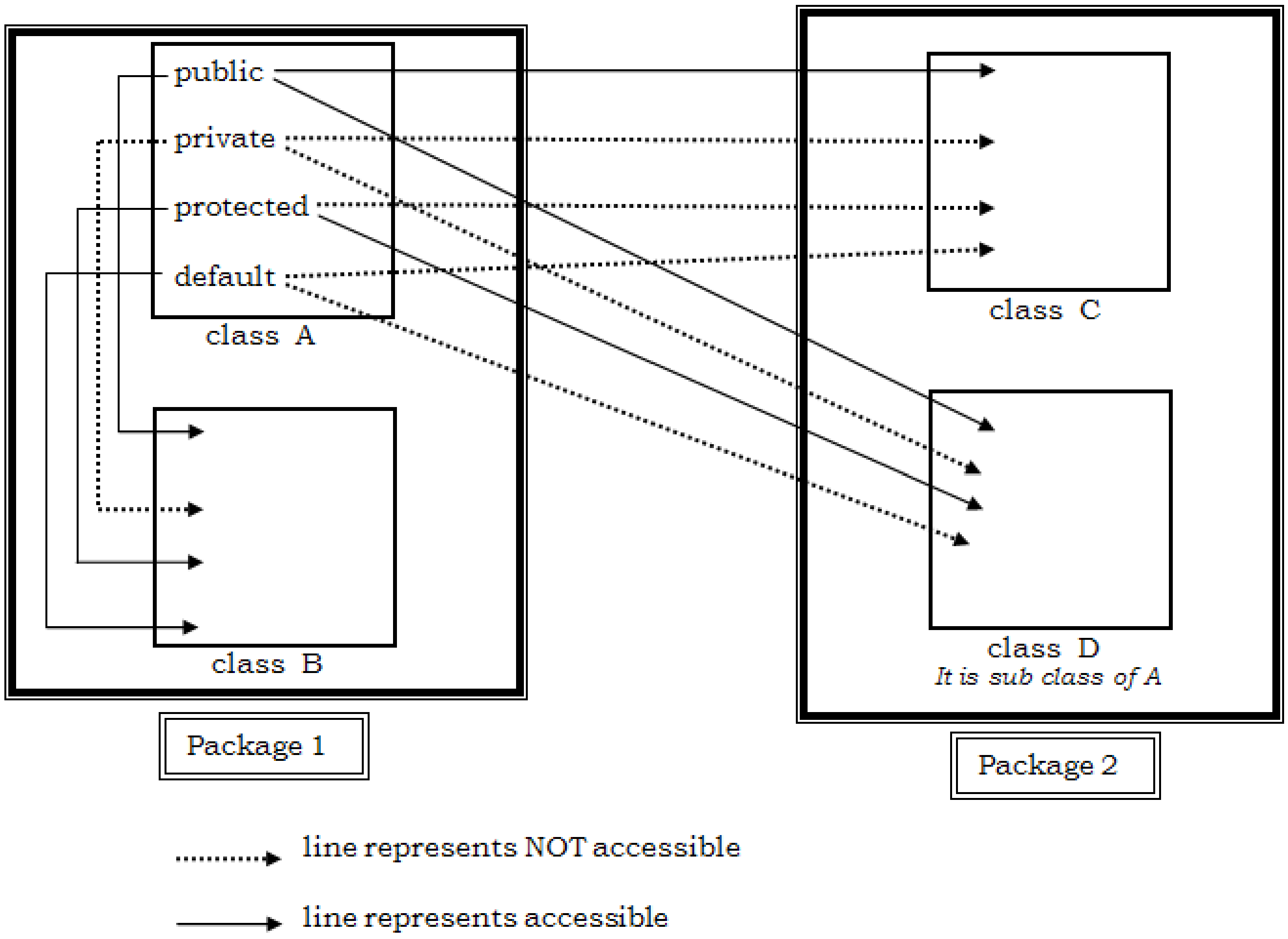
Access Specifier (Member Access Controls) in Java:

- Access Specifier is a keyword that specifies, How to access the members of class or a class itself.
- We can also, use access Specifier before a class and its members.

There are 4 access specifier in Java language VIZ.

1) public 2) private 3) protected 4) default

Following figure shows different access Specifier with their accessibility:



Let us see all these access Specifier in details:

1) public:

- 'public' members of class are <u>accessible everywhere therefore they have **global scope**</u>.
- They are accessible:
  1) Within methods of same class.
  2) Within methods of sub classes.
  3) Within methods of other class of same package.
  4) Within methods of class of other package.

2) private:

- 'private' members of class are only accessible by methods of same class therefore they have class scope. That is private members are not accessible outside the class.
- They are accessible:
  1) Within methods of same class.
- They are NOT accessible:
  1) Within methods of sub classes.
  2) Within methods of other class of same package.
  3) Within methods of class of other package.

3) protected:

- The accessibility of 'protected' members of class is given as follow;
- They are accessible:
  1) Within methods of same class.

2) Within methods of sub classes.
3) Within methods of other class of same package.
4) Within methods of classes of other package (Since, other package class should be sub class)

- They are NOT accessible:

  1) Within methods of classes of other package (If other package class should **NOT** be sub Class)

4) default:

- If no access specifier is written by the programmer then, Java compiler uses ‘default’ access specifier.
- The ‘default’ members of class should be accessible by the methods of class of same package i.e. they are not accessible out the package. Therefore ‘default’ access specifier has ‘package’ scope.
- They are accessible:
  1) Within methods of same class.
  2) Within methods of sub classes.
  3) Within methods of other class of same package.
- They are NOT accessible:

  1) Within methods of classes of other package.

Methods in Java:

In Java, there are two kinds of methods found;
1) Instance Methods (Non-Static Methods)   2) Class Methods (Static Methods)
Let us see all these methods in details:

1) <u>Instance Methods (Non-Static Methods):</u>

- The method which is defined within class body <u>*without*</u> using keyword ‘static’ are called as ‘Instance methods or Non-static methods’.
- These types of methods are act on ‘instance variable’ of class.
- They are called ‘Instance’ method because their content is in <u>instance (object) of class</u>.
- Instance methods are called with the help of <u>object of class</u> using syntax:

  object.method(arg1,arg2, ----);
- One specialty of instance method is that they are capable to access <u>instance variable</u> (non-static variable) and <u>class variable</u> (static variable) <u>directly</u>.

Types of Instance Methods:

Further, Instance methods are of two types:
1) Accessor Method   2) Mutator Method

Let’s see these methods in details:

1) Accessor Method: (Getter Method)

- This type of instance methods are <u>*only* access or read instance variables</u> i.e. <u>they do not modify</u> value of instance variable are called as “Accessor Method”.
- This method is also called the <u>Getter method</u>. Because, this method returns the variable value.

2) Mutator Method: (Setter Method)

- This type of instance methods are access or read instance variables and also modify value of instance variable are called as “Mutator Method”.
- This method is also called the <u>Setter method</u>. Because, this method sets the variable value.

Following program shows the use of Accessor and Mutator methods.

```
class    Access
{
        int     age;
        String  name;
        int     getAge( )  //mutator method
        {
                return(age);
        }
        String  getName()   //mutator method
        {
                return(name);
        }
        void   setAge(int      x)    //Accessor method
        {
              age=x;
        }
void  setName(String   nm)     //Accessor method
{
                name=nm;
}
public  static  void  main(String  [ ] args)
{
        Access   t=new   Access();
        t.setAge(55);
        t.setName("RAHUL");
        System.out.println("Age="+t.getAge());
        System.out.println("Name="+t.getName());
        }
}
```

2) Class Methods (Static Methods):

- The method which is defined within class body using keyword 'static' is called as 'class methods or static methods'.
- These types of methods are act on 'class variable (static variable)' of class.
- They are called 'Class' methods because they are defined using 'static' keyword & it is related with class.
- Class methods are called with the help of class name using syntax:

  Class_Name.method(arg1,arg2, ----);
- Class methods are capable to access only class variable (static variable) directly.
- Still, if we want to access instance variable inside class method then it can be accessed only by using object of class.

Following program shows use of class method:

```
public   class      Empolyee
{
    static  int       no;
    static  String   name;
    static  void  get()
    {
         no=65;
         name="KIRAN";
    }
    static   void   show()
    {
         System.out.println("Number="+no);
         System.out.println("Name="+name);
    }
   public   static   void   main(String  []args)
         {
                  Empolyee.get( );              //class methods accessed with class name.
                  Empolyee.show( );
         }
}
```

(**HOME WORK**: *Write Difference between Instance Methods and Class Methods)*

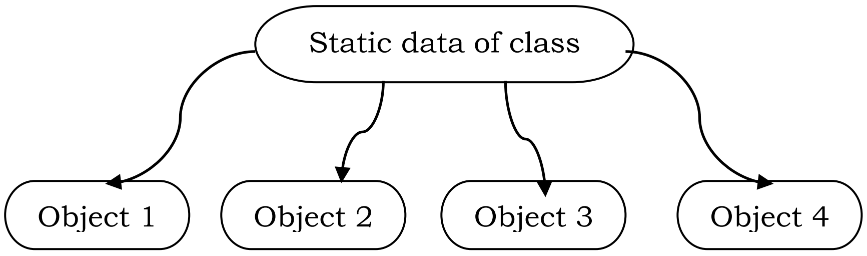
Static data:

- We know that, for single class we can create multiple objects. And each object holds their individual record. But there are some data which are common to all objects in such situation that data can be made as static.
- Once the data is made as static then this data is common to all objects of class.
- In short, static data is common data and which shared by all objects of class.
- Only one copy of static data members is maintained for entire class therefore it is shared among all objects.
- Static data by default initialized to <u>zero</u>.
- Static data are initialized at <u>once therefore it is used like a counter</u>.
- <u>Static data are stored separately rather than as a part of an object</u>.
- Syntax to <u>declare</u> static variable:

```
static      datatype     variable;
```

Here,	'static'	is keyword which is used to declare member as static
	'datatype'	is any valid data type in Java.
	'variable'	is name of static variable.

Following fig. shows sharing of static data by different objects of class:



In above fig. static data of class is common to all objects Object1, Object2, Object3, Object4. Or the same copy of static data is shared among all objects.

Following program demonstrate the use of static variable that counts total object created for class:

```
class    count
{
          static   int      cnt;
          count( )
          {
              cnt++;
          }
          static void show( )
          {
              System.out.println("Total Objects= "+cnt);
          }
public    static   void   main(String  [ ]args)
{
                    count  a=new count( );
                    count  b=new count( );
                    count  c=new count( );
                    count  d=new count( );
                    count.show( );
}
}
OUTPUT:       Total Objects= 4
```

Method Overloading:

- Method overloading is type of compile time polymorphism where we can take or define many methods having same name but all these methods are differ from each other according to their signature (type of argument and number of argument accept by method).
- It is type of compile time polymorphism, hence all overloaded methods are get selected by JVM at compile time of java program.
- Note that: When we overload the methods then all overloaded methods functionality must be same.
- Following program shows method overloading that finds addition of two numbers:

```
public    class    MethodOverload
{
    int   a,b,c;
    void   add( )
    {
          a=5;
          b=7;
          c=a+b;
          System.out.println("Addition="+c);
    }
    void   add(int   x)
    {
          a=10;
          c=a+x;
          System.out.println("Addition="+c);
    }
    void    add(int    x,int   y)
    {
          c=x+y;
          System.out.println("Addition="+c);
    }
    public   static   void   main(String    []args)
    {
          MethodOverload     p=new MethodOverload();
          p.add();
          p.add(15);
          p.add(45,65);
    }
}
```

Constructor:

- The constructor is special method whose name is same as that of class name.
- The main task of constructor is to initialize values for object of its class.
- It is called constructor because it construct the values for object of the class.
- The constructors are implicitly invoked by the compiler whenever an object of class is created.
- A constructor is implicitly called whenever the object of class is created.
- Example:

Consider, there is one class having name “person” then its constructor is given as:

```
class              person
{
          person( )        // constructor
          {
                    -------
                    -------
          }
}
```

Properties/Characteristics/Features of Constructor:

1) Constructor name is same as that of class name.
2) Constructor does not return any value but it can accept some parameter as input.
3) Constructor does not have any return type even void.

4) Constructors are implicitly invoked by the compiler whenever object of class is created.
5) Constructor can be overloaded.
6) Constructors are not static.

Types of constructor:

Depending upon arguments passed to constructor, it has two types.

1) Default Constructor
2) Parameterized constructor

NOTE:

1) *Java language does not support for copy constructor.*
2) *If we specify return type for constructor then Java compiler treats that method as normal method.*

1) Default constructor:

- The constructor that does not have any argument or parameter as input, such constructor is called as "Default Constructor".
- Default constructor is implicitly invoked by compiler whenever object of class is crated

Syntax:

```
class      class_name
{
              class_name( )        // default constructor.
              {
                     -------
                     -------
              }
}
```

Following program shows implementation of default constructor:

```
class      person
{
   int       age;
   String    name,add;
   double    sal;
    person( )    // default constructor
   {
      age=70;
      name= "Sachin";
      add= "Pune";
      sal= 4568.50;
    }
          void  show( )
          {
              System.out.println("Age="+age);
              System.out.println("Name="+name);
              System.out.println("Address="+add);
              System.out.println("Salary="+sal);
          }
public   static  void    main(String [ ] args )
{
        person   x=new person( );   //call to def. constructor
        x.show( );
 }
}
```

2) Parameterized constructor:

- The constructor that accepts any argument or parameter as input, such constructor is called as "parameterized Constructor".
- Parameterized constructor is implicitly invoked by compiler when we pass some values direct to object at the time of its creation.

Syntax:

```
class      class_name
{
             class_name( int   x, String  y)        // parameterized constructor.
              {
                     -------
                     -------
              }
}
```

Following program shows implementation of parameterized constructor:

```
class      person
{
   int       age;
   String    name,add;
   double    sal;
void  show( )
{
   System.out.println("Age="+age);
   System.out.println("Name="+name);
   System.out.println("Address="+add);
   System.out.println("Salary="+sal);
 }
```

```
   person(int  a, String  n ) //para.constructor
   {
      age=a;
      name=n;
      add= “Pune”;
      sal= 4568.50;
    }
```

```
public   static  void     main(String [ ] args )
{
    person   x=new person(45, “Raj” );  //call to parameter constructor
    x.show( );
 }
}
```

Overloading of Constructor (Multiple Constructor in Class):

- We know that in case of Method overloading, we can define or implement many methods with same name <u>within same class</u>, but all these methods are differ from each other according to their signature (Type of argument and number of argument accept by methods).
- Like method overloading, we are also able to overload the constructor.
- Constructor overloading means implementing or defining multiple constructors for single class. But all these constructors are differing from each other according to their signatures.
- Following program shows constructor overloading:

```
class     person
{
   int       age;
   String   name,add;
   double  sal;

    person( )    // default constructor
   {
      age=70;
      name= “Sachin”;
      add= “Pune”;
      sal= 4568.50;
    }
  person(int  a, String  n ) //para.constructor
  {
      age=a;
      name=n;
      add= “MUMBAI”;
      sal= 54508.75;
  }
```

```
 void  show( )
 {
   System.out.println(“Age=”+age);
   System.out.println(“Name=”+name);
   System.out.println(“Address=”+add);
   System.out.println(“Salary=”+sal);
  }
public   static  void     main(String [ ]  args )
{
    person   p=new  person( );
    person   q=new person(45, “Raj” );
    p.show( );
    q.show( );
 }
}
```

What is the destructor in Java?

- We know that, destructor is a method that destroy (release the memory) the object. It is opposite of constructor.
- It is a special method that automatically gets called when an object is no longer used (i.e. destroyed)
- When an object completes its life-cycle the <u>garbage collector automatically deletes that object and de-allocates or releases the memory occupied by the object</u>.
- In Java, automatically memory management is done by garbage collector and hence Java language does not have concept of destructor. But Java language has finalize( ) method whose work is same as that of destructor.
- For automatically memory management, JVM implicitly called gc( ) method. (we called it explicitly as System.gc())

finalize( ) Method:

- It is difficult for the programmer to forcefully execute the garbage collector to destroy the object.
- But Java provides an alternative way to forcefully execute the garbage collector to destroy the object.
- The Java *Object* class provides the *finalize()* method that works the same as the destructor.
- It is a protected method of the Object class that is defined in the *java.lang* package.
- It can be called only once.
- We need to call the finalize() method explicitly if we want to override the method.
- The syntax of the finalize() method is as follows:

Syntax:

```
        protected void finalize ( )
        {
            //resources to be close
        }
```

Following program shows use of finalize( ) method:

```
public  class  DestructorDemo
{
        DestructorDemo()
       {
           System.out.println("This is Constructor");
        }
public  static  void  main(String[] args)
{
     DestructorDemo de = new   DestructorDemo();
     de.finalize();
}
protected  void  finalize()
{
     System.out.println("Object is destroyed ...");
}
}
```

Parameter passing technique in Java:

- In JAVA, when we pass any variable or any object or even object-reference to method then they are passed as ‘Pass by value’ or ‘Call by value’
- That is, Default parameter passing technique in Java is ‘Pass by value’ or ‘Call by value’.
- Pass by address (Pass by pointer) or call by address (Call by pointer) is invalid in Java, because Java does not support for pointer.
- Everything is passed to method in Java by ‘Pass by value’ or ‘Call by value’ concept.

Passing Object as ‘pass by value’:

- We know that, everything is passed to method in Java by ‘Pass by value’ or ‘Call by value’ concept whether it is normal variable or any objet or any object-reference.

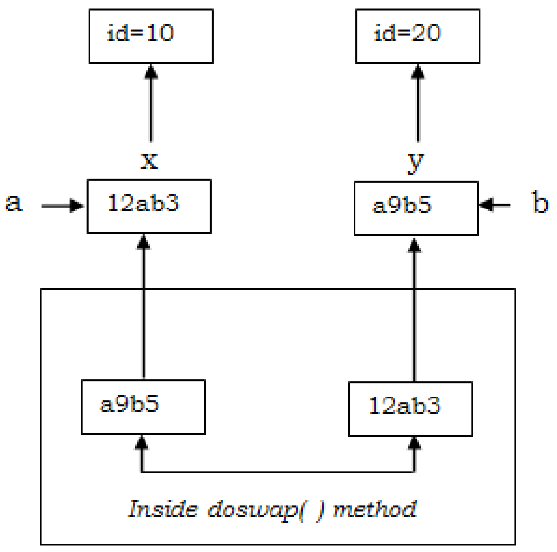
Consider, following program that illustrate passing object as pass by value.

```
class    emp
{
            int   id;
            emp(int   k)
            {
                        id=k;
            }
}
class    ParaPass
{
            emp   c;
            void    doswap(emp  a,emp   b)
            {
                        c=a;
                        a=b;
                        b=c;
            }
```

```
public   static   void   main(String   [ ]sd)
   {
            emp   x=new   emp(10);
            emp   y=new   emp(20);
            ParaPass   p=new   ParaPass();
            System.out.println("Before swapping ID's= "+x.id+"    "+y.id);
            p.doswap(x,y);
           System.out.println("After swapping ID's=  "+x.id+"    "+y.id);
   }
}

OUT PUT:
Before swapping ID's=10      20
After swapping ID's= 10    20
```

- In above program, we pass two objects ‘*x’, ‘y’* of ‘*emp’* class to *doswap( )* method. Here, objects are pass by value. These passed objects are stored in corresponding formal objects ‘*a’* and *‘b’*.
- We do some swapping mechanism over objects *‘a’* and *‘b’* inside *doswap( )* method that does not effects on values of *‘id’* hold by objects *‘x’* and *‘y’* because they are passed as pass by value.
- Here, we interchange the objects that ***interchange the references*** of objects. That’s why we got same output corresponding to input i.e. no interchanging of *‘id’* is done.
- This is shown in following figure.



Static Block:

A static block is block of statements declared with keyword 'static', something like this:

```
static
{
        ------------
        ------------
        ------------
}
```

- *Static block has highest priority* therefore JVM interprets this block first even before main( ) method also.
- In, Java interpretation, JVM first searches for main( ) method, if main( ) found and also static blocks are found then execution of static blocks are done first and then main( ) method is executed.
- If JVM not found main( ) method in java source program then this source code will not execute.
- Also, Single Java program may contains multiple static blocks; in such situation JVM executes them one after another in sequence (FIFO) manner.

Following program demonstrate use of static block.

```
class     Test
{
    static
    {
       System.out.println("First");
    }
   public  static  void  main(String  [ ] as)
   {
     System.out.println("Main Method");
   }
   static
    {
       System.out.println("Third");
    }
    static
    {
       System.out.println("Second");
    }
 }
OUTPUT:
          First
          Third
          Second
          Main Method
```

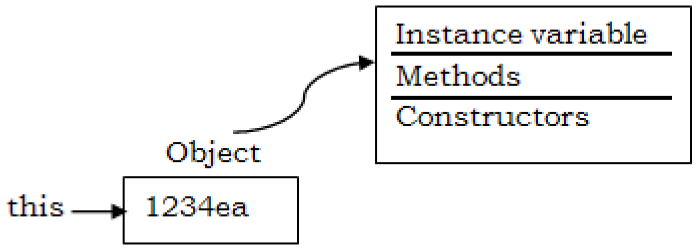
Note:

Java program may be compile and run without main( ) method with presence of static block.

‘this’ keyword in Java:

- When an object is created to a class, then *JVM implicitly create default reference to created object* and that *default reference* is called ‘this’.
- ‘this’ keyword always refers to *current object of class*.
- ‘this’ is non-static therefore it *cannot use within static method or static block*.
- ‘this’ is hidden parameter for every non-static method of class.
- Generally, we write instance variable, methods and constructors in a class. All these members are stored in object. As well as *all these members are also refers by ‘this’ reference*.

‘this’ reference is shown in following figure:



Use of ‘this’:

We know that ‘this’ is reference for current object of class therefore it is used like object i.e.

1) ‘this’ is used to invoke default constructor of present class using syntax: **this( );**
   Also, we made call to parameterized constructor of present class using syntax: **this (arg1, arg2, ...);**
2) ‘this’ is also used to call instance methods of class using syntax: **this.method(arg1,arg2,...);**
3) ‘this’ is also used to access instance variable of class using syntax: **this.variable;**
4) When instance variables of class and formal parameters of methods are same then that creates confusion. To solve this confusion ‘*this’ keyword is used to access instance variables* of class.

Following program shows use of ‘this’ keyword:

```
class     sample
{
          int       age;
          String   name;
          sample( )
          {
                    this(45,"Sachin");          //call to parameterized constructor
          }
          sample( int    age, String     name)
          {
                    /* Instance variable & formal parameters are same. Therefore 'this' is used to access instance variable*/
                    this.age=age;
                    this.name=name;
          }
          void    show( )
          {
                    System.out.println("Age="+age);
                    System.out.println("Name="+name);
          }
          public   static   void main(String    [ ]args)
          {
                    sample     p=new   sample( );
                    p.show( );
          }
}
```

OUTPUT:

```
          Age= 45
          Name=Sachin
```

Final class in Java: (We will discuss in Inheritance)

Object class in Java:

- Object class is present in *java.lang* package.
- 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 inheritance hierarchy in any Java Program.
- The Object class is the parent class of all the classes in java by default. In other words, it is the topmost class of java.
- The Object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference can always refers the child class object, known as “upcasting”
- Let's take an example, there is getObject() method that returns an object but it can be of any type like Employee, Student etc, we can use Object class reference to refer that object. For example:

  Object obj=getObject();   //we don't know what object will be returned from this method
- The Object class provides some common behaviors to all the objects such as object can be compared, object can be cloned, object can be notified etc.

Methods of Object class

- The Object class provides many methods. They are as follows:

Method	Description
public final Class getClass()	returns the Class class object of this object. The Class class can further be used to get the metadata of this class.
public int hashCode()	returns the hashcode number for this object.
public boolean equals(Object obj)	compares the given object to this object.
protected Object clone() throws CloneNotSupportedException	creates and returns the exact copy (clone) of this object.
public String toString()	returns the string representation of this object.
public final void notify()	wakes up single thread, waiting on this object's monitor.
public final void notifyAll()	wakes up all the threads, waiting on this object's monitor.
public final void wait(long timeout) throws InterruptedException	causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait(long timeout,int nanos)throws InterruptedException	causes the current thread to wait for the specified milliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait() throws InterruptedException	causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).
protected void finalize()throws Throwable	is invoked by the garbage collector before object is being garbage collected.

Garbage Collection:

- Garbage collection in Java is a process by which Java programs perform automatic memory management. We know that when Java programs compile it would create byte code and that can be interpret by JVM. When Java programs run on the JVM, objects are created on the heap, and memory is allocated to each object. Eventually, some objects will no longer be needed. Then garbage collector finds these unused objects and deletes them to free up memory.
- Garbage Collection is process of retrieving the runtime unused memory automatically. In other words, garbage collection is a way to destroy the unused objects and free the memory.

- To do so, we were using free() function in C language and *delete* in C++. But, in java it is performed automatically. So, java provides better memory management.
- In C/C++, a programmer is responsible for both the creation and destruction of objects. Usually, programmer neglects the destruction of useless objects. Due to this negligence, at a certain point, sufficient memory may not be available to create new objects, and the entire program will terminate abnormally, causing OutOfMemoryErrors.
- But in Java, the programmer need not care for all those objects which are no longer in use. Garbage collector destroys these objects.
- The main objective of Garbage Collector is to free heap memory by destroying unreachable objects.
- The garbage collector is the best example of the Daemon thread as it is always running in the background.

How Does Garbage Collection in Java works?

- Java garbage collection is an automatic process. Automatic garbage collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects.
- An in-use object, or a referenced object, means that some part of your program still maintains a pointer to that object. An unused or unreferenced object is no longer referenced by any part of your program. So the memory used by an unreferenced object should be destroyed.
- The programmer does not need to mark objects to be deleted explicitly. This task automatic done by garbage collector.

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.

Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects)

Theory Assignment No: 03

1) Explain following OOP's concepts:
   1) Class 2) Object 3) Data Encapsulation 4) Data Abstraction 5) Polymorphism
   6) Inheritance 7) Message Passing 8) Persistence
2) Explain member access controls (Access Specifier) in Java.
3) What is Variable? Explain types of variable.
4) Explain Java methods with types.
5) What is method overloading?
6) What is constructor? Explain types of constructor with example.
7) Write short note on:
   1) finalize( ) method 2) static block 3) 'this' keyword 4) Parameter passing technique
   5) Garbage collection

Practical Assignment No: 05

1) Write a program to overload method that calculate area of triangle.
2) Write a program that demonstrate use of getter and setter method.
3) Write a program to implement default constructor.
4) Write a program to implement parameterized constructor.
5) Write a program for constructor overloading. (Multiple constructor in a class)
6) Write following programs by using Constructor.
   a) Find the largest and smallest number from the array.
   b) Swap two numbers by using parameterized constructor.
   c) Find the sum of diagonal elements of the matrix.
   d) Print the Fibonacci series up to 'n' numbers.
7) Write a program that demonstrate use of finalize( ) method.
8) Write a program that demonstrate use of 'this' keyword.
9) Write a program that demonstrate use of static block.
10) Write a program that demonstrate passing object as pass by value.