Chapter 3: Classes, Objects and Methods in java:

3.1 Class and Object in java:

We will learn about java objects and classes. In object-oriented programming technique, we have design a program using objects and classes. Object is the physical as well as logical entity and class is the logical entity only.

A class is nothing but just a blueprint for creating different objects which defines its properties and behaviours. Java class objects exhibit the properties and behaviours defined by its class. A class can contain fields and **methods** to describe the behaviour of an object.

General syntax of define class in java

```
class-modifiers class class-name extends clause  implements clause
{
        Declaration variables;(static and non-static members)
        Declaration method();(static and non-static members)
        Declaration inner class;
        Declaration nested interface
        Declaration Constructor()
        Declaration Main Method();
}
```

Below is an example showing the Objects and Classes of the area class that defines 2 fields namely length and breadth. Also the class contains a member function getArea().

```
public class area
{
        int length;
        int breadth;

        public int getArea ()
        {
                return (length * breadth);
        }
}
```

Simple Example of Object and Class:

In below example, we have created a Member class that have two data members' id and name. We are creating the object of the Member class by new keyword and printing the objects value.

Example 1:

```
class Member
{
    int id;                 //data member (also instance variable)
   String name;             //data member(also instance variable)

public static void main(String args[])
```

```
    {
            Member m1=new Member1();        //creating an object of Student
            System.out.println(m1.id);
            System.out.println(m1.name);
      }
}
```

Output:
0
Null

Example 2: Now following example shows the use of method.

```
public class Demo
{
        private inta,b,c;
        public void input()
          {
        a=10;
        b=15;
          }
        public void sum()
          {
        c=a+b;
          }
        public void display_data()
          {
             System.out.println("Answer is =" +c);
          }
        public static void main(String args[])
          {
            Demo object=new Demo();
            object.input();
            object.sum();
            object.display_data();
           }
}
```

Output:
Answer is =25

Here In above program, the statement **Demo object=new Demo ();** is used for create object of Demo class. Now then we can access class members through this created object using dot operator. Here following these are member methods,

```
object.input();
object.sum();
object.print_data();
```

In the first line we created an object, and then these three methods are called by using the dot operator. When we call a method the code inside its block is executed. The dot operator is used to call methods or access them.

❖ **Creating "main()"method in a separate(different) class**

We can create the main method in a separate class, but during compilation you need to make sure that you compile the class with the main() method.

Example 1:

```
public class Demo
{
        private inta,b,c;
        public void input()
        {
        a=5;
        b=15;
        }
        public void sum()
        {
        c=a+b;
        }
        public void display_data()
        {
           System.out.println("Answer is =" +c);
        }
}

publicclass SumDemo
{
public static void main(String args[])
   {
        Demo object=new Demo (); //object of Demo class created from here.
        object.input ();      //input method of Demo class called from here.
        object.sum();
        object.display_data();
   }
}
```

Output:

Answer is 20

- **Access the Member variables by using dot operator:**
  We can access the variables by using dot operator. Following program shows the use of dot operator.

Example 1:

```
class Demo
{
inta,b,c;
publicvoid sum()
   {
c=a+b;
   }

public void show()
   {
```

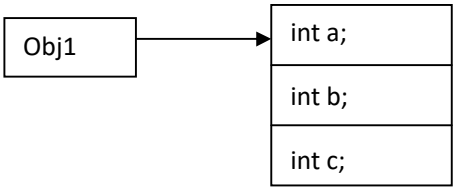
```
        System.out.println("The Answer is "+c);
    }
}
class SumDemo
{
        publicstaticvoid main(String args[]){
        Demo obj1=new Demo();
        Demo obj2=new Demo();

        obj1.a=20;              //assign 20  value of var a for object 1
        obj1.b=25;              //assign 25  value of var b for object 1
        obj2.a=50;              //assign 50  value of var a for object 2
        obj2.b=30;              //assign 30  value of var b for object 2
        System.out.println("call Sum Method and Show Method for Object 1");
        obj1.sum();
        obj1.show();
        System.out.println("call Sum Method and Show Method for Object 2");
        obj2.sum();
        obj2.show();
    }
}
```

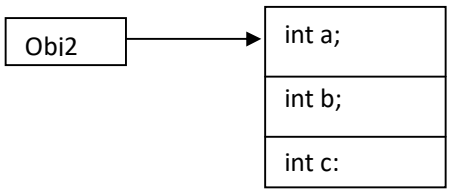
Output:

```
call Sum Method and Show Method for Object 1
The Answer is 45
call Sum Method and Show Method for Object 2
The Answer is 80
```

See below figure is represented graphically for how to memory occupy by obj1 and obj2.



(a) Memory occupy by object1



(b) Memory occupy by object2

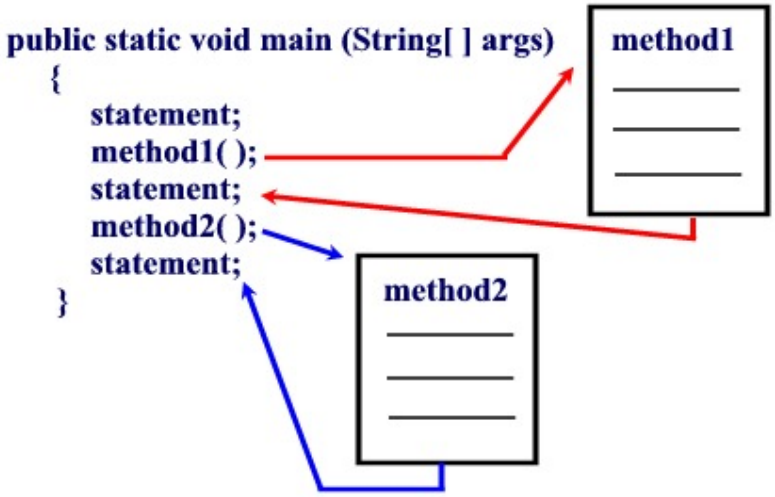
❖ **Instance Variable:**

In above, all variables are also known as instance variable because of that each instance or object has its own copy of values for the variables.Hence other use of the "dot (.)" operator is to initialize or access them to the value of variable for that instance.

3.2 Method(s) in java:

A method is a set of code which is referred to by name and can be called (invoked) at any point in a program. Think of a method as a subprogram that acts on data and often returns a value. Each method has its own name. Then that name is encountered in a program, the execution of the program branches to the body of that method. When the method is finished,

execution returns to the area of the program code from which it was called, and the program continues on to the next line of code.

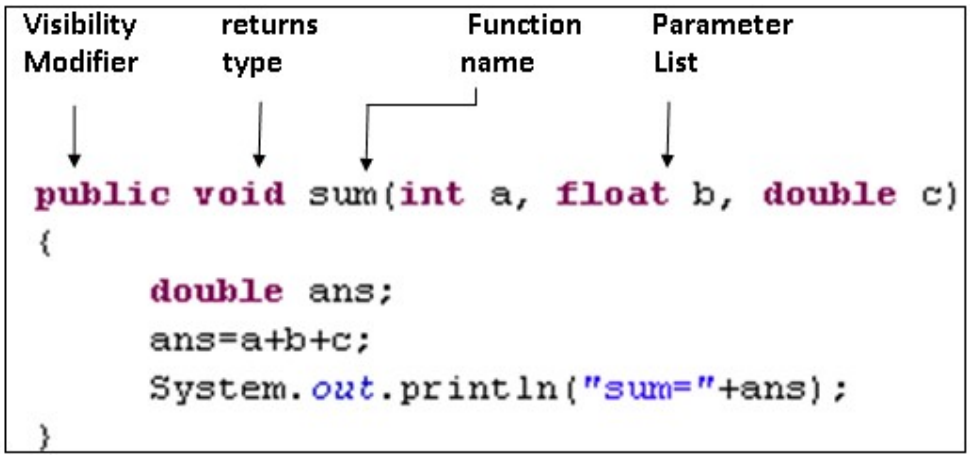


There are two basic types of methods:

(A) Built-in :Built-in Method are part of the complier package,
Example: System.out.println () and System.exit (0).

(B) User Define: User defines Method are created by user or programmer. These methods takes on name that you assign to them and perform tasks that programmer created.

❖ **The Structure of a Method**



The method visibility modifier goes first. Here in above example it is **public**. Then method's **return type** goes second, which is **void** type in the code above. After the method type, you need a space followed by the name of your method. We've called the one above **sum**. In between a pair of round brackets we have told Java that is parameter List. Here we have used three different variable which names are a, b and c respectively.

To separate this method from any other code, you need a pair of curly brackets. Your code for the method goes between the curly brackets. Then here we have used those variables which are passed in the parameter. And remember it here we have not return any value so we do not wrought **return** statements.

If sometimes we have needed something to return from our method. Then see below figure.

```
public double sum(int a, float b, double c)
{
      double ans;
      ans=a+b+c;
      return (ans);
}
```

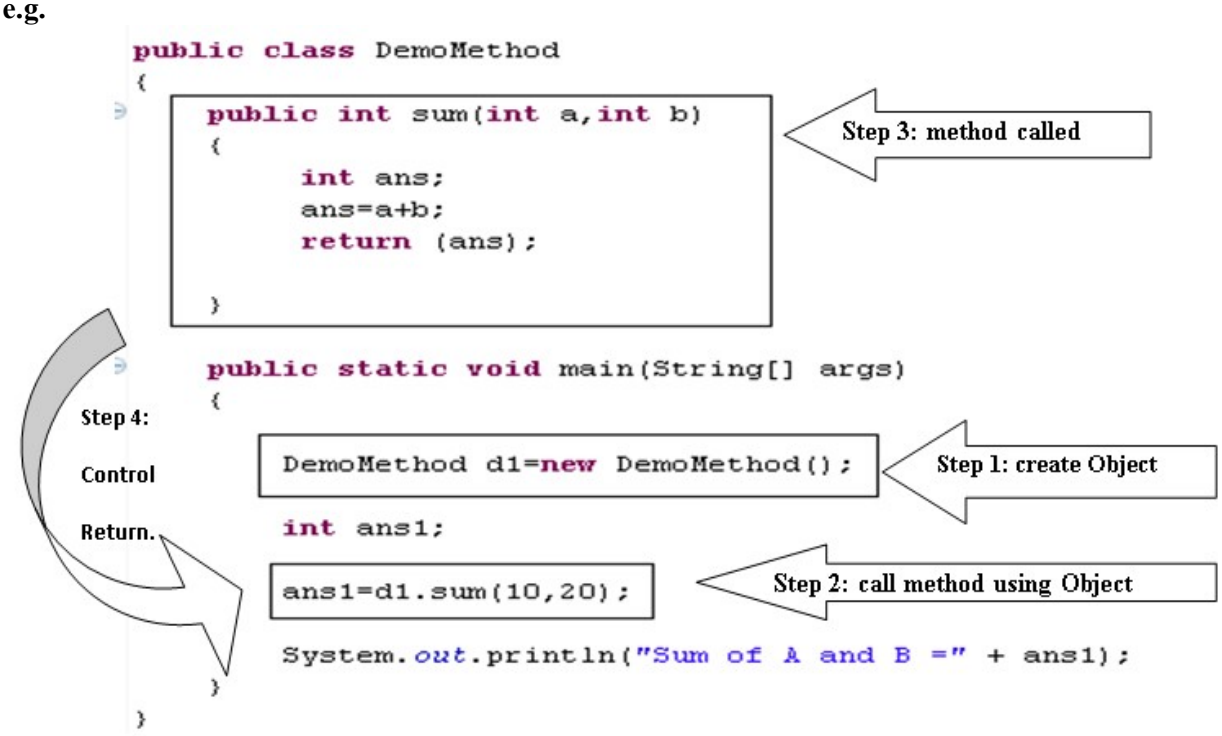
❖ **How to Create an Object and Called User define Method:**

Before we called java methods, we need an Object. So first we have need creation of an Object. As you know, a class provides the blueprint for objects; you create an object from a class.

Syntax:

```
class_name object_name=new class_name();
```

e.g.



Let's now we will discuss about all four step see in above figure.
Step 1: create an Object. Here class name is DemoMethod and d1 is an Object name.
Step 2: this step is indication of how to call method using an Object. Here sum is method name and values 10 and 20 are as parameters.
Step 3: this step is indicated that is a Method body. When we called user define method using an object then control goes on function body.
Step 4: after finished execution of all statements in method body then control will back to main method with return value.
Return value is not compulsory. It is depend on user logic.

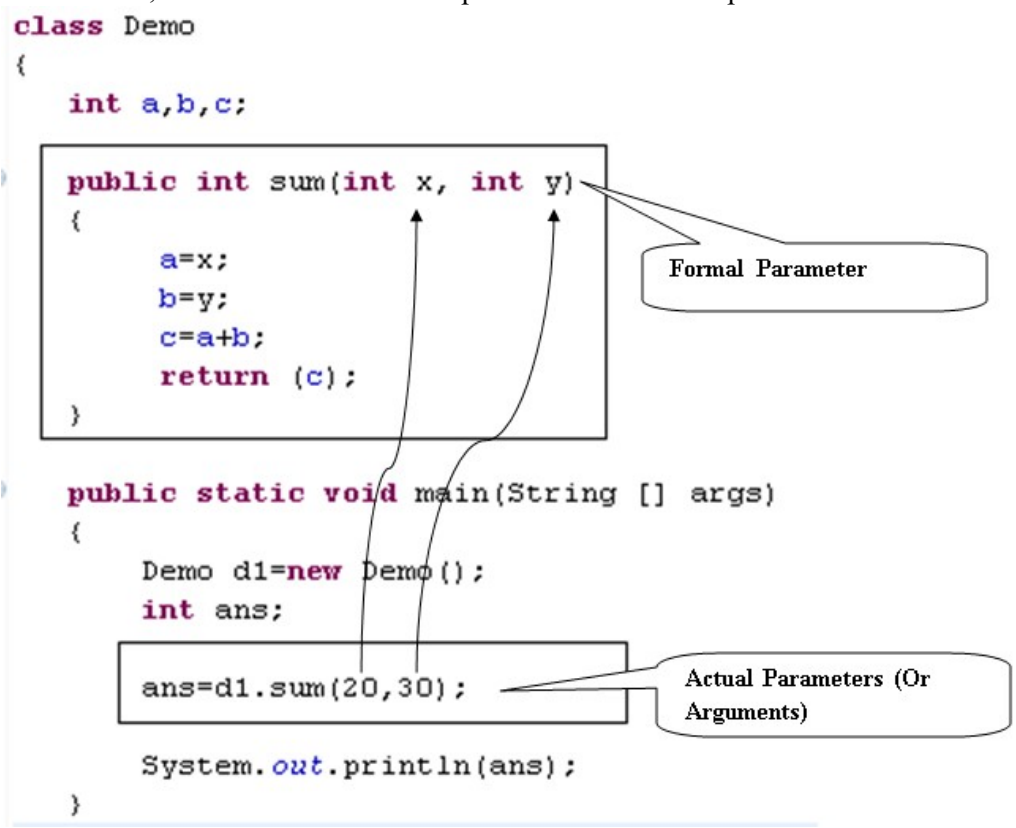
❖ Method with parameters:

We have passed value(s) as an argument(s) at method calling time it's known as method with parameter(s).we have passed one or more than one parameters in same method.
There are two types of parameters.

Formal parameters are the parameters as they are known in the function/method definition.

Actual parameters are also known as arguments and are passed by the caller on method invocation (calling the method).

Let's we see that, different between actual parameter and formal parameters.



Here 20 and 30 values are passed at method calling time so these are called actual parameters or arguments, so the 20 and 30 values have passed in variables x and y for the respectively. And these variables x and y are called formal parameters of sum method in above example.

Example 1:

```
class Demo
{
        publicvoid sum(int x, int y)
          {
             System.out.println("addition is="+ (x+y));
          }
        publicvoid sub(intx, int y)
```

```
        {
            System.out.println("subtraction is="+ (x-y));
        }
    publicvoid mul(int x, int y)
        {
            System.out.println("multiplication is="+ (x*y));
        }
    publicvoid div(int x, int y)
        {
            System.out.println("division is="+ (x/y));
        }
    public staticvoid main(String [] args)
        {
                Demo d1=new Demo();
                d1.sum(50,30);
                d1.sub(50, 30);
                d1.mul(50, 30);
                d1.div(50, 30);
        }
}
```

Output:

```
addition is=80
subtraction is=20
multiplication is=1500
division is=1
```

❖ **Method with a Return Type**

When sometime method returns some value that is the return-type of that method .For Example: sometimes method is takes arguments but does not return any things .so that method has return type is void. If method returns integer value then the return-type of method is an integer.
Following program shows the method with their return type.

Example 1:

```
class Demo
{
    publicint sum(int x, int y)
        {
    return (x+y);
        }
    publicint sub(int x,int y)
        {
    return (x-y);
        }
    publicint mul(int x, int y)
        {
    return(x*y);
        }
    publicint div(int x,int y)
```

```
        {
    return(x/y);
        }
    publicstaticvoid main(String [] args)
        {
                Demo d1=new Demo();
            int ans1,ans2,ans3,ans4;

                ans1=d1.sum(50,30);
                ans2=d1.sub(50, 30);
                ans3=d1.mul(50, 30);
                ans4=d1.div(50, 30);
                System.out.println("Addition is       "+ans1);
                System.out.println("Subtraction is    "+ans2);
                System.out.println("multiplication  is "+ans3);
                System.out.println("Division is       "+ans4);
        }
}
```

Output:

```
Addition is       80
Subtraction is    20
multiplication  is 1500
Division is       1
```

- **Call Static Method in same Class:**

  Static methods in Java can be called without creating an object of class. we write static keyword when defining main it's because program execution begins from main and no object has been created yet. Consider the example below to improve your understanding of staticmethods.

Example:

```
publicclass FirstJavaClass
 {
      public static void sum(int a, int b)          //static method
      {
            int c;
            c=a+b;
            System.out.println("Sum 0f two number is="+c);
      }
      public static void main(String[] args)
      {
      sum(10,20);     //Call static sum method from here , no need object
      }
}
```

Output:

```
Sum 0f two number is=30
```

- **Call static Method from other class:**

If you wish to call static method of another class then you have to write class name while calling static method as shown in example below.

Example:

```
publicclass FirstJavaClass
{
        public static void sum(int a, int b)   //static method
        {
                int c;
                c=a+b;
                System.out.println("Sum 0f two number is="+c);
        }
}
publicclass SecondjavaClass
{
        publicstaticvoid main(String[] args)
        {
          FirstJavaClass.sum(10,20); //call static method sum
        }
}
```

Output:

```
Sum 0f two number is=30
```

- **There are three categories of Method:**

  (a) No arguments (parameters) and No return value.
  (b) With argument but No return value.
  (c) With arguments and return value.

Let's see have explain all three categories with an example.

Example 1: sum of two numbers using first category

```
publicclass DemoMethod
{
publicvoid sum()
   {
        int a=10,b=20;
        int ans;
         ans=a+b;
         System.out.println("Sum of A and B =" + ans);
   }

        publicstaticvoid main(String[] args)
        {
                DemoMethod d1=new DemoMethod();
                d1.sum();
        }
}
```

Output:
Sum of A and B =30

Example 2: sum of two numbers using second category

```
publicclass DemoMethod
{
publicvoid sum(int a, int b)
   {
        int ans;
         ans=a+b;
         System.out.println("Sum of A and B =" + ans);
   }

        publicstaticvoid main(String[] args)
        {
                DemoMethod d1=newDemoMethod();
                d1.sum(10,20);
        }
}
```

Output:
Sum of A and B =30

Example 3: sum of two numbers using Third category

```
publicclass DemoMethod
{
publicint sum(int a, int b)
   {
        int ans;
         ans=a+b;
        return(ans);
   }

        publicstaticvoid main(String[] args)
        {
                DemoMethod d1=new DemoMethod();
                int ans1;
                ans1=d1.sum(10,20);
                System.out.println("Sum of A and B =" + ans1);
        }
}
```

Output:
Sum of A and B =30

3.3 Method overloading:

If a class have multiple methods by same name but different signatures, it is known as **Method Overloading**. Each method has a signature, which is types and Order of the parameters in the formal parameter list at method calling time. Method overloading can be used when the same logical operation requires multiple implementations.

Advantages of method overloading:

Perform only one operation, having same name of the methods increases the readability of the program.

There are two different way to overload the methods:

By changing number of arguments and

By changing the data type

Example:

Suppose we have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as sum(int ,int) for two parameters, and sum(int, int, int) for three parameters then it may be difficult for we to understand the behaviour of the method because its name differs. So, we have performed method overloading.

```
publicclass DemoMethodOverloding
{
publicvoid sum(int a, int b)
  {
          System.out.println("call sum method with integer arguments          ="+(a+b));
  }
publicvoid sum(int a, float f)
  {
   System.out.println("call sum method with int and float  arguments          ="+(a+f));
  }
publicvoid sum(int a, float f, double d)
  {
  System.out.println("call sum method with int, float and Double arguments ="+(a+f+d));
  }
        publicstaticvoid main(String[] args)
        {
         DemoMethodOverloding d1=new DemoMethodOverloding();
            d1.sum(10,20);
            d1.sum(20,20.23f);
            d1.sum(10, 12.25f,20.25d);
        }
}
```

Output:

```
call sum method with integer arguments          =30
call sum method with int and float  arguments   =40.23
call sum method with int, float and Double arguments =42.5
```

Example 2: method overloading example with methods called from other class:

```
publicclass DemoMethodOverloding
{
        publicvoid display(char ch)
       {
```

```
                System.out.println(ch);
        }
        publicvoid display(char ch, int n)
        {
                System.out.println(ch + " "+n);
        }
        publicvoid display(char ch, int n, String s)
        {
                System.out.println(ch + " "+n + " "+s);
        }
}
publicclass ABC
 {
        publicstaticvoid main(String args[])
          {
            DemoMethodOverloding  obj = new DemoMethodOverloding ();
            obj.display('a');
            obj.display('a',10);
            obj.display('a',10,"Hello");
          }
}
```

Output:

```
a
a 10
a 10 Hello
```

3.4 Constructor (Default constructor): [summer 2014, winter 2013]

Constructors in java are the methods which are used to initialize objects. Constructor declaration looks like method declaration. Constructor must have the same name as that of the class and have no return type. Constructors are called (invoked) when an object of class is created and constructor cannot be called explicitly. If you do not define a constructor, then the compiler creates a default constructor. Default constructors do not contain any parameters.

Characteristics of constructor:

- Constructors have must same name with that class name.
- Constructors have not any accessible modifier/visibility modifiers.
- They do not have any return type.
- Super and this keyword are not used to call constructor implicitly.
- Constructor used only once for an object at creating time.
- Constructor must declare inside the class.

Syntax:

```
class class_name
{
    class_name()                    //constructor definition
    {
```

```
            // body of default constructor
    }
    class_name(parameters)
    {
            // body of parameterised constructor
    }
     . . . . . .
     . . . . . .
    public static void main(String [] args)
    {
        class_name object_name = class_name ();          // call constructor. No need to object.
    }
}
```

Example 1:

```
package alpha;
publicclass Constructor
{
        Constructor()              //default constructor
        {
            System.out.println("called constructor when object created");
        }
        publicvoid userdefinemethod()   //user define method
        {
          System.out.println("called user define method through an object");
        }
        publicstaticvoid main(String args[])
        {
           Constructor obj =newConstructor();
           obj.userdefinemethod();
        }
}
```

Output:

called constructor when object created
called user define method through an object

3.5 Parameterized Constructor:

Constructors that does will take parameter then these Constructors are called as **"**Parameterized Constructor**".**

1. Constructor Can Take Value(s), Value(s) is/are Called as – "Argument".
2. Arguments can be of any type i.e Integer, Character, Array or any Object.
3. Constructor can take any number of Arguments.

Example 1: demo of parameterized constructor:

```
package alpha;
publicclass Rectangle
{
```

```
        publicintlength;
        publicintbreadth;

         Rectangle ()                    //constructor
         {
                length =0;
                breadth =0;
         }
         Rectangle(int len,int bre)  //parameterized constructor
         {
        length  = len;
        breadth = bre;
         }
        publicvoid display()
         {
                System.out.println("Length="+ length);
                System.out.println("breadth="+ breadth);
         }
}
publicclass RectangleDemo
{
        publicstaticvoidmain(String args[])
        {
         Rectangle r1 = new Rectangle();
         Rectangle r2 = new Rectangle(100,200);
         System.out.println("r1 object length and breadth");
         r1.display();
         System.out.println("r2 object length and breadth");
         r2.display();
        }
}
```

Output:

```
r1 object length and breadth
Length=0
breadth=0
r2 object length and breadth
Length=100
breadth=200
```

3.6 copy constructor:

There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in java. They are:

- By constructor
- By assigning the values of one object into another
- By clone() method of Object class

In this example, we are going to copy the values of one object into another using java constructor.

Example 1:

```
package alpha;

publicclass Employee
{
intid;
String name;
public Employee(int i, String string)
{
id = i;
name = string;
   }
   Employee(Employee s)
   {
id = s.id;
name =s.name;
   }
void display()
   {
         System.out.println("id="+id);
         System.out.println("name="+name);
   }
publicstaticvoid main(String args[])
  {
            Employee e1 = new Employee(007,"jhon");
            Employee e2 = new Employee(e1);   //copy e1 object data into object e2
            e1.display();
            e2.display();
  }
}
```

Output:

```
id=7
name=jhon
id=7
name=jhon
```

3.7 Multiple constructors or Constructor overloading:

1. Constructor overloading is similar to method overloading in Java.
2. You can call overloaded constructor by using this() keyword in Java.
3. Overloaded constructor must be called from another constructor only.
4. make sure you add no argument default constructor because once compiler will not add if you have added any constructor in Java.
5. if an overloaded constructor called , it must be first statement of constructor in java.
6. Its best practice to have one primary constructor and let overloaded constructor calls that. this way.

Your initialization code will be centralized and easier to test and maintain.

Example 1:

```
package alpha;

publicclass MyOverloading
{
public  MyOverloading()
   {
      System.out.println("called  default constructor");
   }
public MyOverloading(int i)
   {
      System.out.println("constructor parameterised with int value");
      System.out.println("i="+i);
   }
public MyOverloading(String str)
   {
      System.out.println("constructor parameterised with string object");
      System.out.println("String Object is="+str);
   }
public MyOverloading(int i,int j)
   {
      System.out.println("constructor parameterised with two integer arguments");
      System.out.println("i="+i);
      System.out.println("j="+j);
   }
publicstaticvoid main(String a[])
   {
      MyOverloading m1 =newMyOverloading();
      MyOverloading m2 = newMyOverloading(10);
      MyOverloading m3 = new MyOverloading(10,20);
      MyOverloading m4 = newMyOverloading("methodoverloding demo");
   }
}
```

Output:

```
called  default constructor
constructor parameterised with int value
i=10
constructor parameterised with two integer arguments
i=10
j=20
constructor parameterised with string object
String Object is=methodoverloding demo
```

3.8 Java Constructor Chaining Sample Code:

Example:

```
package alpha;

publicclass Chainingconstrucor
{
public Chainingconstrucor()
```

```
    {
        System.out.println ("In constructor...");
    }
public Chainingconstrucor(int i)
    {
this();
        System.out.println ("In single parameter constructor...");
    }
public Chainingconstrucor(int i, int j)
    {
this(j);
        System.out.println ("In double parameter constructor...");
    }
publicstaticvoid main(String a[])
    {
          Chainingconstrucor ch = new Chainingconstrucor(10,20);
    }
}
```

Output:

```
In constructor...
In single parameter constructor...
In double parameter constructor...
```

3.9 Different between method and constructor: [winter 2012, Nov dec 2011, dec 2010]

Constructor	Method
Constructor no need to called, they are called automatically when we created an object of that class.	In this case, we need call method through an object of that class.
Constructor is used to initialize the state of an object.	Method is used to expose behaviour of an object.
Constructor name must be same as the class name.	Method name may not be same as class name.
Constructor must not have return type.	Method must have return type.
Constructor is invoked implicitly.	Method is invoked explicitly.
The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in any case.

3.10 recursion in java:

There are two types of recursion in java. **Single recursion and multiplerecursions.**
Single recursion means that the function calls itself **only once**.
Multiple recursionsmean that the function calls itself more than once.

Example: calculating the factorial:

```
import java.io.DataInputStream;
import java.io.IOException;

public class FactorialUsingRecursion
```

```
{
publicstaticvoid main(String args[])throws NumberFormatException,IOException
    {
int  a ;
        DataInputStream s1 = new  DataInputStream(System.in);
        System.out.println("Enter value of A");
        a=Integer.parseInt(s1.readLine());

int result= fact(a);            //call fact function from here
        System.out.println("Factorial of the number is: " + result);
}

Static int fact(int b)
 {
if(b <= 1)
        Return 1;
        else
        return b * fact(b-1);
}
}
```

Output:

```
Enter value of A
5
Factorial of the number is: 120
```

3.11 pass by value and pass by reference: [dec 2010]

- **Pass by value** in java define like, passing a value as an argument in method at calling time. So the called method is only operating on original argument and some changes have made in called method are not reflected in the main calling method or program.

Example1:

```
publicclass DemoPassbyValue
{
publicvoid swap(int a ,intb)
  {
        int temp;
        temp=a;
        a=b;
        b=temp;
        System.out.println("After swap value of a=" +a+ "   b=" +b);
  }
publicstaticvoid main(String args[])
  {
        DemoPassbyValue obj =new DemoPassbyValue();
        int a=10,b=20;
        System.out.println("before swap value of a=" +a+ "   b=" +b);
        obj.swap(a,b);
```

```
  }
}
```

Output:

```
before swap value of a=10   b=20
After swap value of a=20   b=10
```

- **Pass by Reference** means the passing the address of the argument in the method call. So if the called method changes the state of the argument object then it is reflected in the calling program.

Objects are always passed by reference. When we pass a value by reference, the reference or the memory address of the variables is passed. Thus any changes made to the argument Causes a change in the values which we pass.

In other words, if an Object reference is passed as an argument, then a copy of the object reference is sent to the called method. This means that both the object reference in the calling program and copy of object reference sent to the called method are pointing to the same object.

Example 2:

```
publicclass DemoPassbyref
{
privatestaticclass Vehicle
  {
private String type;
public Vehicle(String type)
        {
this.type = type;
        }
  }
publicstaticvoid printVehicletype(Vehicle v)
{
        v.type = "Bike";
        System.out.println("Type of vehicle in printVehicletype(): "+v.type);
  }

publicstaticvoid main(String args[])
  {
        Vehicle v =new Vehicle("CAR");
        System.out.println("Type of Vehicle in main() before: "+ v.type);
        printVehicletype(v);
        System.out.println("Type of Vehicle in  main() after: "+ v.type);
}
}
```

Output:

```
Type of Vehicle in main() before: CAR
Type of vehicle in printVehicletype(): Bike
Type of Vehicle in  main() after: Bike
```

3.12 “this” reference: [June 2011]

this is a keyword in Java which can be used inside method or constructor of class.it works as a reference to current object whose method or constructor is being invoked. **this** keyword can be used to refer any member of current object from within an instance method or a constructor. There are lot of use of this key word in java. 'this' is a **reference variable** that refers to the current object.

Let us see below use of this key word.

- 'this' keyword can be used to refer current class instance variable.
- 'this()' can be used to invoke current class constructor.
- 'this' keyword can be used to invoke current class method (implicitly)
- 'this' can be passed as an argument in the method call.
- 'this' can be passed as argument in the constructor call.
- 'this' keyword can also be used to return the current class instance.

Use of 'this' references: If there is ambiguity between the instance variable and parameter, this keyword resolves the problem of ambiguity.

Example 1:

```
package alpha;
publicclass Employee
{
        intid;
          String name;

          Employee(int id, String name)
          {
                this.id = id;
                this.name = name;
          }
        void display()
          {
                 System.out.println("id="+id);
                 System.out.println("name="+name);
          }
        publicstaticvoid main(String args[])
          {
                   Employee e1 = new Employee(101,"jhon");
                   Employee e2 = new Employee(102,"carter");
                   e1.display();
                   e2.display();
          }
}
```

Output:

```
id=101
name=jhon
id=102
```

name=carter

3.13 static keyword in java: [June 2011]

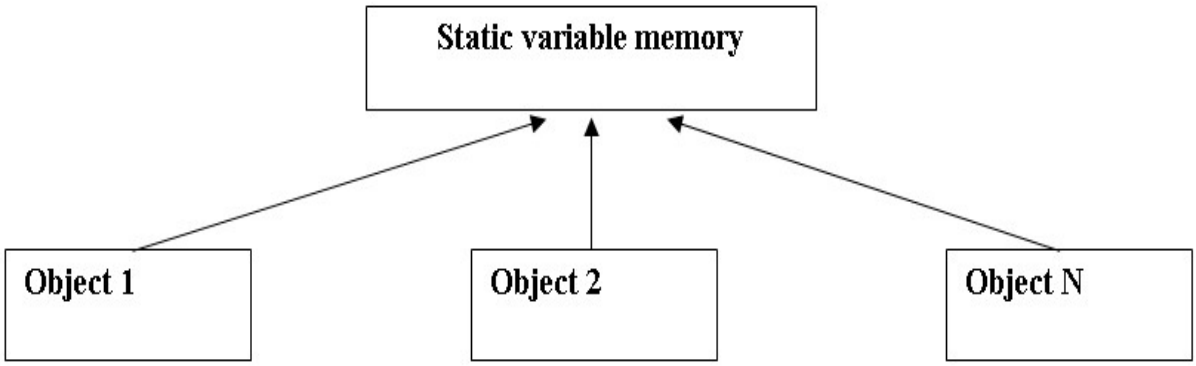
(a)static variable:

The **static keyword** in java is used for memory management. We can apply static keyword on variables, methods, blocks and nested classes. **Static keyword** we cannot apply on top level class means last outer class in java program. Static variables are associated with class objects. Static variable value in java kept same value for each and every object Means static variables value shared to all other objects in class. And we cannot use non static variable in static method.

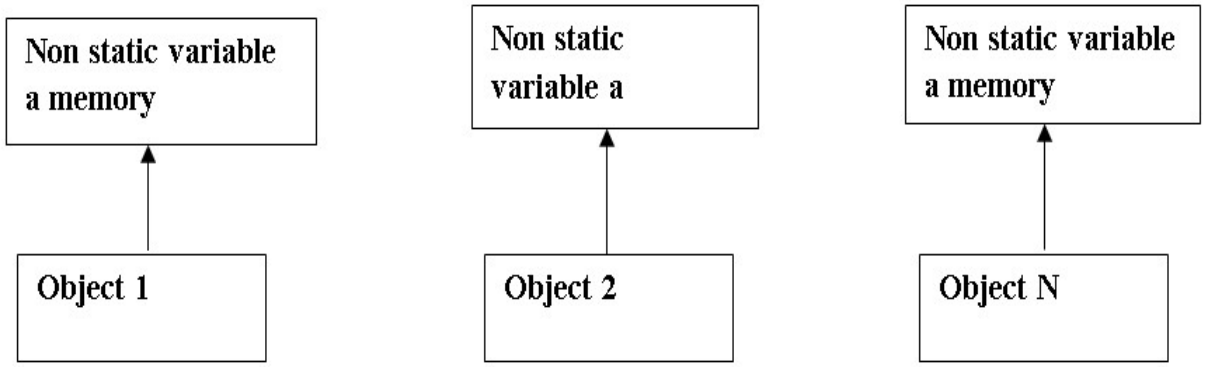
Characteristics of static variables:

- Static variables are called by only static member function/methods.
- Static member variable cannot referred as a 'this' or super keyword.
- Static members cannot call by object of any classes.

See figure for illustration of static variable which is common memory for all Objects.



See figure for illustration of non-static variable which is created memory for its own objects.



Example 1:

```
package alpha;
public class ClassA
{
public static int counter=0;
public void getcountervalue()
 {
        System.out.println("Value of counter="+counter);
 }
public void countervalueincrease()
 {
        counter++;
 }
public static void main(String [] args)
 {
        ClassA a1 =new ClassA();
        ClassA a2 =new ClassA();
        ClassA a3 =new ClassA();
        System.out.println("print counter before increase value by a1 a2 and a3 objects");
        a1.getcountervalue();
        a2.getcountervalue();
        a3.getcountervalue();
        a1.countervalueincrease();
        a2.countervalueincrease();
        a3.countervalueincrease();
        System.out.println("print counter After increase value by a1 a2 and a3 objects");
        a1.getcountervalue();
        a2.getcountervalue();
        a3.getcountervalue();
        }
}
```

Output:

```
print counter before increase value by a1 a2 and a3 objects
Value of counter=0
Value of counter=0
Value of counter=0
print counter After increase value by a1 a2 and a3 objects
Value of counter=3
Value of counter=3
Value of counter=3
```

(b) Static method:

Static methods in Java can be called without creating an object of class. Static method doesn't modify state of object. Static method mostly operates on arguments, almost all static method accepts arguments, perform some calculation and return value.

Example1: Java static method and instance method called from same class

```
package alpha;
public class DemoStatic
{
        static void display()
         {
                System.out.println ("Static method Called.");
         }
        void show()
         {
                System.out.println ("Non static method Called.");
         }
        public static void main(String[] args)
         {
                display();          //static method calling without object
               DemoStatic a1 = new DemoStatic();
               a1.show();          //non static method calling using object
         }
}
```

Output:

```
Static method Called.
Non static method Called.
```

Example 2: Java static method and instance method called from different class.

```
package alpha;
public class DemoStatic
{
static void display()
         {
              System.out.println ("Static method Called.");
         }
        void  show()
         {
             System.out.println ("Non static method Called.");
         }
}
package alpha;
public class ClassB
{
        public static void main(String[] args)
         {
             DemoStatic.display();         //static method call with class name
             DemoStatic a1 = new DemoStatic();
             a1.show();                    //non static method calling using object
         }
```

```
}
```

Output:

```
Static method Called.
Non static method Called.
```

Example 3: static method access only static variables.

```
package alpha;
publicclassDemoStatic
{
intno;
    String name;
staticString company = "Axisray";

staticvoid edit()                    //static method
    {
        company = "TCS";          // static method access only static variable
    }
    DemoStatic(int r, String n)
    {
no = r;
name = n;
    }
voiddisplay ()
    {
            System.out.println(no+" "+name+" "+company);
    }
publicstaticvoid main(String args[])
   {
            DemoStatic.edit();
            DemoStatic s1 = new  DemoStatic (1,"ABC");
            DemoStatic s2 = new  DemoStatic (2,"PQR");
            DemoStatic s3 = new  DemoStatic (3,"CDE");
            s1.display();
            s2.display();
            s3.display();
   }
}
```

Output:

```
1 ABC TCS
2 PQR TCS
3 CDE TCS
```

3.14 garbage collectors / finalized () method OR no destructor in java: [winter 2014, summer 2014, winter 2013, summer 2013,Nov dec 2011, June 2011]

In java, destructors are not supports. So finalize () method comes into pictures. Finalized method in java is a very special method like main method in java. Finalize () method called before garbage collectors of objects. Its last chance for any object to clean up all activities such as system resource held, closing connection if it's open it etc.

Sometimes an object will need to perform some action when it is destroyed. For example, if an object is holding some resource such as a file operation handle, then you make sure these resources are freed before an object is destroyed. To handle these type situations, java provides a mechanism called finalization.

And these type mechanism provided by finalize **()** method. Inside the **finalize ()** method you will specify those actions that must be performed before an object is destroyed. The garbage collector runs periodically, checking for objects.

Syntax:

```
protected void finalize ()
 {
        // finalization code here
 }
```

Here, the keyword **protected** is a access modifier or visibility mode that prevents access to **finalize ()** by code defined outside its class.

Example

```
publicclass FinalizedMethodDemo
{
        public void method()
         {
                System.out.println ("Calling method through object d1...");
         }
        public static void main(String[] args)
        {
                 FinalizedMethodDemo d1=new FinalizedMethodDemo();
                 d1.method();
        try
                {
                        System.out.println ("Finalizing...");
                       d1.finalize();
                       System.out.println ("Finalized.");
                }
        catch (Throwable ex)
                {
                      ex.printStackTrace();
                }
        }
}
```

Output:

```
Calling method through object d1...
Finalizing...
Finalized.
```

3.15 access control/ access modifier/visibility mode (private, protected, public) for method(s): [winter 2014]

Java provides a private, protected and public access modifiers to set access levels for classes, variables, methods and constructors.

- **Private method:**
  Methods that are declared with private can only be accessed within the declared class itself. Private access (visibility) modifier is the most restrictive access level. Class and interfaces cannot be private. Private methods are inaccessible.

- **Protected Method:**
  Methods which are declared as a protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.

- **Public method:**
  A method declared as a public can be accessed from any other class. If the public class that is trying to access is in a different package, then the public class still need to be imported.

Example 1: public, protected and private methods are accessed from same class as well as same package;

```
package alpha;
publicclass ClassA
{
        privateinta;
        protectedintb;
        publicintc;
        privatevoid privateMethodclassA()
        {
          System.out.println("private method in classA is called");
          System.out.println("private variable a="+a);
        }
        protectedvoid protectedMethodclassA()
        {
          System.out.println("protected method in classA is called");
          System.out.println("protected variable b="+b);
        }
publicvoid publicMethodclassA()
        {
          System.out.println("public method in classA is called");
          System.out.println("public variable c="+c);
        }
publicstaticvoid main(String[] args)
        {
         ClassA obj1=new ClassA();
         obj1.a=10;                          //set 10 value in public variable a;
         obj1.b=20;            //set 20 value in protected variable b;
```

```
            obj1.c=30;              //set 30 value in public variable c;
            obj1.privateMethodclassA();       //valid
            obj1.protectedMethodclassA();     //valid
            obj1.publicMethodclassA();        //valid
        }
}
```

Output:

```
private method in classA is called
private variable a=10
protected method in classA is called
protected variable b=20
public method in classA is called
public variable c=30
```

Example 2: public, protected and private methods are accessed from different class but same package;

```
package alpha;
publicclass ClassA
{
        privateinta;
        protectedintb;
        publicintc;
        privatevoid privateMethodclassA()
        {
          System.out.println("private method in classA is called");
          System.out.println("private variable a="+a);
        }
        protectedvoid protectedMethodclassA()
        {
          System.out.println("protected method in classA is called");
          System.out.println("protected variable b="+b);
        }
publicvoid publicMethodclassA()
        {
          System.out.println("public method in classA is called");
          System.out.println("public variable c="+c);
        }
}
package alpha;
publicclass ClassB
{
publicstaticvoid main(String[] args)
  {
        ClassA obj1=new ClassA();
        //obj1.a=10;                    //invalid because var a is private;
        obj1.b=20;                  //set 20 value in protected variable b;
        obj1.c=30;                  //set 30 value in public variable c;
        //obj1.privateMethodclassA();     //invalid because private method
        obj1.protectedMethodclassA();    //valid
        obj1.publicMethodclassA();       //valid
```

```
  }
}
```

Output:

```
protected method in classA is called
protected variable b=20
public method in classA is called
public variable c=30
```

Example 3: public, protected and private methods are accessed from different class as well as different package;

```
package alpha;
publicclass ClassA
{
        privateinta;
        protectedintb;
        publicintc;
        privatevoid privateMethodclassA()
        {
          System.out.println("private method in classA is called");
          System.out.println("private variable a="+a);
        }
        protectedvoid protectedMethodclassA()
        {
          System.out.println("protected method in classA is called");
          System.out.println("protected variable b="+b);
        }
publicvoid publicMethodclassA()
         {
          System.out.println("public method in classA is called");
          System.out.println("public variable c="+c);
        }
}
package Beta;

import alpha.ClassA;

publicclass ClassC
{
        publicstaticvoid main(String[] args)
        {
          ClassA obj1=new ClassA();
        //obj1.a=10;                          //invalid because var a is private
        //obj1.b=20;                           //invalid because var b is a protected
          obj1.c=30;              //set 30 value in public variable c;
        //obj1.privateMethodclassA();        //invalid, because private method
        //obj1.protectedMethodclassA();     //invalid, because protected method
          obj1.publicMethodclassA();          //valid
        }
}
```

Output:

```
public method in classA is called
public variable c=30
```

❖ **Accessible Table for method:**

Accessible mode for method	Same class and same package	Different class But same package	Different class as well as different package
Private method	✓	✗	✗
Protected method	✓	✓	✗
Public method	✓	✓	✓

3.16 Nested class (inner class) static inner-class and non-static inner class: [Nov dec 2011, June 2011]

In Java nested classes means one or more classes are defined inside another class.Java developers refer to nested classes as inner classes, but inner classes have several different types of nested classes such as static inner class, non-static inner class, local class and Anonymous inner class.

Static inner class:

Declaration of static inner class:

```
public class Outer
{
public static class inner
        {
                ……
        }
}
```

In order to create an instanceobject of the Nested class you must reference it by prefixing it with the Outer class name, like this:

```
OuterClassName.InnerClassName instance_obj_name = OuterClassName.InnerClassName();
```

Example:

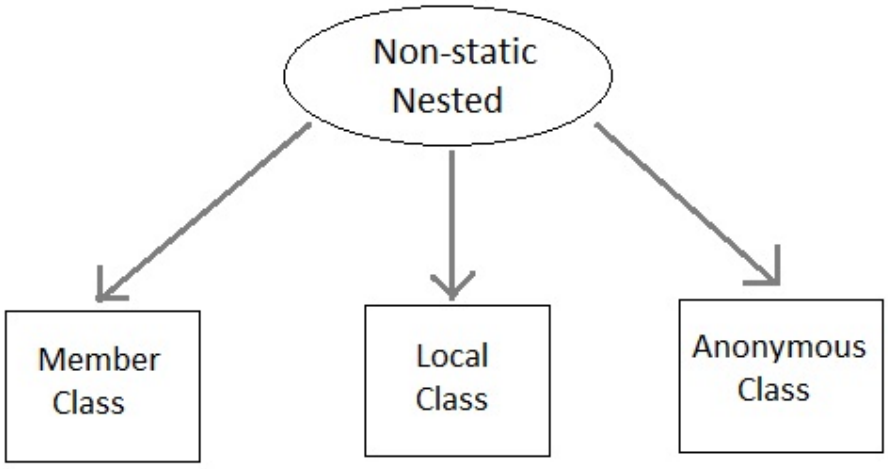
```
public class OuterDemo
{
        intb=20;
        static class inner
        {
        int a=10;
        public void innerMethod()
            {
```

```
                System.out.println("inner class method is called ");
            }
        }
        publicstaticvoid main(String []args)
        {
           OuterDemo.inner obj=new OuterDemo.inner();
           System.out.println("Value of a =" + obj.a);
        obj.innerMethod();
           //System.out.println(obj.b); cannot access; var b is declared in outer class
        }
}
```

Output:
Value of a =10
inner class method is called

Non static inner class:

Non-static nested classes in Java are also called *inner classes*. Inner classes are associated with an instance of the enclosing class. Thus, you must first create an instance of the enclosing class to create an instance of an inner class.



Let see non Static inner class definition:

```
public class Outer
{
        public class inner
        {
        ……….
        ……….
        }
}
```

Now howto create an instance Object of the Inner class:

```
OuterClassName outerobj =new OuterClassName();
OuterClassName.InnerClassName InstanceObj= outerobj.new InnerClassName();
```

Notice how you put new after the reference to the outer class in order to create an instance of the inner class

Example: Example of Inner class instantiated outside Outer class.

```
publicclass OuterDemo
{
        publicvoid OuterclassMethod()
        {
                System.out.println("Outer class method is called ");                    }

class inner
        {
                publicvoid InnerclassMethod()
                {
                    System.out.println("inner class method is called ");
                }
         }
        publicstaticvoid main(String []args)
        {
                 OuterDemo obj =new OuterDemo();
                 obj.OuterclassMethod();          // call outer class method
                 OuterDemo.inner i =obj.new inner();
                 i.InnerclassMethod();            // call inner class method
        }
}
```

Output:

```
Outer class method is called
inner class method is called
```

Example:

```
publicclass OuterClass
{
        publicvoid display()
        {
                System.out.println("Outer class method display() called");
                inner i=new inner();
                i.show();
        }
        class inner
        {
                publicvoid show()
                {
                System.out.println("inner class method show() called");
                }
        }
        publicstaticvoid main(String[] args)
```

```
  {
            OuterClass obj=new OuterClass();
            obj.display();
      }
}
```

Output:

Outer class method display() called
inner class method show() called

3.17 Anonymous inner class:

In java Anonymous classes are nested classes without a class name. Anonymous classes are declared as either subclasses of an existing class or as implementations with use of an interface. (Note interface and inheritance we will discuss in later chapter).

Define Anonymous classes when they are instantiated.

Syntax:

```
Public class SuperClass
{
      Public void Methodname ()
      {
            System.out.println ("Super class do it");
      }
      SuperClass InstanceObj=new SuperClass ()
      {
      Public void Methodname ()
            {
              System.out.println ("Anonymous class do it");
            }
      };
     InstanceObj.Methodname ();
. . . . . .
. . . . . . . . . .
}
```

In above syntax the anonymous class subclasses (extends) SuperClass and overrides the **Methodname ()** method.

Then, Java anonymous class can also implement an interface instead of extending a class. See in below example:

```
public interface InterfaceName
{
      public void  Methodname ();
      {
```

```
        }
        InterfaceName InstanceObj =new InterfaceName ();
        {
                public void MethodName()
                {
                        System.out.println ("Anonymous class call it");
                }
        };
InstanceObj. MethodName ();
}
```

Now you can see that an anonymous class is implementing an interface that is similar to an anonymous class extending another class. An anonymous class can access members of the enclosing class. Anonymous class can also access local variables.we can declared member variables and methods inside anonymous class but we can not declared constructor in side anonymous class.

Example:

```
publicinterface Car
{
        void Method();
}
publicclass Test
 {
        publicstaticvoid main(String args[])
         {
            Car c = new Car()              //Annonymous class created from here
            {
        public void Method()
                {
                        System.out.println("Annonymous CAR");
                 }
            };
            c.Method();  // instance of anonymous class called  method from here
        }
}
```

Output:
Annonymous CAR

3.18 local classes:

In java, The Local classes are like inner classes (non-static nested classes) that are defined inside a method or scope block means in between { } (open and close curly brackets) inside a method.

Declaration of Local Class:

```
                class outerclass
                {
                public void Methodname()
```

```
        {
class LocalClass
        {
        }
        LocalClass Object_name= new LocalClass ();
}
```

Local classes can only be accessed from inside the method or scope block in which they are defined. Local classes can access members (variables and methods) of its enclosing class just like regular inner classes. Local classes can also access final local variables inside the same method or scope block. Local classes can also be declared inside static methods. In that case the local class only has access to the static parts of the enclosing class.

Example:

```
publicclass LocalClassDemo
{
        publicvoid method()
        {
        System.out.println("called method");
        int a=10;
        class Local
        {
                intb=20;
        publicvoid localMethod()
        {
                  System.out.println("Local method");
                  System.out.println("b="+b);
                //System.out.println("a="+a); //invalid because declare outside to local class
        }
}
        Local L1 =new Local();
        L1.localMethod();
}

    publicstaticvoid main(String[] args)
{
        LocalClassDemo d1=new LocalClassDemo();
          d1.method();
}
}
```

Output:

called method
Local method
b=20

3.19 Abstract class:

Abstraction discusses to the ability to make a class abstract in OOP. An abstract class is one that cannot be instantiated. Other functionality of the abstract class is that still exists, and its fields, methods, and constructors are all accessed in the same manner.Abstract class is declared with the **abstract** keyword.
More detail about abstract class we will discuss in next chapter.

3.20 Abstract Method:

The abstract keyword is also used to declare an abstract method. An abstract method consists of a method signature, but no method body.
More detail about abstract method we will discuss in next chapter.

3.21 GTU Question paper Program:

Program 1:

Declare a class called coordinate to represent 3 dimensional Cartesian coordinates(x, y, and z) define following method.
Constructor
Display to print values of members
Add_coordinates, to add three such coordinates object to produce a resultant coordinates object. Generate and hendle exception if x,y and z coordinates of the result are zero
Main , to show use of above method [Summer 2013]

```
class  coordinate extends Exception
{
        double x,y,z;
        coordinate()
        {
                x=0;
                y=0;
                z=0;
        }
        coordinate(double a,double b,double c)
        {
                x=a;
                y=b;
                z=c;
        }
        void  display()
        {
                System.out.println("X="+x);
                System.out.println("Y="+y);
                System.out.println("Z="+z);
        }
        coordinate  add_coordinates(coordinate obj1,coordinate obj2,coordinate obj3) throws
coordinate
        {
                coordinate obj4=new coordinate();
                obj4.x=obj1.x+obj2.x+obj3.x;
```

```
                obj4.y=obj1.y+obj2.y+obj3.y;
                obj4.z=obj1.z+obj2.z+obj3.z;
                if(obj4.x==0 || obj4.y==0 || obj4.z==0)
                {
                        throw new coordinate();
                }
          return obj4;
        }
}
class test
{
        public static void  main(String[] args)
      {
                coordinate c1= new  coordinate();
                coordinate c2= new  coordinate(1,2,3);
                coordinate c3= new  coordinate(4,5,6);
                coordinate c4= new  coordinate(7,8,9);
                try
                {
                        c1=c1.add_coordinates(c2,c3,c4);
                        c1.display();
                }
                catch(coordinate c)
                {
                        System.out.println("exception for zero value");
                }
        }
}
```

Output:

X=12.0

Y=15.0

Z=18.0

Program 2:

Design a class named Fan to represent a fan. The class contains:
- Three constants named SLOW, MEDIUM and FAST with values 1,2 and 3 to denote the fan speed.
- An int data field named speed that specifies the speed of the fan (default SLOW).
- A boolean data field named f_on that specifies whether the fan is on(default false).
- A double data field named radius that specifies the radius of the fan (default 4).
- A data field named color that specifies the color of the fan (default blue).
- A no-arg constructor that creates a default fan.
- A parameterized constructor initializes the fan objects to given values.
- A method named display() will display description for the fan. If the fan is on,the display() method displays speed, color and radius. If the fan is not on, the method returns fan color and radius along with the message "fan is off".Write a test program that creates two Fan objects. One with default values and the other with medium speed, radius 6, color brown, and turned on status true. Display the descriptions for two created Fan objects. [Nov Dec 2011]

```
class fan
{
	final int slow=1;
	final int medium=2;
	final int fast=3;
	int speed;
	boolean f_on;
	double radius;
	String color;
	fan()
	{
		speed=slow;
		f_on=false;
		radius=4;
		color="blue";
	}
	fan(int s,booleanf,doubler,String s1)
	{
		speed=s;
		f_on=f;
		radius=r;
		color=s1;
	}
	void show()
	{
		if(f_on)
		{
			System.out.println("FAN IS ON");
			System.out.println(speed);
			System.out.println(radius);
			System.out.println(color);
		}
		else
		{
			System.out.println("FAN IS OFF");
			System.out.println(radius);
			System.out.println(color);
		}
	}	}
class test
{
	public static void main(String s[])
	{
		fan f1=new fan();
		fan f2=new fan(2,true,6,"brown");
		f1.show();
		f2.show();
	}
}
```

Output:

```
FAN IS OFF
4.0
blue
FAN IS ON
2
6.0
brown
```

Program 3:

Define the Rectangle class that contains:
Two double fields x and y that specify the center of the rectangle, the data field width and height ,A no-arg constructor that creates the default rectangle with (0,0) for (x,y) and 1 for both width and height.A parameterized constructor creates a rectangle with the specified x,y,height and width.A method getArea() that returns the area of the rectangle.A method getPerimeter() that returns the perimeter of the rectangle.A method contains(double x, double y) that returns true if the specified point (x,y) is inside this rectangle.Write a test program that creates two rectangle objects. One with default values and other with user specified values. Test all the methods of the class for both the objects. [Nov Dec 2011]

```
class rectangle
{
        double cx,cy,height,width,xleft,xright,yup,ydown;
        rectangle()
        {
                cx=0;
                cy=0;
                height=1;
                width=1;
        }
        rectangle(double x,doubley,doubleh,double w)
        {
                cx=x;
                cy=y;
                height=h;
                width=w;
        }
        double getarea()
        {
                return (height*width);
        }
        double getperimeter()
        {
                return (2*(height+width));
        }
        boolean contains(double x,double y)
        {
                xleft=(cx-(width/2));
```

```
                xright=(cx+(width/2));
                yup=(cy+(height/2));
                ydown=(cy-(height/2));
                if((x>xleft&& x<xright) && (y<yup && y>ydown))
                {
                        return true;
                }
                else
                {
                        return false;
                }

        }
}
class test
{
        public static void main(String s[])
        {
                rectangle r1=new rectangle();
                rectangle r2=new rectangle(10,30,10,20);
                System.out.println(r1.getarea());
                System.out.println(r2.getarea());
                System.out.println(r1.getperimeter());
                System.out.println(r2.getperimeter());
                if(r1.contains(3,5))
                {
                        System.out.println("point is inside the rectangle");
                }
                else
                {
                        System.out.println("point is outside the rectangle");
                }
                if(r2.contains(250,310))
                {
                        System.out.println("point is inside the rectangle");
                }
                else
                {
                        System.out.println("point is outside the rectangle");
                }
        }
}
```

Output:

```
1.0
200.0
4.0
60.0
point is inside the rectangle
point is inside the rectangle
```

3.22 Asked Question in GTU Papers:

Q 1.Explain and illustrate by example use of method finalize. [June 2013]
Q 2. What is used of java keyword “static”.[Dec 2012]
Q 3. Different between constructor and method of class.[Dec 2012]
Q 4.Explain Method overloading with example.[June 2012]
Q 5.Explain the following: “this”. [Dec 2011]
Q 6.Explain method overloading with help of example.[June 2011]
Q 7.Explain this reference, static keyword and garbage collection. [June 2011]
Q 8. Different between constructor and method of class. Define method overloading and Its purpose. Write a program to demonstration of the constructor. [Dec 2010]
Q 9. Define and write a program to differentiate between pass by value and pass by reference.[Dec 2010]
Q 10. Explain inner class by giving example.[June 2011]
Q 11.how the concept of inner class is used? [Dec 2012]