Unit-3
Object oriented programming concepts

3.1 ABSTRACTION:

- **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.
- Another way, it shows only important things to the user and hides the internal details for example *sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.*
- Abstraction lets you focus on what the object does instead of how it does it.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class
2. Interface

3.2 ENCAPSULATION:

- Encapsulation: Wrapping up of data and function into single unit is called encapsulation.
- Encapsulation is the technique of making the fields in a class private and providing access to the fields via public methods. If a field is declared private, it cannot be accessed by anyone outside the class, thereby hiding the fields within the class. For this reason, encapsulation is also referred to as data hiding.
- The main benefit of encapsulation is the ability to modify our implemented code without breaking the code of others who use our code. With this feature Encapsulation gives maintainability, flexibility and extensibility to our code

3.3 POLYMORPHISM:

- Polymorphism means—ability of function to take more than one form.
- Polymorphism is a characteristic of being able to assign a different behaviour or value in a subclass, to something that was declared in a parent class.
- .

Two types of polymorphism:

- **Compile time polymorphism: Method(function) Overloading:**

  In same class, if name of the method remains common but the number and type of parameters are different, then it is called method overloading in Java.

  Run time polymorphism: Method(function) Overloading:

  If Super class and sub class contain same method having same name, same parameters and same return type, then it is called method overriding.

3.4 INTRODUCTION TO CLASS

- Java is true object oriented language.
- Anything in java program must be encapsulated in class.
- Class **defines** state and behavior of objects.
- Class create objects and objects use method to communicate between them.
- **Class** provide **convenient method for packing together a group of logically related data items and functions to work on them.**
- **Class** is **user defined data type with a template that serves to define its properties.**
- Once class has been defined we can create variables of that type ,these variables are termed as instance of classes they are **actual objects**.
- Everything inside square brackets is optional

Syntax:

```
class classname[ extends superclass name ]
        {
                [ variable declaration;  ]
                [ methods declaration; ]
        }
```

- Any class at least have,

```
class classname
{
}
```

- If class does not contain any properties and therefore can't do anything.but can compiled and even create object using it.
- *Classname* and *super classname* are any valid java identifiers.
- Keyword extends indicates ,the properties of *super classname* are extended to *classname* class.

Adding variables to class

- Data encapsulated in a class by placing field inside body of class, these variable are instance variable.

```
class Rectangle
{
        int length;
        int width;
}
```

Instance variable
Or
Class varaible

- Instance variable are created whenever object of class is instantiated.

- Here, variable are only declared and no storage space has been created in memory.

Example:

```
class Rectangle
{
        int length;
        int width;

        void getData ( int x , int y)
        {
                length=x;
                width=y;
        }

        int rectArea ( )
        {
                int area= length * width ;
                return (area);
        }
}
```

Example-2

```
class A
{
        int x;
        void method1( )
        {
                int y;
                x=10;                   // valid
                y=x;                    // valid
        }

        void method2( )
        {
                int z;
                x=5;                    // valid
                y=1;                    // invalid
        }
}
```

3.4 Creating objects

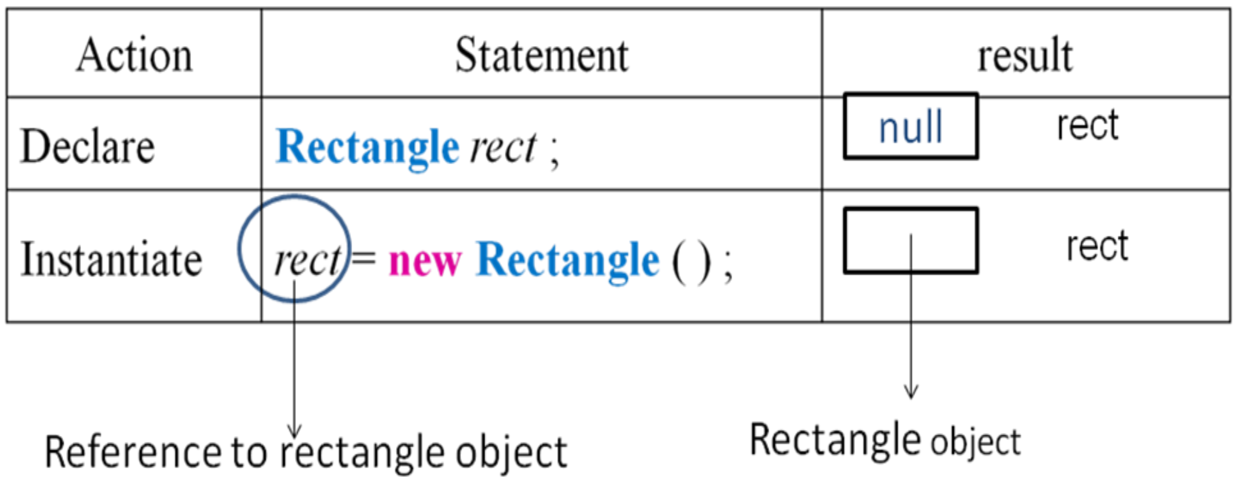
How to create object?

- Object is a essential block of memory that contains space to store all instance variable.
- Creating object is referred as instantiating an object.
- **new** operator creates objects in java.

it creates object of specified class and returns reference to that object.

Declaration: **classname** *objectname*;
Instantiate: *objectname* = **new classname( );**

assigns object reference to the variable



Action	Statement	result
Declare	**Rectangle** *rect* ;	null rect
Instantiate	*rect*= **new Rectangle** ( ) ;	rect

Ex:

Rectangle rect;
rect= new **Rectangle( );**
or
Rectangle *rect* = **new** **Rectangle( );**

Default constructor

- **Class can have any no. of objects of Rectangle.**
- Each object has its own instance variables copy so change to variable of one object have no effect on variable of another.

Accessing class member

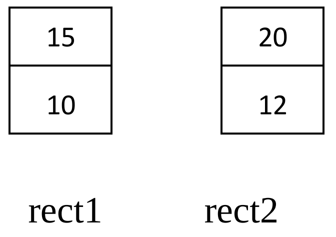
We can access varible and method using object by using dot(.) operator.

Syntax:

Objectname . variablename;
Objectname . method name (parameter list);

Example:

rect1 . length *=15;*
rect1 . *Width =10;*
rect2 . *Length =20;*
rect2 . *Width =12;*
rect1 . *getData(15 ,10) ;*



Example :

```
class Test
{
    void display( )                    //method definition
    {
            System.out.println( "  call method using object " );
    }


}

class D
{
    public static void main( String args[ ])
    {
            Test obj=new Test( );          // object creation
            obj.display( );                // method calling
    }
}
```

Output :

```
call method using object
```

static KEYWORD

- class has two section in which it declares instance variable and instance methods they are accessing using object (. Dot operator).They are called instance because every time the class is instantiated, a new copy of each of them is created.
- Static members are associated with class rather than individual objects , they are sometimes referred as class members and class method.
- Definition: **Static is a member that is common to all objects and accessed without using a any object.**
- We you want to define classmember that will be used independently of any object of class.They can be called without using object.
- **Static methods and varibles are called using only with classname.**

Syntax : classname.variablename ;

classname.methodname ( );

- **We can define static varible,static method and static block.**

1) static varible: static int count;

2) static method :

- Methods that are of general utility but do not directly affect an instance of that class are declared as class method(static methods).

  Math. sqrt( )
- we can define any method as static as follow:

```
static int max( int x , int y)
        {
                //code
        }
```

3) static block

- static block executed before main() method gets executed.

```
static
{
        // code
}
```

Note: non static block will execute only after creating object of class in which that block is defined

<u>Limit to use static methods:</u>

1. They can only call other static methods.

2. They can only use static data(variable).
3. They can not refer to this or super in any way.

Example 1:

```
class Mathop
        {
                static float mul ( float x, float y)
                {
                return x * y;
                }
                static float divide ( float x, float y)
                {
                return x / y;
                }
        }
class StaticDemo
        {
                public static void main( String args[ ])
                {
                float a=Mathop .mul (4.0,5.0);
                float b=Mathop .divide( a , 2.0);
                System .out. println( " b " + b );
                }
        }
```

Output:

```
javac Staticdemo.java
java Staticdemo
        b=10.0
```

Example-2 write a program which executes static block before main method.

```
1) class A
{
        static int a,b;                 //static variable
        A(int x,int y)                  //constructor
       {
                a=x;
                b=y;
       }
```

```
        static void mul()                    //static method
        {
                int c=a*b;
                System.out.println("multiplication is="+c);
        }
        static                               //static block
        {
                  System.out.println("this is static block");
        }
}

  class static_key
  {
            public static void main(String args[])
            {
                      int i,j;
                      i=Integer.parseInt( args[0]);
                      j=Integer.parseInt( args[1]);
                      A a1=new A(i,j);
                      a1.mul();
            }

  }
```

Output :

```
              javac static_key.java
              java static_key  2  3

   this  is static blockk
   mutliplication is 6
```

2)

```
class Staticdemo
{
              static int a=3;
              static int b;
              static void meth(int x)  //static method
              {
              System.out.println("x = " + x);
              System.out.println("a = " + a);
              System.out.println("b = " + b);
              }
              //non static block declaration
              {
                      System.out.println("non static..");
              }
              {
                      System.out.println("non static..");
              }
              //static block declaration
              static
              {
              System.out.println("Static block initialized....1");
              }
              static
              {
              System.out.println("Static block initialized....2");
              b = a * 4;
              }
              public static void main(String args[])
              {
                Staticdemo s=new Staticdemo();
                  meth(42); //call static method
              }

}
```

○ METHOD OVERLOADING

- In java, it is possible to create methods that have same name but different definitions.
- **Definition : when multiple methods of same name with different arguments are defined in a single class is called Method overloading.**
- It is used when objects are required to perform similar tasks but using different input parameters.

- When we call up a method in java using object, it matches up method name first then the number of argument and type of parameters to decide which one of definitions to execute this process is **polymorphism**.
- In definition all method should have same name but different parameter lists (number of parameter or type of parameters).
- In java, we can overload method but not variable or operator.
- Java uses type and number of argument to call a method at runtime. We cannot overload method by changing return type.

Example-1:

```
class Demo
{
    void display( )                    //method definition without argument
    {
    System .out .println( " method overloaded");
    }

    void display (int x)          //method definition with argument
    {
     System .out .println( " x=" + x);
    }
}
class Overload
{
public static void main(String args[ ] )
{
    Demo obj= new Demo( );
    obj.display( );                //method calling without parameter
    obj.display(10);      //calling of method having  parameter
}
}
```

Output:

```
method overloaded
x=10
```

Example:2

```
import java.lang.Math;
class Methodoverloading
{
        public static void main(String args[])
        {
        int a=4,b=2;
        float f1=4.0f ,f2=2.5f;

        System.out.println("Max of"+a+" and "+b+" is:"+Math.max(a,b));
        System.out.println("Max of"+f1+" and "+f2+" is:"+Math.max(f1,f2));
        System.out.println("Min of"+a+" and "+b+" is :"+Math.min(a,b));
        System.out.println("Min of"+f1+" and "+f2+" is: "+Math.min(f1,f2));
        System.out.println("Min of"+a+" and "+f2+" is: "+Math.min(a,f2));
        }
}
```

Output :

```
        Max of 4 and 2 is :4
        Max of  4.0 and 2.5 is 4.0
        Min of  4 and 2  is :2
        Min of 4.0 and 2.5  is: 2.5
        Min of 4  and  2.5 is: 2.5
```

Example :3 : Write a program which overloads sum( ) method.

```
class Demo
{
        void sum( int x,int y)
        {
                System.out.println( "sum of two int values=" + (x+y)  );
        }
        void sum(float x,float y)
        {
                float ans=x+y;
                System.out.println( "sum of float values=" + ans  );
        }
        void sum(int x,int y,int z)
        {
                int ans=x+y+z;
                System.out.println( "sum of three int value=" + ans  );
        }

}
```

```
class Overload
{
        public static void main(String args[ ])
        {
                Demo d1=new Demo( );
                d1.sum(10,20);
                d1.sum(10.4f,20.6f);
                d1.sum(1,2,3);

        }
}
```

Output:

```
sum of two int values=30
sum of float values=31.0
sum of three int value=6
```

this keyword

- **this** keyword hides Instance variable.
- It is used in method body to refer to the members (variables) of current object. (Current object is an object whose method is being called.)
- **this** keyword used in constructor and method.
- **this** is used to (disambiguate) make different variable name if one of argument or parameter has same name as other method(constructor).

Ex:
```
        Circle (int x, int y, int z)
        Circle (int x, int y)
```

- Method argument can have same name as one of class's member variable.
- Typically with a method body you can just refer to directly to the member variable.

Use of this keyword

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

Example :

```
class Circle
{       int  x , y ;
        public Circle (int x ,int y, int radius )
        {
                this. x =x;
                this. y=y;
                this. radius=radius;
        }
}
class  Circle_this
{
public static void main(String args[])
        {
        Circle c1=new Circle( 50,50,25);
        Circle c2=new Circle( 10,10,5);
        System.out. println("C1:"+c1.x + ", " +  c1.y +", "+ c1.radius);
        System.out. println("C2:"+c2.x + " ," +  c2.y +", "+ c2.radius);
}
}
```

Output:

```
C1: 50, 50, 25
C2: 10,10,5
```

2)

```
        class Test
        {
                int a;
                int b;
                //Default constructor
                Test()
                {
                        System.out.println("Inside default constructor ---"+a+ " ..."+b);
                }
                //Parameterized constructor
                Test(int a, int b)
                {

                   this();
                        this.a = a;
                        b = b;
                System.out.println("Inside parameterized constructor---"+a+ " ..."+b);

                }
```

```
		public static void main(String[] args)
		{
			Test object = new Test(10,20);

		}
	}
```

3)

```
	class Test
	{
		int a;
		int b;

		//Default constructor
		Test()
		{
			this(10, 20);
			System.out.println("Inside default constructor ---"+a+ " ..."+b);
		}

		//Parameterized constructor
		Test(int a, int b)
		{
			this.a = a;
			b = b;
		System.out.println("Inside parameterized constructor---"+a+ " ..."+b);

		}

		public static void main(String[] args)
		{
			Test object = new Test();
		}
	}
```

final KEYWORD

- **final** is a keyword.
- final means can not be changed or it is final.
- It is same as const in c++.
- It is used for efficiency.
- final can be used with
  1. Variable (data)
  2. Methods
  3. classes

1. **final variables**

- final specifies that the value of variable is final and must not be changed.
- final variable name must be in capital letters.
- All final variable must have an initial value.(it can't specify later).

  Ex: final int **SIZE**=100;
  final float **PI**= 3.14f;
- final variable behaves like class variable.
- They do not take any space on individual object.

2. final method

- All methods and variable can be overridden by default in subclasses.
- We can prevent overriding of methods of super class by declared as final.
- If we declare super class's method as final then it cannot be override by its subclass.

Example:

```
class A
{
        final void display( )
        {
        ---------
        }
}
class B extends A
{
    void display()   //invalid
    {
        --------
```

```
            }
        }
```

Here, display () method of super class A cannot be overridden by sub class B because it is declared as **final.**

3. final class

- If class is declared with final modifier, then class cannot be extended and this will ensure that the exact properties and behavior of class.
- final class cannot be sub classed.
- If we declare class as final, it prevents any unwanted extension to the class.
- It allows compiler to perform some optimization when method of class is invoked.
- Many classes of java .lang are declared as final.
- Methods of non abstract class can be declared as final.

```
final class A
{
        //body
}
class B extends A          //invalid ,it gives Error
{
        //body
}
```

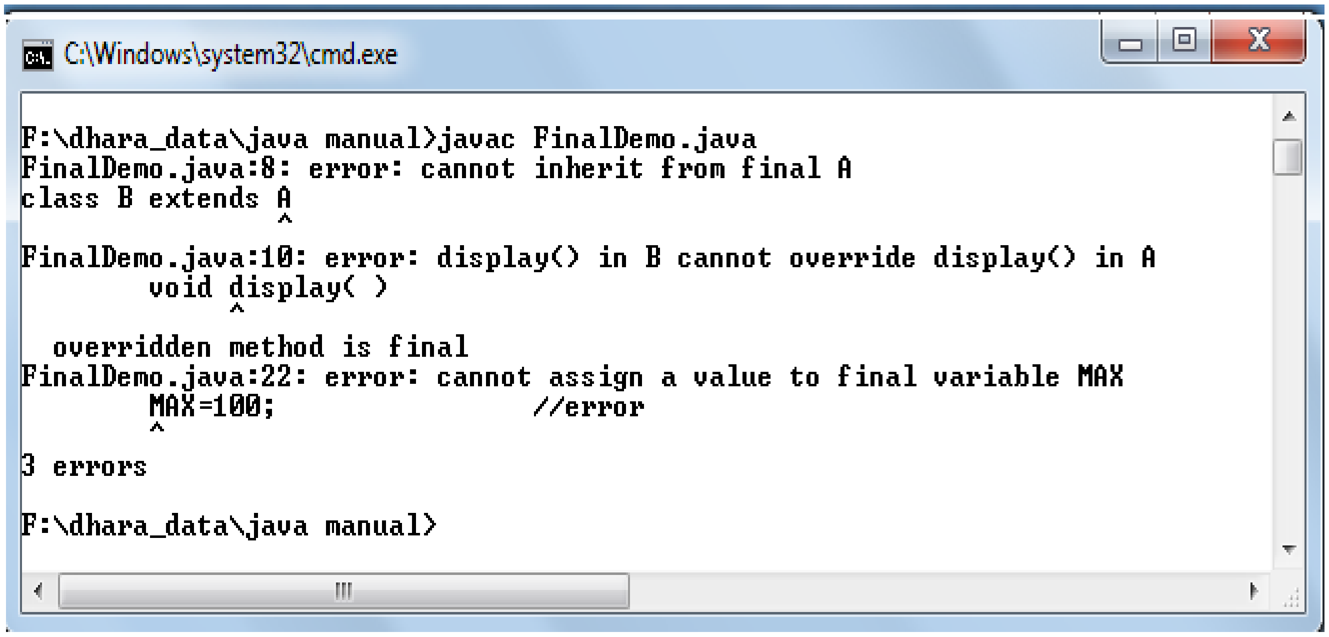
Here, class B cannot extend class A ,because it is declared as **final .**

Example : Write a program which makes use of final keyword.

```
final class A
{
   final void display( )
   {
        System.out.prinltn( " method can't overridden");
   }
}
class B extends A
{
        void display( )
        {
                System.out.println(" same method as super class");
        }
    }

    Class FinalDemo
    {
        public static void main( String args[ ])
        {
        final int MAX=50;
        System.out.println( "MAX="+MAX);
        MAX=100;                //error
        B obj=new B(  );
        obj.display( );         //error
        }
    }
```

Output:



```
F:\dhara_data\java manual>javac FinalDemo.java
FinalDemo.java:8: error: cannot inherit from final A
class B extends A
                ^
FinalDemo.java:10: error: display() in B cannot override display() in A
        void display( )
             ^
  overridden method is final
FinalDemo.java:22: error: cannot assign a value to final variable MAX
        MAX=100;                //error
        ^
3 errors

F:\dhara_data\java manual>
```

3.5 CONSTRUCTOR

Why to use constructor?

- Each object that is created in program must be given initial value.
- We can give values to object using two way:
    1. Uses the (.) dot operator to access the instance variable and then assigns values to them individually.
       Ex *obj* .length=15;
    1. Takes the help of a method like getData ( ) to initialize each object individually.
       Ex. *Obj.* **getData (**15, 10**);**
- **It is simple and convenient to initialize an object when it is first created.**
- Constructor is a *special type of method*.
- **Definition :Constructor is a special method which initialize object when it is created.**
- Constructor enables an object to initialize itself when it is created.

Characteristics:

1. Constructors have the **same name as the class name itself.**
2. Constructors do **not specify a return type,** not even **void** (because they return the instance of the class itself).
3. Constructor cannot be inherited, through a derived class can call the base class constructor.

Types of constructor:

1) default constructor
2) Parameterized constructor
3) copy constructor

1) Default constructor

- When you do not explicitly define a constructor for a class, java creates default constructor.
- **Default constructor** automatically initialize instance variable to **zero**.
- Once you define your own constructor, the default constructor is no longer used.

```
Constructorname ( )
{
        //statement
}
```

```
        Ex:   Rectangle ( )
              {
                    length=50; width=20;
              }
```

Program:

```
    class Rectangle
    {
        int length=50, width=20;

        Rectangle ( )        //defining default constructor
        {
                System.out.prinltn ("constructor demo");
        }

        int rectArea( )
        {
                return ( length * width );
        }
    }

    class ConstDemo1
    {
        public static void main (String args[ ])
        {

        Rectangle rect1=new Rectangle ( );  // calling default constructor
        int area1=rect1.rectArea ( ) ;
        System. out .println ( " Area= " + area1) ;

        }
    }
```

2) Parameterized constructor

- When we pass argument in constructor, it is called parameterized constructor.

```
                Constructorname (parameters)
                {
                        //statement
                }
        Ex:     Rectangle (int x, int y)
                {
```

```
                length=x;
                width=y;
            }
```

Example:

```
class Rectangle
{
    int length, width;

    Rectangle ( int x , int y)  //defining constructor
    {
            length=x;
            width=y;
    }

    int rectArea( )
    {
            return ( length * width );
    }
}

class ConstDemo1
{
    public static void main ( String args[ ])
    {
    Rectangle rect1=new Rectangle (15, 10);    // calling constructor
    int area1=rect1.rectArea ( ) ;
    System. out .println ( " Area= " + area1) ;

    }
}
```

Output :

```
    Area=150
```

3) Copy constructor

- Copy constructor is used to copy the state (data) of one object into other object.
- We can copy constructor into another constructor by passing existing object of that class as an argument in new constructor.

```
        A a1=new A(parameters);
        A a2=new A( a1);          //passing object as parameter in constructor
```

Example :

```
    class A
    {
        int x;
        A(int a)
        {
        x=a;
        }
        A(A obj)
        {
        System.out.println("copy cosnstructor invoked");
        System.out.println("x="+obj.x);
        }
    }
    class Copy1
    {
    public static void main(String args[])
    {
        A a2=new A(10);
        A a3=new A(a2);
    }
    }
```

Output :

```
        Copy constructor invoked
        X=10
```

Example:

```
    class Complex
    {
      double x, y;
      Complex(double a, double b)
      {
        x = a;
        y = b;
    }
    Complex(Complex a)
         {
        System.out.println("Copy constructor called");
        x = a.x;
        y = a.y;
         System.out.println(x+" \n" +y);
    }
}
class Copy
{
    public static void main(String[] args)
 {
        Complex c1 = new Complex(10, 15);
        Complex c2 = new Complex(c1);
    }
}
```

Output:

```
Copy constructor called
10.0
15.0
```

Constructor Overloading

- When multiple constructor of same name having different arguments is defined in a single class is called Constructor overloading.

Ex

```
class Demo
{
        Demo( )            //constructor without parameter
        {
         -----
        }
```

```
                Demo( int a)        //constructor with parameter
                {
                    ------
                }
        }
```

Example:

```
class A
{
    A ( )                               // default constructor
    {
            System. out. println ( “ Constructor demo “ );
    }
    A (int x)                           // parameterized constructor
    {
            System . out .println (“ value of x=“ + x);
    }
}
class ConstructorDemo

{
    public static void main( String args[ ])
    {
    A obj1=new A( );   // calling default constructor
    A obj2=new A(5);  // calling parameterized constructor
    }
}
```

Output:

```
Constructor Demo
Value of x=5
```

Use of private constructor

- we can create private constructor by putting **private** access modifier before constructor definition.
- We cannot call private constructor directly by creating object of class because it is private and private members cannot be accessed outside the class.

- We can call constructor by creating method which will contain object creation code.(as shown in below example)
-

```
class Student
{

    String s1,s2;
    private Student(String s1,String s2)
    {
            System.out.println("i am private constructor");
            this.s1=s1;
            this.s2=s2;
    }
    static Student call(String s1,String s2)
    {
    return new Student(s1,s2); //this method will return instance of Student class
    }
    void show()
    {
            System.out.println(s1);
            System.out.println(s2);


    }
}

class Private_const1
{
    public static void main(String args[])
    {
            //A obj=new A(); this line will give error
            //A obj2=obj.call();
            Student obj=Student.call("GP","Ahemdabad");
            obj.show();
    }
}
```

Passing object as parameter :

We can pass object as parameter in any method or constructor.

```
class A
{
        int a,b;
        static String s="pqr";
        A( )
        {
        System . out . println( "Constructor demo " );
        }

        A( int x)
        {
                a=x;
        }
        A( int x,int y)
        {
                a=x;
                b=y;
        }
        A(A obja)   //OBJECT AS AN ARGUMENT IN CONSTRUCTOR
        {
                a=obja.a;
                b=obja.b;
        }
        void show()
        {
        System . out .println (" value of a="+a);
        System . out .println (" value of b="+b);
        System . out .println (" name="+s);

        System.out.println("--------------------------");
        }
}
class ConstructorDemo
{
        public static void main( String args[])
{
        A obj1;

        obj1.show();

        A obj2=new A(5);
        obj2.s=”abc’;
        obj2.show();

        A obj3=new A(10,20);
        obj3.show();

}
}
```

Returning of object

- A method can return any type of data including class type (object) ,which is created by user.
- In this example, each time call( ) is invoked ,a new object is created and a reference to it is returned to calling routine.

```
class  Test
{
        int a;
        Test ( int i)
        {
                a=i ;
        }
        Test call( )
        {
                Test  temp= new Test (a+10 );
                return temp;   //returns object type of Test class
        }
}
class  Return_obj
{
        public static void main (String args[ ] )
        {
        Test obj1=new Test( 2 );
        Test obj2;
        obj2=obj1.call ( );
        System . out . println( “obj1.a=“ + obj1.a );
        System . out . println( “obj2.a=“ + obj2.a );
        obj2=obj2.call ( );
        System . out . println( “obj2.a after second increase=“ + obj2.a );
        }
}
```

Output:

```
obj1.a=2
obj2.a=10
obj2.a after second increment =22
```