UNIT - I

Need for OOP :-

In early days, programs are collections of procedures. A procedure is defined as a collection of instructions executed in sequential order. Each procedure operates on some data. Data is ~~[illegible]~~ independent of procedures and programmers have to keep track of functions and the way they modify data. This type of programming is called as "structured programming".

In structured programming, a complex program is divided into a set of small tasks. using structured programming, programmers can write complex programs easily. the drawbacks :- 1. As the program size is increasing, the structured programming is failed to get the desired result.
2. programmers are finding new solutions to old problems.

To overcome these drawbacks OOP came into existence. This provides a technique for managing high complexity & achieving reuse of the software.

OOP concept was introduced by Xerox Corporation in 1970's. The first object oriented language was Smalltalk. But it was not fully successful object oriented language. The object oriented languages that are fully successful are C++, Java & Eiffel.

OOP deals with things called 'objects'. An object is the combination of code & data. Each object has a unique name. The data of an object can be accessed by the method which is associated with that object. But methods of one object cannot access the methods of other objects.

Some features of OOP are -

1. importance is on data rather than procedure.
2. programs are divided into objects.
3. data is hidden & cannot be accessed by external functions.
4. one object can communicate with other object through methods.
5. new data & methods can be easily added if necessary.
6. it follows bottom-up approach.

Differences between OOP & procedural oriented programming :-

In procedural, there are techniques to write a code that performs some task. The important technique is functions or subroutines, The large program is divided into some functions, that each performs a specific task. The main program calls these functions and they may call another function.

Data is passed from one non-member function to another and they are available everywhere throughout the program. It allows the user can access the any data directly and he can change the data also. ie data is not well protected in procedural.

OOP binds methods & data into a single unit. The data can be manipulated by its own functions. ie data is protected.

benefits of OOP :-

1. code reusability - ie new objects can be derived from old objects.
2. code modularity - everything in OOP is an object; these objects can be interchanged or removed to meet user needs.
3. easy maintenance
4. design stability - once a stable base class is developed, then the new classes that are derived may have less errors.
5. improves communication ~~by~~ between developer & users.

Applications of OOP :-

1. real time systems
2. Simulation & modeling
3. object oriented databases
4. expert systems
5. neural networks
6. CAD/CAM Systems.

Concepts of OOP :- or Key attributes of OOP :-

1. Objects and classes :-

object are the basic runtime entities in an OOP. They may represent a person, a place, a bank account. When a program is executed, the object interact with other object by sending a message. For example, 'customer' & 'account' are two objects in a banking program. The customer object can send a message to account requesting for the balance. Each object contains data & code.

The set of objects is called a class. So, an object is an instance of a class. A class may be thought of as a 'datatype' & an object as a 'variable' of that data type. once a class is defined, we can create any number of objects belonging to that class.

Data Abstraction and Encapsulation :-

Abstraction means act of representing essential features without including the background details.

eg: in purchasing a car, the customer focuses on ~~wit~~ cost, company & speed of car. But he is not focusing on how many workers are involved & how much time it takes.

The binding of data & method into a single unit is known as encapsulation. The data is not accessible to the outside world & only by the methods present in the class.

3. Inheritance :-

Inheritance is the process by which objects of one class acquire the properties of objects of another class.

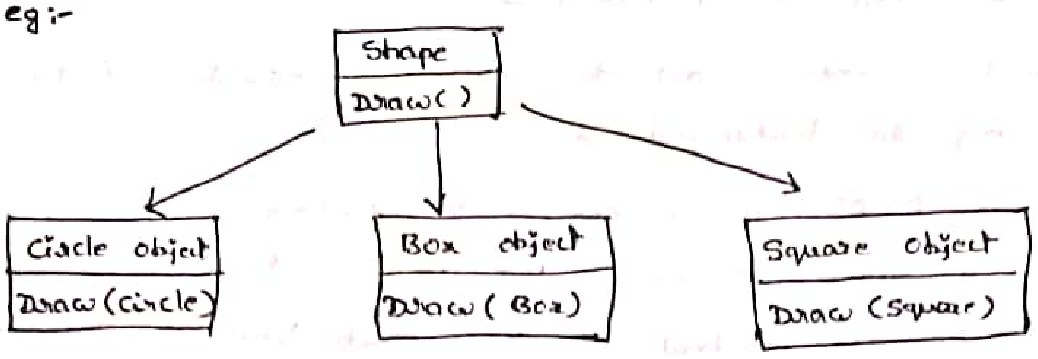
In OOP, inheritance provides the 'reusability'. ie we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing class. The new class will have combined features of both the classes.

4. Polymorphism :-

It means the ability to take more than one form. For example, an operation can have different behaviour in different situations. The behaviour depends upon the types of data used in the operation. For example, consider 'addition' operation. If the operands are numbers, operation generates a sum. If the operands are strings, operation generates a third string by concatenation.

✓ .

eg:-



here, the single method name is used to handle different number & different types of arguments.

5. Dynamic binding :-

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding means that the code associated with a given procedure call is not known until the time of call at runtime.

6. Message Communication :-

It has 3 steps.

a. Create a class.

b. Create object from that class

c. establish Communication among objects.

History of Java :-

James Gosling, patrick Naughton, Chris Warth, Mike Sheridan & Ed Frank created Java at Sun MicroSystems in 1991. Java was initially called oak. In 1995, it was renamed as Java. The motivation of Java was the need for a platform independent language which can be used to create a software to be, used in different consumer electronic devices, like remote controls, microwave ovens etc. This team began to work on portable, platform independent language which could be used to produce code that would run on a variety of CPU's. This leads to creation of Java. In 1993, this team identifies that the problems of portability prelated to electronic devices are also found when attempting to create code for the internet. So that, Java was switched from electronic devices to Internet.

Due to the similarities between C++ & Java, Java is called as " Internet version of C++ ", but " Java is ~~an~~ not extension version of C++ ".

Differences between Java & C :-

1. Java does not support keywords like goto, sizeof, typedef.
2. " " datatypes struct, Union, enum
3. " " type modifiers auto, extern, register, signed & unsigned.
4. " " pointers.
5. " " preprocessor, so we cannot use # define, # include statements.

✓

6. Java supports object & classes
7. " inheritance
8. " polymorphism
9. " dynamic binding.

Differences between Java & C++ :-

1. Java doesnot support operator overloading
2. " " template classes as in C++.
3. " " multiple inheritance, this is achiev by using interface concept.
4. " " global variables.
5. " " pointers
6. " " header files.

=

Java buzzwords :-

1. object - oriented : Everything in Java is in terms of object. data & code present within the object. Java is said to be a true object oriented language.

2. Compiled and interpreted : a computer language will be either compiled or interpreted, but Java combines both these. In first stage, Java compiler translates source code into byte code, which is not machine code. Then Java interpreter generates machine code, which is ready to execute. So, Java is both compiled and interpreted language.

3. platform independent : Java programs can be run on any CPU, on any platform. If we do changes in CPU's and operating systems, then there will be no changes in the Java programs.

4. portable : Java programs can be moved from one system to another, anywhere & anytime easily.

5. Distributed : Java is a distributed language, because it has the ability to share data & programs. Java programs can open & access remote objects on internet very easily. So, that different users from different locations can work on a single project.

6. Robust and Secure : Java provides many safeguards in order to ensure reliable code. Java has strict compile time & run time checking for datatypes. So, Java is said to be robust language.

Java not only verifies the memory access, but also ensures that no viruses communicate with an applet. So, Java is a secure language.

7. Familiar, Simple & Small : If we know the object oriented concepts, then easily we can learn & we can understand the Java language. If we know the C++, then we can learn Java language easily. So, Java is a small & simple.

8. multithreading : Java supports multithreading ie it is not necessary for an application to finish one task before starting another. In multithreads, multiple threads (tasks) can execute simultaneously.

✓

9. Dynamic and Extensible : Java is a dynamic language, which is capable of dynamically linking in new class libraries, methods & object. Java also supports extensibility, because it support functions written in other languages like C & C++.

10. High performance : because of intermediate byte code, Java performance is excellent for interpreted language.

Structure of a Java program :- (Syntax)

A Java program consists of one or more classes. only one of these classes defines the main() method. A class consist of data & methods.

Syntax is -

```
Documentation
package Statement
Import statements
Interface Statements
Class definitions
main method class definition
  {
      main() method definition
  }
```

1. documentation section : It consists of a set of comments about the program, like name of the ~~Cam~~ program. This section is suggested, because it helps in understanding the program.

2. package Statements : This is the first statement in every Java program. This statement tells the Compiler that the Classes defined here belongs to this package. This statement is optional.

   eg : package pack1;

3. import statements : This statement tells the interpreter to include the classes from the package defined. This is the next statement after the package declaration & before the class definition. There may be a number of import statements. This statement is optional.

   eg : import java.io.*;

Scanned with CamScanner

4. interface Statements : It defines method declaration without body for the subclasses to provide implementation. This section is optional . It is useful when we want to Implement multiple inheritance.

5. class definition : This section Consist of a number of class definitions where each class Consists of data & methods.

6. main method class : This is the essential section of Java program. Every Java program must have a class definition that defines the main () method. This is essential because main() is the starting point for running Java programs. The program terminates on reaching the end of the main() method.

Simple Java program :-

```
1.      // A first Java program
2.      class Simple
3.      {
4.          public static void main ( String args[] )
5.          {
6.              System. out. println (" Welcome to Java");
7.          }
8.      }
```

line 1 : documentation section, ie Information about program.

2 : declaration & definition a class. 'Simple' is the name of the class.

3 : beginning of class definition

4 : this line defines a method called main(). Every Java program must Contain main() method. This is the starting point for the execution. A Java program Can have any number of classes, but only one of them must Contain main().

✓

Keywords are –

public – It is an access specifier, main() method is accessed by all other classes.

static – it declares this method as one that belongs to the entire class. The main() method must always be declared as 'static'

void – it declares that main() method does not return any value.

String args[] – it declares a parameter named 'args', which contains an array of objects of type 'String'.

line 5 : beginning of main() method

6 : this is similar to printf() statement in 'C' & Cout << in 'C++'. 'println' method is a member of 'out' object, which is a member of 'System' class.

7 : ending of main() method.

8 : ending of class definition

Take input from the standard input device. Then the following statement is called before taking input values.

```
BufferedReader br = new BufferedReader (new
                    InputStreamReader (System.in));
```

Connect the standard keyboard to input stream object. Here, we use 'InputStreamReader' that can read data from the keyboard. So, create one object of 'InputStreamReader' and connect it to keyboard.

ie `InputStreamReader obj = new InputStreamReader (System.in);`

Connect InputStreamReader to BufferedReader, which is another input type of stream. So, create object of BufferedReader and connect it to the object of InputStreamReader.

ie BufferedReader br = new BufferedReader (obj);

These two steps can be combined into a single step as –

```
BufferedReader br = new BufferedReader (new InputStreamReader
                                         ( System. in));
```

Now, we can read data from the keyboard using read() and readLine() methods.

The accepted input is in the form of a string. This should be converted into 'int' by using parseInt() method, which is a method of 'Integer' class.

ie int n = Integer. parseInt (br. readLine());

program 1 :

```
class one
{
 public static void main ( String args[ ])
 {
   int a = 5, b = 10, c;
   c = a+b;
   System. out. println (" the sum = " + c);
 }
}
```

✓

program 2 :
```
import java.io.*;
class two
{
 public static void main (String args[])
 { int a, b, c;
   BufferedReader br = new BufferedReader (new
                    InputStreamReader (System.in));
   ~~Print~~ System.out.println ("enter a value");
  int a = Integer.parseInt (br.readLine());
  System.out.println ("enter b value");
    b = Integer.parseInt (br.readLine());
  c= a+b;
  System.out.println (" sum =" + c) ;
 }
}
```

program 3:
```
import java.io.*;
class three
{
 public static void main (String args[])
  {
   BufferedReader br = new BufferedReader (new
                   InputStreamReader (System.in));
  System.out.printlan (" enter a value");
  int a = Integer.parseInt (br.readLine());
  System.out.println (" Square of a =" + a*a);
  }
}
```

=

(V)

Concepts of classes :-

A class is defined as a skeleton or blueprint or template for an object. A class defines all the variables & methods, an object should have. An object is an instance of a class. consider an example, let 'fruit' is a class, then mango is an object of class 'fruit'.

Class is a userdefined data type. once a class is defined, then any number of objects can be created from it. A class contains variables which indicate the attributes of object and methods that operate on the variables.

A class is declared by 'class' keyword.

Syntax of a class is -

```
Class classname
{
   type      instance - variable 1;
   type      instance - variable 2;
             :
   type      instance - variable n ;
   type      method - name 1 ( list of parameters)
   {
         // code  for  method 1
   }
```

```
        :
    type  method-name n  (list of parameters)
    {
          // code for method n
     }
}
```

where - 'classname' is the name of the class
'type' is the datatype of variable
'type' is the datatype of return type of the method.
'method-name' is the name of the method

program:

```
class employee
{
    String name = " xyz" ;
      int salary = 10000 ;
      int age = 30 ;
    void display ()
    {
        System.out.println (" employee name = " + name);
                 "         (" Salary = " + salary);
                 "         (" age = " + age);
    }
   public static void main ( String args[])
   {
        Employee e = new employee();
                 e. display();
   }
}
```

data hiding :-

Data hiding is accomplished with the help of encapsulation mechanism. This mechanism is responsible for binding the manipulated code & data. This binding will keep the data & code safe from arbitrary accesses.

Classes accomplish datahiding with the help of public & private access specifiers. If a method is declared as 'private', then only its members can access this method.

Objects :-

An object is an instance of a class. A class contains data & code. Objects are created using 'new' operator. The 'new' operator creates an object of the specified class & returns a reference to that object.

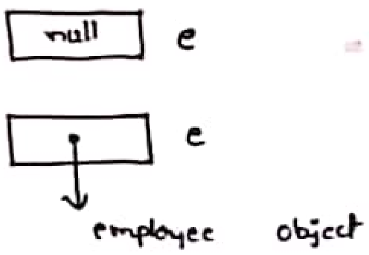
eg: employee e; // declare

e = new employee(); // create an instantiate

It creates a object of class 'employee'. The first statement declares a variable to hold the object reference. The second line assigns the object reference to the variable. Here, 'e' is an object of class 'employee'.

Action step 1 :

Step 2 :



The above two lines can be combined into one line as-

employee e = new employee();

we can create any number of objects of 'employee'.

eg: employee e_1 = new employee ( ) ;
employee e_2 = new employee ( ) ;

Each object has its own copy of the instance variables of its class. This means that any changes to the variables of one object have no effect on the variables of another object. It is possible to create two or more references to the same object.

we can access the instance variable & method by using 'dot' (.) operator. Syntax is -

objectname . variablename ;
objectname . methodname (parameter_list) ;

where 'objectname' is the name of the object, 'variablename' is the name of the instance variable inside the class, 'method name' is the method ~~we~~ that we want to call, 'parameter_list' is the list of values passed to the method.

eg: e. salary ;
e. age ;
e. display ( ) ;

=

Constructors :-

Java allows objects to initialize themselves when they are created. This initialization is done with the help of Constructors. A Constructor is a special type of method that has the same name as that of its class. whenever an object of a class is created, its constructor will be invoked and it will initialize the object. The return type of a constructor is not defined, implicitly it takes type of class.

The 'new' operator creates an object of a class and the newly created object invokes the constructor. The constructor initializes the state of an object.

eg: the following statement creates two objects of 'employee' class.

```
employee e1 = new employee();
employee e2 = new employee(" xyz", 10000, 30);
```

The first statement creates object e1 and ~~class~~ calls the default constructor. A default constructor does not accept any arguments and initializes all the instance variable of class of zero. The second statement creates the object e2 & calls the parameterized Constructor. The Parameters to the 'employee' supplies any arguments needed in the initialization of the ~~ele~~ object.

program :

```
class rectangle
{
    int length, width;
    rectangle (int x, int y)      // defining Constructor
    {
        length = x ;
        width = y ;
    }
```

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

    class   rectanglearea
    {
        public, static   void   main ( String   args[])
        {
            rectangle   r  =  new   rectangle(10,15);   // calling constructor
                  int    a  =  r. area();
            System. out. println(" Area = " + a);
        }
    }
```

properties of a Constructor :-

1. it has no return type, not even void
2. it creates an instance of a class
3. it cannot be abstract, final or static
4. its name is same as that of its class & by convention starts with capital letter.
5. constructors cannot be inherited.
6. By default, a constructor with no arguments is provided.

=

Methods :-

A method is provided by classes for packing a group of logically related data items and functions which work on data items. These data items & functions are called fields & methods. We use methods for manipulating data present in the class. We declare methods inside the body of a class, but immediately after the declaration of instance variables.

General form of a method is. -

```
type methodname (list of parameters)
{
        // body of method
}
```

There are 4 parts of method declaration. They are -

1. method name
2. the return type of method
3. list of parameters
4. body of the method.

In the above syntax, 'type' specifies the return type of method. 'list of parameters' is enclosed in parenthesis, which contains variable names & types of all the variables. 'body of method' specifies the operations to be performed on the variables. We can declare method as 'final' ie the method is not redefined in a subclass.

eg :-

```
class rectangle
{
        int length;
        int width;
        void getdata (int x, int y)
        {
            length = x;
            width = y;
        }
```

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

Here 'getdata' is a method which return type is 'void', because it does not return any value. we pass two integer values to the method which are assigned to instance variables 'length' & 'width'. The method 'area' computes area of the rectangle & returns the result. Since, the return type is ~~int~~ integer, the return type of method is taken as 'int'.

program :
```
        class  box
        {
            int  w, h, d;
            void  volume ( )
            {
                System. out. println (" volume =" +  w*h*d);
            }
        }
    class  one
    {
        public  static  void  main ( String  args[])
        {
            box  b1  =  new  box( );
            box  b2  =  new  box( );
            b1 . w  =  10;
            b1 . h  =  2;
            b1 . d  =  4;
            b2 . w  =  5;
            b2 . h  =  6;
            b2 . d  =  6;
            b1 . volume( );
            b2 . volume( );
        }
    }
```

parameter passing to a method :-

The parameters that are specified within the parenthesis of a method call are called as actual parameters. These actual parameters are passed to a method while calling that method.

eg: e. display ("xyz", 10000, 30);

this statement calls the method 'display' using a object 'e'. Then it passes the specified actual parameters to that method.

The parameters that are specified in the method definition are called as formal parameters. The actual parameters can have the same name as formal parameters, but the datatypes of actual & formal parameters must be same.

program:
```
class two
{
    two (int a, int b, int c)          // a,b,c are formal
    {
        volume = a*b*c;
        System.out.println ("volume is ="+ volume);
    }
}

class one
{
    public static void main (String args[])
    {
        int a=10, b=20, c=3;           // actual parameters
        two obj = new two(a,b,c);
    }
}
```

Access Control :-

We know that, it is possible to inherit all the members of a class by its subclass. And also, the variables & methods of a class are visible everywhere in the program. In some situations, it is necessary to restrict the access to certain variables and methods from outside the class. For example, we want that the objects of a class cannot access the value of a variable or cannot access the method. In Java, we can achieve this by using 'visibility modifiers' to the variables & methods. They also known as 'access modifiers'. Java provides 3 types of access modifiers - public, private & protected.

public access :-

Any variable or a method is visible to the entire class in which it is defined. If we want to make it visible to all the classes, then it is declared as 'public'.

eg :
```
public int a;
public void Sum ( )
{
    ≡
}
```

here variable 'a' & method 'sum' are declared as public, so they can be accessible everywhere.

private access :-

private fields have the highest degree of protection. They are accessible only within their own class. They cannot be inherited by subclasses, and so they cannot accessible in subclasses. A method declared as 'private' behaves like a method declared as 'final'.

~~protected access~~:-

friendly access :-

In the previous examples, we have not used 'public' modifier, yet they are still accessible in other classes in the program. When no access modifier is specified, the member by default is taking as 'friendly' access.

The difference between 'public' & 'friendly' access is that 'public' modifier makes fields visible in all classes, regardless of their package. 'friendly' access makes fields visible only in the same package, but not in other packages.

protected access :-

The visibility of 'protected' field lies in between the public access & friendly access. ie 'protected' modifier makes the fields visible not only to all classes & subclasses in the same package, but also to subclasses in other packages.

private protected access :-

A field can be declared as both private & protected as -

private protected int number;

It gives visibility in between private & protected access. It makes the fields visible in all subclasses regardless of what package they are in. These fields are not accessible by other classes in the same package.

location ↓	public	protected	friendly	private protected	private
Same class	yes	yes	yes	yes	yes
Subclass in same package	yes	yes	yes	yes	NO
other classes in same package	yes	yes	yes	NO	NO
Subclass in other package	yes	yes	NO	yes ✓	NO
other ~~sub~~ classes in other packages	yes	NO	NO	NO	No

program :-

```
    class two
    {
        int a;           // default access
        public int b;        // public access
        private int c;       // private access
        void setc (int i)
        {
            c = i;
        }
        void get c ( )
        {
            return c;
        }
    }
class one
{
  public static void main (String args[])
  {
        two obj = new two( );
        obj . a = 10;    }  these are OK, a & b accessed directly
        obj . b = 20;    }
        obj.c = 100;          // this is not ok & so error
        obj.setc (100);       // ok, we can access 'c' through its methods
        System.out.println ("a,b & c =" + obj.a + "   " + obj.b +
                               "  " + obj.getc());
    }
}
```

=

'this' keyword :-

when a method wants to refer to the object which invoked it, then we use 'this' keyword. we can use 'this' keyword inside any method for referring the current object. This keyword is always a reference to the object on which the method was invoked.

program :-

```
public class one
{
    int x,y;
    one (int x, int y)
    {
        this.x = x;
        this.y = y;
    }
    public static void main (String args[])
    {
        one obj = new one (10,20);
        System.out.println( obj.x );
        System.out.println (obj.y);
    }
}
```

o/p :
```
10
20
```

=

Overloading methods and Overloading Constructors :-

Overloading methods :-

Method overloading is the process of writing more than one method with the same name with different parameters and different datatypes. Method overloading is used when objects are required to perform similar tasks, but using different parameters. When we call a method in an object, Java matches the method name first and then the number and type of parameters to decide which one of the definitions to execute. Thus overloaded methods must differs in the type and/or number of arguments.

When an overloaded method is called, then Java Compiler matches the arguments of the method called & the method's arguments. This match need be always exact. For example, if a method is called with an int type argument & there is overloaded method of double type, then the method of double type will be called. This is because Java has capability of automatic type conversion, only if no exact match is found.

program :-

```
        class   two
        {
            void  test( )
            {
                System.out.println (" No parameters");
            }
            void  test (int   a)
            {
                System.out.println ("a :" + a);
            }
            void  test ( int   a ,  int   b)
            {
                System.out.println (" a and  b :" + a + "   " + b);
            }
            void  test ( double   a)
            {
                System.out.println (" double   a :", + a) ;
            }
        }
        class   one
        {
            public   static  void  main ( String  args[])
            {
                two   obj  =  new   two( ) ;
                obj. test( ) ;
                obj. test (10) ;
                obj. test (10,20) ;
                obj. test  (10.123) ;
            }
        }
```

o/p

```
No  parameters
a =10
a  and  b  :  10    20
double   a  =  10.123
```

Overloading Constructors :-

Constructor overloading is the process of writing more than one constructor with the same name, but with different types and different number of arguments. A constructor does not have the return type. To overload a constructor, write constructor of a class with same name but with different parameters. constructors are overloaded to initialize the objects of class in variety of forms.

The proper constructor is invoked by matching the number, types and order of arguments passed in the method with that specified in the constructor call. The matching constructor is called automatically.

program :-

```
    class Box
    {
        double width, height, depth;
        // constructor used when all dimensions specified
        Box ( double w, double h, double d )
        {
            width = w;
            height = h;
            depth = d;
        }
        // constructor used when no dimensions specified
        Box ( )
        {
            width = -1;        // use -1 to indicate uninitialized box
            height = -1;
            depth = -1;
        }
        // constructor used when cube is created
        Box( double len )
        {
           width = height = depth = len ;
        }
        // compute & return volume
        double volume( )
        {
             return   width * height * depth ;
        }
}
class   overloaded
{
      public  static  void  main ( String  args[] )
      {
        // create boxes using various constructors
        Box  obj1 =  new  Box (10,20,15);
        Box  obj2 =  new  Box() ;
        Box  obj3 =  new  Box(6);
        double  vol ;
        vol  =  obj1 . volume() ;
        System.out.println ("volume of mybox1 is = " + vol);
        vol  =  obj2. volume() ;
        System. out. println ("volume of mybox2 is = " + vol);
        vol = obj3. volume();
        System. out. println (" volume of cube is = " + vol );
      }
}
```