

Institute of Engineering & Management, Kolkata

MANNUAL & ASSIGNMENT

FOR

OBJECT ORIENTED PROGRAMMING WITH JAVA LAB

3rd year 1st Sem IT

3rd year 2nd Sem CSE

Written by : Sourav Mukherjee

Session - I

What is Object-Oriented Programming ?

Ans. : Our real world is full of objects. Living objects include birds, animals, human etc. Objects without life include furniture, buildings, food, dress etc. Each of these objects have certain properties. The properties of the objects describe the state of the object. Certain operations can also be performed on these objects. The operations that can be performed, describe the behavior of the objects. For example, a bird has wings. It can fly. Humans have brain, legs. They can walk, talk and think. Furniture have leg, arms. They can be moved from one place to another place. Object – oriented programming involves representing objects in a programming language and use them. C++, java, SmallTalk, .NET are example of object-oriented programming language.

What are classes?

Ans.: A class represents a group of objects of the same kind. For example class of animals represent – cow, dog, cat, tiger etc. A class is blue print or template for creating multiple objects.

What are objects?

Ans.: An object is an instance of a class. For example Maruti is an instance of the class car. Zen is another instance of the class Cars. So Maruti and Zen are objects here. You can observe all the cars have similar features such as have wheels, have break, have seat etc.

How JAVA works ?

Ans.: A compiler converts the java program into an intermediate language representation called Bytecode which is platform independent. A java file will have extension .java, like word file has .doc, text file has the .txt extension.

The concept of “write once, run anywhere” is possible in java. The Java program can be compiled on any platform having a java compiler. The resulting bytecodes can then be run on Windows 98, Window’s 2000 or Solaris or Macintosh or any other machine. The machine should have a Java platform to run Java code. Java platform consists of Java Virtual Machine (JVM) and a package of ready-made software components. This package is known as the Java Application Programming Interface (Java API). The compiled java program can run on any hardware platform having Java Virtual Machine(JVM) install on it.

How you compile java program?

Ans.: Write the following program named firstClass.java through edit and save it to jdk1.3/bin directory.

1. import java.io.*;

```
public class firstClass {  
    public static void main(String[] args) {  
        System.out.println("Hi ! This is my First Program in Java") ;  
    }  
}
```

2. c:\jdk1.3\bin> **javac firstClass.java**

javac will convert it to bytecode. Then you find firstClass.class

3. c:\jdk1.3\bin>**java firstClass**

The following program is to input a int value from keyboard and display it.

Example : 1

```
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;

public class inputInt {
    public static void main(String[] args) throws IOException {
        BufferedReader keyBoardInput=new BufferedReader ( new
        InputStreamReader(System.in ));
        System.out.println("Please enter an Integer value ");
        int val1=Integer.parseInt(keyBoardInput.readLine() );
        System.out.println("Value is :"+Integer.toString(val1) );
    }
}
```

Example : 2

```
/* Program to add two number */

class AddNum{

public static void main(String args[]){
    int a=10;
    int b=20;
    int sum=a+b;
    System.out.println("Sum="+sum);
}
}
```

Example: 3

```
/* Write a program to find largest of integer */
class Larg {

public static void main(String args[]){
    int x1=10;
    int x2=20;

    if (x1>x2)
        System.out.println("x1 is larger than x2");
    else
        System.out.println("x2 is larger than x1");
    }
}
```

Assignment

1. Write a program in java to calculate the factorial of 5.
2. Write a program in java to convert miles to kilometers.
3. Write a program to find out the greatest value among the three int.

Session - II

Array

Arrays are a means of storing a list of items. Multiple values of the same type can be stored together. Arrays can hold data of similar type.

To use an array we require the following to be done.

- Declare a variable to be used as the array name.
- Create an array object and assign to the variable.

```
int values[]=new int[10];
```

This creates an array with the name values, the type of the values stores in integer and the number of elements that can be store in the array is 10. This is also called **dimension** of the array. When the array is created initially all the values are initialized to **zero**. The individual elements of the array can be access by using an index with the array name. For example the first element of the above array is referred as values[0], second element is referred as values[1] and so on.

Arrays can contain any legal java data type including reference types such as objects or other arrays. For example, the following declares an array that can contain ten String objects.

```
String[] str1 = new String[10];
```

Or

```
String str1[] = new String[10];
```

Creating and initialization of an array can be done at the same time as follows:

```
int[] values=new {5,6,7,8,9,10};
```

Two dimensional arrays are used as follows.

```
int[][] myArray = new int[5][5];
```

Example 2.1

// Program to show the use of Continue statement.

```
class ArrayUse{
    public static void main(String args[]){
        int[] values = {5,6,7,8,9,10};
        for(int I=0;I<6;I++)
            System.out.println("Values["+I+"] = "+values[I]);
    }
}
```

Example 2.2

//Program shows how to use command line argument

```
public class cmdLineArgv {
    public static void main(String[] args) {
        int n = args.length ;
        if (n !=2)
        {
            System.out.println("Error Input") ;
            System.out.println("Syntex : cmdLineArgv val1 val2") ;
            System.exit(0) ;
        }
        int i=Integer.parseInt(args[0]);//In that case the all values of args
are string, you mast convert it
        int j=Integer.parseInt(args[1]);
        int v=i+j;
        System.out.println("The Sum of "+args[0]+" & "+args[1]+" is "+v) ;
    }
}
```

Exercise

- 2.1 Write a java program to sort n integer numbers and display all the number. The value should given through command line.

java sortValue 10 20 15 20 30 25 12 14

What is static?

Ans.: There will be times when we will want to define a class member that will be used independently. A class member must be accessed only in conjunction with an object. However, it is possible to create a member that can be used by itself, without reference to a specific instance. To create such a member, precede its declaration with the key static. When a member declare static, it can be accessed before any objects of its class are created, and without reference to any objects. We can declare both member and variables to be static. The most common example of a static is main(). main() is declare as static because it must be called before any objects exist.

Instance variables declared as static are, essentially, global variables. When objects of its class are declared, no copy of static variable is made, all instance of the class share the same static variable.

Methods declared as static have several restrictions:

- They can only call other static methods
- They must only access static data.
- They cannot refer to this or super in any way(The keyword super relates to inheritance)

Simple Class

Here is a class called **Point** that define two variable x , y and no method.

A class defines a new type of data. In this case, the new data type is called **Point** . You will use this name to declare objects of type Point. It is important to remember that a class declaration only creates a template; it does not create an actual object.

To actually create a Point object, you will use a statement

Point point = new Point();

To assign an variable
point. x = 2;

Example 2.3

```
//Program is a example of simple class
class Point{
    int x;
    int y;
}
public class simpClass {
    public static void main(String[] args) {
        Point point = new Point();
        point.x = 10;
        point.y = 20;
        int z= point.x + point.y ;
        System.out.println("The sum =" +z);
    }
}
```

Each object has its own copies of the instance variables. This means that if you have two **Point** objects, each has its own copy of **x** & **y**. It is important to understand that changes to the instance variables of one object have no effect on the instance variables of another. For example, the following program declares **two** Point objects:

Example 2.4

```
//Program declares two class and add a method.
class Point{
    int x;
    int y;
    void addVal(){
        System.out.print("The sum is :") ;
        System.out.println(x+y) ;
    }
}
public class simpClass {
    public static void main(String[] args) {
        Point point = new Point();
        Point point1 = new Point();
        point.x = 10;
        point.y = 20;
        point.addVal() ;
        point1.x = 5;
        point1.y = 30;
        point1.addVal() ;
    }
}
```

A variable declared can be used only within the block it has been declared. It has a limited scope within which it can be used. When a variable is referred within a method, java checks for the definition of the variable within a current scope and then in the next outer scope and then up to the current method definition. If the variable is not local variable, that is if it is not defined within the current method then java checks for the definition of that variable as an instance or class variable in the current class. If the definition is not found still then it is checked in each superclass in turn. Consider the case when same variable name is used for both local variable and instance or class variables. However the value of instance or class variables can be accessed by using the keyword ***this*** along with the variable name. The example below shows the use of ***this*** keyword.

Example 2.5

```
public class TestThis {
    int x = 10;
    int y = 20;
    int z = 30;
    void displayValue(){
        int x = 5;
        int y = 10;
```

```
int z = 20;
System.out.println("Printing Loc. Ver:");
System.out.println(x+" "+y+" "+z+" ");
System.out.println("Printing instance Variable");
System.out.println(this.x+" "+this.y+" "+this.z);
}
public static void main(String[] args) {
TestThis temp = new TestThis();
temp.displayValue();
System.out.println("Printing value in main()");
System.out.println(temp.x+" "+temp.y+" "+temp.z);
}
}
```

Exercise:

2.2 Create a object named studData. The property of that object is int roll, String name, int sub1,sub2,sub3 and two methods markAdd(), dispVal(). Now you create a class name Stud and calculate the percentage,total, name, roll, marks of three subject and of a student. The value should given through keyboard.

Session - III

Overloading Methods

You can have same name for more than one method. The number of arguments or the type of arguments are to be different for creating two or more methods with the same name. The return types of the methods can be different as long as the method signature is different. The example below shows overloading of a method called add. The first add() takes integers as arguments and returns the sum as floating point number. The second add() takes floating point numbers as arguments and returns the sum as a floating point number. The third add() takes string as arguments, converts them to integer add then return sum as floating point number. The program below shows overloading of methods.

Example 3.1

// This program shows overloading of method.

```
class OverLoadMethod{

static float add(int x,int y)
{
    return x+y;
}

static float add(float t1, float t2)
{
    return t1+t2;
}

static float add(String s1, String s2)
{
    float sum;
    sum = Integer.parseInt(s1)+Integer.parseInt(s2);
    return sum;
}

public static void main(String args[]){
    int x=10, y=20;
    float m = 5.5f;
    float n = 10.5f;
    String s1="25";
    String s2="35";
    System.out.println(add(x,y));
    System.out.println(add(m,n));
    System.out.println(add(s1,s2));
} }
```

Constructor Methods

Constructor method is called when an object is created. Constructor methods are called automatically by java. Constructor methods are written keeping the following points in mind.

- There can be more than one constructor method. But the number or type of arguments have to be different.
- Constructor methods have the same name as the name of the class.
- Typically they are used as to set initial values to instance variables.

Example 3.2

// Program to show the use of constructors

```
class Point{
    int x;
    int y;
    Point(int x1, int y1)
    {
        x=x1;
        y=y1;
    }
} // end of Point class

class Circle{
    int originX=5;
    int originY=5;
    int radius=3;
    // Default constructor. This is to be defined since it is
    overridden
    Circle(){
    }
    Circle(int x1, int y1, int r)
    {
        originX=x1;
        originY=y1;
        radius=r;
    }
} // Constructor taking first parameter as the type Point and the second
as an int
Circle(Point p, int r){
    OriginX=p.x;
    OriginY=p.y;
    radius = r;
}
void display(){
System.out.println("Center at "+originX+"and"+originY);
System.out.println("Radius = "+radius);
}
```

```
public static void main(String args[]){
    Circle c1=new Circle();
    Circle c2=new Circle(10,25,5);
    Circle c3=new Circle(new Point(15,35),10);
    c1.display();
    c2.display();
    c3.display();
}
}
```

Subclasses/Inheritance/Super class/Extends

When more than one let us say Class1 and Class2 have some properties to be shared, then they could be defined in a single class let us say Class0 as if they belonged to themselves. Here Class1 and Class2 are called **subclasses** and Class0 is called as **super class**. The keyword **extends** is used after the name of class. The keyword extends is followed by the name of the super class. The concept of using methods and variables defined in another class is called as **inheritance**. In java only single inheritance is allowed that means classes can inherit methods only from one intermediate super class. Inheritance provides a chain in which a class inherits not only from its immediate super class, but also super class upwards.

Example 3.3

//Example for Inheritance

```
public class superClass {
    void addNum(int x,int y){
        int sum;
        sum=x+y;
        System.out.println("Sum of two number is "+sum);
    }
    void disp()
    {
        System.out.println("From superclass") ;
    }
} //End of superClass
```

```
class SubClass extends superClass{//new Class subclass call superClass
    public static void main(String[] args) {
        SubClass s1 = new SubClass();// You call the subclass not superclass
        s1.disp();// Here you call the method of superClass.
        s1.addNum(10,40);
    }
}
```

Method overriding

When an object's method is called, java looks for the method definition in the objects class. If it can not find, then it checks one level up in the hierarchy of classes. Consider the case when the same method name is used in both the subclass and superclass with the same signature (same number of argument with the type) . Here when the method is called, method defined in the subclass is invoked. The method defined in the super class is overridden. It is now hidden for the objects of the subclass. If the method in the super class has to be used, then the super keyword can be used along with the name of the method. In the example the method display() and this.display() will invoke method display() of the superclass.

Example 3.4

```
//Example for Method overriding
public class superClass1 {
    void display()
    {
        System.out.println("From superclass:");
    }
}
class OverRide extends superClass1{
    void access(){
        System.out.println("Displays from a different place");
        display();// Search locally first
        super.display();//From super class.
        this.display();//search locally first
    }
    public static void main(String[] args) {
        OverRide s1 = new OverRide();
        s1.access();
    }
}
```

Final class, methods and variables

The final modifier indicates that the entity can not be changed.

- A final class can not be subclassed.
- A final method cannot be overridden
- A final variable cannot change it's value.

Variables are declared as final when their values does not change. They are also defined as static since the constant values in a class will be generally classwide information.

Final methods run efficiently. Java class library declares many of the commonly used methods final so that they can be executed more quickly.

A final class cannot be subclassed by another class. This gives some speed benefits. Many popular java classes such as java.lang.Math, java.lang.String are final. All methods in a final class are automatically final.

Abstract Classes

When the keyword abstract appears in a class definition, it means that zero or more of its methods are abstract. An abstract method has no body. Some of the subclass has to override it and provide the implementation. Objects cannot be created out of abstract classes. Abstract classes basically provide a guideline for the properties and methods of an object. In order to use abstract classes, they have to be subclassed. If all the methods in the abstract classes are abstract that is without any implementation, then it is better to implement as an interface than as an abstract class.

Example 3.5

```
//Example for Method overriding
abstract class AbstractClass {
    void display()
    {
        System.out.println("From abstract class");
    }
    abstract void printMSG();// abstract method
}
class TestABS extends AbstractClass{
    void printMSG(){
        System.out.println("Example for ABS class");
    }
    public static void main(String[] args) {
        TestABS s1 = new TestABS();
        s1.display();
        s1.printMSG();
    }
}
```

Exercise:

- 3.1 Write a program in java to create a sum function. The argument of the sum function may 2 or 3.

Session – IV

(For two week)

Exception Handling

The java language uses exceptions to provide error-handling capabilities for its programs. An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions. It is basically a shorthand to indicate exceptional event.

When such an event occurs within a Java method, the method creates an exception object and hands it off to the runtime system. The exception object contains information about the exception, including its type and the state of the program when the error occurred. The runtime system is then responsible for finding some code to handle the error. In java terminology, creating an exception object and handing it to the runtime system is called throwing an exception.

By using exceptions to manage errors, java programs have the following advantages over traditional error management technique.

1. It helps to separate the error handling code from the regular code.
2. The runtime system searches back words through the call stack, beginning with the method in which the error occurred, until it finds a method that contains an appropriate *exception* handler. A java method can “duck” any exceptions thrown within it, thereby allowing a method further up the call stack to catch it. Thus only the methods that care about errors have to worry about detecting errors.
3. Grouping or categorization of exceptions is possible.

Example 4.1

```
//Example Exception handling
class Excep1{
public static void main(String args[]){
    int i,j;
    try{
        i=0;
        j=30/i;
        System.out.println("The value of J="+j);
    }catch(ArithmeticException ex){
        System.out.println("Trying to division by zero....");
    }
}
}
}
```

You can use “Exception” in replace of “ArithmeticException” or any other exception.

Exception	Description
ArithmeticException	Arithmetic errors.
ArrayIndexOutOfBoundsException	Array Index out of bound
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid Cast
IllegalArgumentException	Illegal argument used
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalThreadStateException	Requested operation not compatible with current thread state.
IllegalStateException	Application is in incorrect state.
IndexOutOfBoundsException	Some type of index is out of bound
NegativeArraySizeException	Array Created with a negative size.
NullPointerException	Invalid use of null referance
NumberFormatException	Invalid conversion of a string to a numeric format
SecurityException	Attempt to violate security
StringIndexOutOfBoundsException	Attempt to index outside the bounds of string
UnsupportedOperationException	An unsupported operation was encountered.

Build In Exception:

Stream:

A stream is a path traveled by data in a program. To bring in information, a program opens a stream on an information source(a file, a memory, a socket) and reads information .

Similarly ,a program can send information to an external destination by opening a stream to be a destination and writing information out serially.

Using a Stream:

No matter where the information is coming from or going to and no matter what type of data is being read or written, the algorithms for reading and writing data remain the same. The steps involved are.

- Create an object of input/output stream that is associated with the data source/data destination.
- Read/write data using the object's read()/write() methods.
- Finally close the stream by calling close() method.

The *java.io* package contains a collation of stream classes for doing this. These classes are divided into two class hierarchies based on the

data type on which they operate. They are byte streams and character streams.

Predefined Streams:

There are three predefined streams already open and ready to use in every program. These streams are declared in the *java.lang.System* class, and they are all byte streams.

- a) **System.in:** This is the “standard” InputStream. This stream is already open and ready to supply input data. Typically this stream corresponds to keyboard input.
- b) **System.out :** The “standard” PrintStream. This stream is already open and ready to accept output data. Typically this stream corresponds to display output.
- c) **System.err :** The “standard” error output stream. This stream is already open and ready to accept output data. Typically this stream corresponds to display output.

Example 4.2

//Simple program to input data through keyboard.

```
import java.io.*;
```

```
public class simpInput_1 {
```

```
    public static void main(String[] args) {  
        byte name=new byte(60);  
        System.out.println("What is ur name ?") ;  
        try{  
            System.in.read(name) ;  
            System.out.write("UR name is "+name) ;  
        }catch (IOException e){  
            System.out.println("IO Error") ;  
        }  
    }  
}
```

Data Streams

Data Input Stream and Data Output Stream

If you need to work with data that is not represented as bytes or characters, you can use data input and data output streams. These streams filter an existing byte stream so that each of the following primitive types can be read or written directly from the stream : Boolean, byte, double, float , int, long and short.

A data input stream is created with the *DataInputStream(InputStream)* constructor. The argument should be existing input stream such as a buffered input stream. Conversely, a data output stream requires the *DataOutputStream(OutputStream)* constructor, which indicates the associated output stream.

Example 4.3

```
//Enter name through keyboard using DataInputStream.d
import java.io.*;
import java.lang.String;

public class simpInput_1 {

    public static void main(String[] args) {
        DataInputStream name=new DataInputStream(System.in);
        System.out.println("What is ur name ?") ;
        try{
            String name1=name.readLine() ;
            System.out.println("UR name is "+name1) ;
        }catch (IOException e){
            System.out.println("IO Error") ;
        }
    }
}
```

Some features of Stream:

length()	- count number of character of string
charAt(int)	- Returns a charter from a string at n'th position.
compareTo(Object)	- Returns 0 if equal.
compareTo(String)	- Returns 0 if equal.
substring(int,int)	- Give the starting position, ending position.
toUpperCase()	-Return to block letter of a string.
toLowerCase()	-Return to lowercase of a string.

Example: 4.4

```
String str1="Computer";  
System.out.println(str1.length() ); //output 8  
System.out.println(charAt(2)); // Output "m"  
System.out.println(substring(2,4)); //Output "mp"
```

Exercise:

- 4.1 Write a program to check a inputted string palindrome or not and display it in uppercase if inputted string is in lower case.
- 4.2 Write a program to display reverse of a string.
- 4.3 Write a program to concat two string.
- 4.4 Create a validation program for date using constructor overloading and give the proper message. The inputted date may be dd-mm-yy or dd-mmm-yy or mm-dd-yyyy.
- 4.5 Create a class stud and another class mark. The member of stud class are ROLL, NAME, PHONE . The member of mark class are ROLL, m1, m2,m3 and disp(). The method will display the total marks and grade. Now you create a program to input n numbers of student's record and short it. The short may on name/total/roll.

Session – V

Multithreading

Under time sharing operating systems, a computer system can give the impression of doing several things simultaneously by running each process for a few milliseconds, then saving its state and switching to the next process, and so on. Threads simply extend that concept from switching between several different functions executing simultaneously within a single program .

Application :

When a program download large files such as audio clips or video clips from the net, we don't want to wait until an entire clip is downloaded before starting the playback. So we can put multiple threads to work, one that downloads a clip and another that plays the clip so that these activities may proceed concurrently.

Example 5.1

```
public class MyThread implements Runnable{
    Thread t1,t2,t3;
    String s;
    public void create(){
        t1=new Thread(this,"First Thread");
        t2=new Thread(this,"Two Thread");
        t3=new Thread(this,"Thrid Thread");
    }
    public void run(){
        int sleepTime=(int)(Math.random()*5000);
        try{
            Thread.sleep(sleepTime);
        } catch(InterruptedExpection e){}
        s=Thread.currentThread().getName();
        System.out.println("My name is: "+s);
        System.out.println(" I slept for "+sleepTime+" Seconds");
    }
    public static void main(String args[]){
        MyThread mt = new MyThread();
        mt.create();
    }
}
```

Exercise:

Write program to create a digital clock using thread. The clock show system hh:mm:ss and date