Savitribai Phule Pune University

WORKBOOK

Programming in Java - CS 359

T. Y. B. Sc. (Computer Science)

SEMESTER V

(From Academic Year 2021)

Student Name:			
College:			
Roll No:		**Exam Seat No:**	
Year:		**Division:**	

Co-ordinators

- **Dr. Prashant Mulay**, Annasaheb Magar College, Hadapsar , Pune.

  Member, BOS Computer Science, Savitribai Phule Pune University

- **Dr. Manisha Bharambe** , MES Abasaheb Garware college, Pune.

  Member, BOS Computer Science, Savitribai Phule Pune University

Editor:

Dr. Manisha Bharambe , MES Abasaheb Garware college, Pune.

Member, BOS Computer Science, Savitribai Phule Pune University

Prepared by:

Ms. Gadekar Manisha J.	Annasaheb Magar College, Hadapsar, Pune.
Mrs. Kulkarni Rupali	Ahmednagar College, Ahmednagar

Table of Contents

Introduction

About the workbook

This workbook is intended to be used by T. Y. B. Sc (Computer Science) students for the Laboratory Course – II CS – 359 based on Programming in JAVA CS- 355. Semester – V.

The objectives of this book are

- Defining clearly the scope of the course
- Bringing uniformity in the way the course is conducted across different colleges
- Continuous assessment of the Students.
- Bring variation and variety in experiments carried out by different students in a batch
- Providing ready reference for students while working in the lab
- Catering to the need of slow paced as well as fast paced learners

How to use this workbook

The Object Oriented Programming using Java, practical syllabus is divided into five assignments. Each assignment has problems divided into three sets A, B and C.

- Set A is used for implementing the basic algorithms or implementing data structure along with its basic operations. **Set A is mandatory.**
- Set B is used to demonstrate small variations on the implementations carried out in set A to improve its applicability. Depending on the time availability the students should be encouraged to complete set B.
- Set C prepares the students for the viva in the subject. Students should spend additional time either at home or in the Lab and solve these problems so that they get a deeper understanding of the subject.

Instructions to the students

Please read the following instructions carefully and follow them.

- Students are expected to carry workbook during every practical.
- Students should prepare oneself before hand for the Assignment by reading the relevant material.
- Instructor will specify which problems to solve in the lab during the allotted slot and student should complete them and get verified by the instructor. However student should spend additional hours in Lab and at home to cover as many problems as possible given in this work book.
- Students will be assessed for each exercise on a scale from 0 to 5
  - Not done 0
  - Incomplete 1
  - Late Complete 2
  - Needs improvement 3
  - Complete 4
  - Well Done 5

Instruction to the Practical In-Charge

- Explain the assignment and related concepts in around ten minutes using white board if required or by demonstrating the software.
- Choose appropriate problems to be solved by students. Set A is mandatory. Choose problems from set B depending on time availability. Discuss set C with students and encourage them to solve the problems by spending additional time in lab or at home.
- Make sure that students follow the instruction as given above.
- You should evaluate each assignment carried out by a student on a scale of 5 as specified above by ticking appropriate box.
- The value should also be entered on assignment completion page of the respective Lab course.

Instructions to the Lab administrator and Exam guidelines

- You have to ensure appropriate hardware and software is made available to each student.

- Do not provide Internet facility in Computer Lab while examination

- Do not provide pen drive facility in Computer Lab while examination.

The operating system and software requirements are as given below:

- Operating system: Linux
- Editor: Any linux based editor like vi, gedit , eclipse etc.
- Compiler: javac (**Note : JAVA 8 and above version**)

Assignment Completion Sheet

Sr. No	Assignment Name	Marks (Out of 5)	Signature
1	Java Tools and IDE, Simple java programs		
2	Array of Objects and Packages		
3	Inheritance and Interfaces		
4	Exception And File Handling		
5	GUI Designing, Event Handling		
Total out of 25			
Total out of 05 (Quiz /Viva)			
Total out of 30			
Total (Out of 15)			

This is to certify that Mr/Ms _____

University Exam Seat Number _____ has successfully completed the course work for *CS 359 - Programming in Java* and has scored _____ Marks out of 15.

Instructor **Head**

Internal Examiner **External Examiner**

CS – 359

Object Oriented Programming Using Java

Assignment 1: Java Tools and IDE, Simple Java Programs

Objectives

- Introduction to the java environment
- Use of java tools like java, javac, jdb and javadoc
- Use of IDE – Eclipse (demo)
- Use of Control Statement and Iterative Statement and Array.

Reading

You should read the following topics before starting this exercise

- Creating, compiling and running a java program.
- The java virtual machine.
- Java tools like javac, java, javadoc, javap and jdb.
- Java keywords
- Syntax of class.

Ready Reference Java Tools

1. **javac:-** javac is the java compiler which compiles .java file into .class file(i.e. bytecode). If the program has syntax errors, javac reports them. If the program is error-free, the output of this command is one or more .class files.
   Syntax: javac fileName.java

2. **java:-** This command starts Java runtime environment, loads the specified .class file and executes the main method.
   Syntax: java fileName

3. **javadoc:-** javadoc is a utility for generating HTML documentation directly from comments written in Java source code.Javadoc comments have a special form but seems like an ordinary multiline comment to the compiler.
   Syntax of the comment:
   /** A sample doc comment */

   Syntax: javadoc [options] [packagenames] [sourcefiles] [@files]
   Where,
   packagenames: A series of names of packages, separated by spaces
   sourcefiles: A series of source file names, separated by spaces
   @files: One or more files that contain packagenames and sourcefiles in any order, one name per line.
   Javadoc creates the HTML documentation on the basis of the javadoc tags used in the source code files. These tags are described in the table below:

Tag	Syntax	Description
@see	@see reference	Allows you to refer to the documentation in other classes.
@author	@author authorinformation	Author-information contains author name, and / or author email address or any other appropriate information.
@version	@version versioninformation	Specifies the version of the program
@since	@since version	This tag allows you to indicate the version of this code that began using a particular feature.
@param	@param name description	This is used for method documentation. Here, name is the identifier in the method parameter list, and description is text that can describes the parameter.
@return	@return description	This describes the return type of a method.
@throws	@throws classname description	This is used when we handle Exceptions. It describes a particular type of exception that can be thrown from the method call.
@deprecated	@deprecated description	The deprecated tag suggests that this feature is no longer supported. A method that is marked @deprecated causes the compiler to issue a warning if it is used.

4. **jdb: -** jdb helps you find and fix bugs in Java language programs. This debugger has limited functionality.
   Syntax: jdb [options] [class] [arguments]
   options : Command-line options.
   class : Name of the class to begin debugging.
   arguments : Arguments passed to the main() method of class.

   After starting the debugger, the jdb commands can be executed. The important jdb commands are:
   i. help, or?: The most important jdb command, help displays the list of recognized commands with a brief description.
   ii. run: After starting jdb, and setting any necessary breakpoints, you can use this command to start the execution the debugged application.

iii. cont: Continues execution of the debugged application after a breakpoint, exception, or step.

iv. print: Displays Java objects and primitive values. For variables or fields of primitive types, the actual value is printed. For objects, a short description is printed.

Examples:

```
print MyClass.myStaticField
print myObj.myInstanceField
print i + j + k
print myObj.myMethod()//if myMethod returns non-null
```

v. dump: For primitive values, this command is identical to print. For objects, it prints the current value of each field defined in the object. Static and instance fields are included.

vi. next: The next command advances execution to the next line in the current stack frame.

vii. step: The step commands advances execution to the next line whether it is in the current stack frame or a called method.

Breakpoints can be set in jdb at line numbers, constructors, beginning of a method.
Example:

stop at MyClass:10 //sets breakpoint at instruction at line 10 of the source file containing
MyClass
stop in MyClass.display // sets breakpoint at beginning of method display in MyClass
stop in MyClass.<init> //sets breakpoint at default constructor of MyClass
stop in MyClass.<init(int)> //sets breakpoint at parameterized constructor with int as parameter

5. **javap: -** The javap tool allows you to query any class and find out its list of methods and constants.

javap [options] class

Example: javap java.lang.String It is a disassembler which allows the bytecodes of a class file to be viewed when used with a classname and the –c option.

javap -c class

Setting CLASSPATH

The classpath is the path that the Java runtime environment searches for classes and other resource files. The class path can be set using either the –classpath option or by setting the CLASSPATH environment variable.

The -classpath option is preferred because you can set it individually for each application without affecting other applications and without other applications modifying its value.

The default value of the class path is ".", meaning that only the current directory is searched. Specifying either the CLASSPATH variable or the -cp command line switch overrides this value.

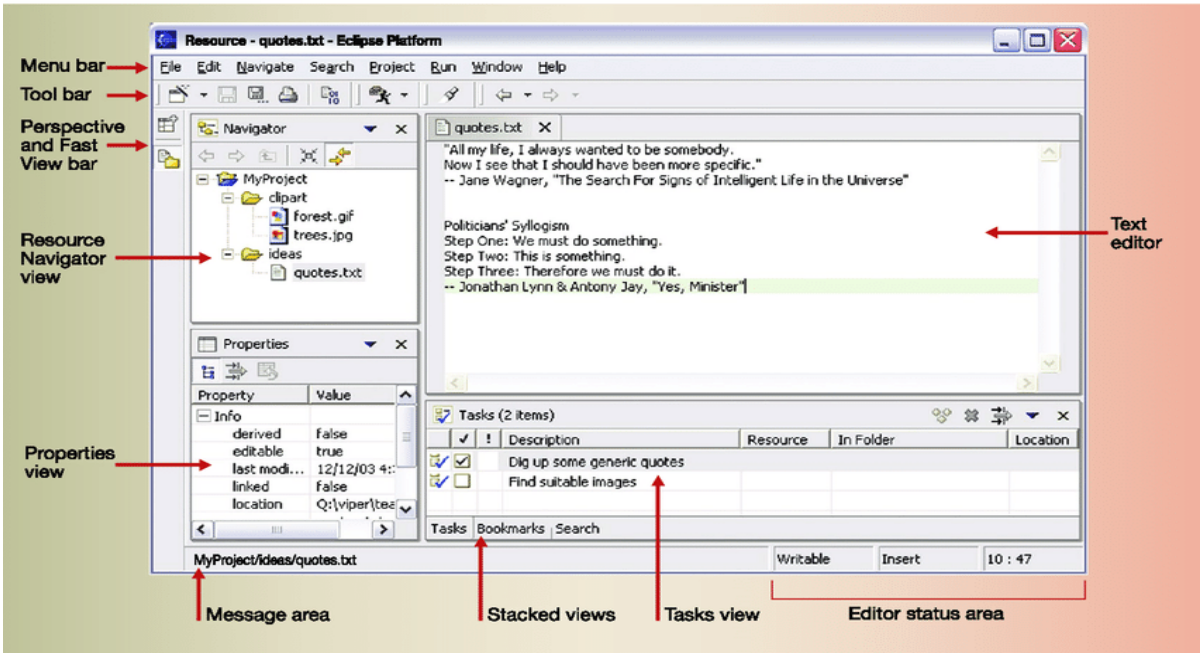
javac –classpath \myProg\myPackage; \myProg\otherclasses
Or
CLASSPATH= classpath1;classpath2...
export CLASSPATH

Example
CLASSPATH=.:/usr/local/classes.jar:/home/user1/myclasses
export CLASSPATH
To retain the existing classpath setting, use $CLASSPATH in the new list.
export CLASSPATH=$CLASSPATH:/home/user1/myclasses

About Eclipse

Eclipse is a popular IDE (Integrated Development Environment) for java programming.
It contains a base workspace and an extensible plug-in system for customizing the environment.
The latest Eclipse version 4.5 was released in 2015. The Eclipse IDE is also available as an IDE for other languages, ranging from C, C++ to Lua, Python, Perl and PHP.
It provides an editor, debugger, source control and other tools.

The GUI looks as shown:



Steps to run a java program using Eclipse:

1. Select workspace
2. Create a new project
3. Project name appears in package explorer, src folder contains source code files
4. Create class (New->Class)
5. Run your program (right click on the class and select Run As -> Java Application)

Self Activity (using IDE and editor)

Sample program 1 /* Program to generate documentation*/

```
/**   This program demonstrates javadoc */

public class MyClass
 {
        int num;
              /**    Default constructor   */
        public MyClass()
         {
            num=0;
          }

             /**
               Member function
               @param x Represents the new value of num
               @return void No return value  */

           public void assignValue(int x)
            {
                  num = x;
            }
}
```

Type the following command: javadoc MyClass.java. See the HTML documentation file MyClass.html.

Sample program 2 : Program for while loop.

```
 public class Examplewhile
{
        public static void main(String[] args)
  {
        int i =1;                  // initalization of variable
        while(i<=10)               // check the condition
     {
       System.out.println(i);
      i++;     //increment variable by 1
     }
   }
}
```

Sample Program 3 : Program do while loop.

```
public class DoWhileDemo
{
        public static void main(String[] args)
        {
            int cnt = 1;
               do
               {
                        System.out.println("Count is: " + cnt);
                        cnt++;
               } while (cnt < 11);
        }
}
```

Sample Program 4 : Program for for loop.

```
public class ForDemo
{
        public static void main(String[] args)
        {
                for(int i=1; i<=10; i++)
                {
                        System.out.println("Value of  i = : " + i +"  ");
                }
        }
}
```

Sample Program 5 : Program for switch_case .

```
public class DemoSwitch
{
    public static void main(String args[])
      {
        char ch = 'S';

           switch (ch)
           {
              case 'S':
                        System.out.println("Sunday");
                        break;
              case 'M':
                        System.out.println("Monday");
                        break;
              case 'T':
                        System.out.println("Tuesday");
                        break;
              case 'W':
                       System.out.println("Wednesday");
                       break;
```

```
                case 't':
                        System.out.println("Thursday");
                        break;
                case 'F':
                        System.out.println("Friday");
                        break;
                case 's':
                        System.out.println("Saturday");
                         break;
             }

       }
}
```

Sample Program 6 : Program for 1D Array .

```
class Demo_onedimarr
{
          public static void main(String args[])
          {
            int a[]={100,200,300,400,500};        //declaration and initialization
            System.out.println("Array elements are :\n");
            for(int i=0;i<a.length;i++)
            {
                  System.out.print(" "+a[i]);
            }
          }
}
```

Sample Program 7 : Program for Multidimentional Array .

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

int arr[][]={{11,22,33},{44,55,66},{77,88,99}};  //declaring and initializing 2D array

for(int i=0;i<3;i++)
 {
 for(int j=0;j<3;j++)
   {
   System.out.print(arr[i][j]+" ");  //display 2D array
   }
   System.out.println();
  }
 }
}
```

Sample Program 8 : Program to define a class and an object of the class

```
public class MyClass
 {
      int num;
      public MyClass()
      {
          num=0;
      }
      public MyClass(int num)
      {
         this.num = num;
      }
      public static void main(String[] args)
      {
         MyClass m1 = new MyClass();
         if(args.length > 0)
         {
             int n = Integer.parseInt(args[0]);
             MyClass m2 = new MyClass(n);
             System.out.println(m1.num);
             System.out.println(m2.num);
         }
          else
              System.out.println("Insufficient arguments");
       }
}
```

Pass one command line argument to the above program and execute it.

Lab Assignment

Set A

a) Using javap, view the methods of the following classes from the lang package: java.lang.Object , java.lang.String and java.util.Scanner. and also Compile sample program 8. Type the following command and view the bytecodes. javap -c MyClass.

b) Write a program to calculate perimeter and area of rectangle.
(hint : area = length * breadth , perimeter=2*(length+breadth))

c) Write a menu driven program to perform the following operations
   i. Calculate the volume of cylinder. (hint : Volume: $\pi \times r^2 \times h$**)**
   ii. Find the factorial of given number.
   iii. Check the number is Armstrong or not.
   iv. Exit

d) Write a program to accept the array element and display in reverse order.

Set B

a) Write a java program to display the system date and time in various formats shown below:

Current date is : 31/08/2021
Current date is : 08-31-2021
Current date is : Tuesday August 31 2021
Current date and time is : Fri August 31 15:25:59 IST 2021
Current date and time is : 31/08/21 15:25:59 PM +0530
Current time is : 15:25:59
Current week of year is : 35
Current week of month : 5
Current day of the year is : 243

Note: Use java.util.Date and java.text.SimpleDateFormat class

b) Define a class MyNumber having one private int data member. Write a default constructor to initialize it to 0 and another constructor to initialize it to a value (Use this). Write methods isNegative, isPositive, isZero, isOdd, isEven. Create an object in main. Use command line arguments to pass a value to the object (Hint : convert string argument to integer) and perform the above tests. Provide javadoc comments for all constructors and methods and generate the html help file.

c) Write a menu driven program to perform the following operations on multidimensional array ie matrix :
   i. Addition
   ii. Multiplication
   iii. Transpose of any matrix.
   iv. Exit

Set C

a) Write a program to accept n names of country and display them in descending order.

b) Write a menu driven program to perform the following operations on 2D array:
   i. Sum of diagonal elements
   ii. Sum of upper diagonal elements
   iii. Sum of lower diagonal elements
   iv. Exit

c) Write a program to display the 1 to 15 tables.

```
( 1 * 1 = 1     2 * 1 = 2     …….     15 * 1 = 15
  1 * 2 = 2     2 * 2 = 4             15 * 2 = 30
  1 * 3 = 3     2 * 3 = 6             15 * 3 = 45
  …             …                     …..
  1 * 10 = 10   2 * 10 = 20           15 * 10 = 150)
```

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Assignment 2: Array of Objects and Packages

Objectives

- Defining a class.
- Creating an array of objects.
- Creating a package.(Using package command)
- Using packages(Using import command)

Reading

You should read the following topics before starting this exercise:

- Structure of a class in java.
- Declaring class reference.
- Creating an object using new.
- Declaring an array of references.
- Creating an array of objects.
- Predefine classes-String class, String Buffer class, Wrapper class
- Syntax of the package and import command.

Ready Reference

General form of a class

```
class classname
{
   type instance-variable1;
   type instance-variable2;
   //...
   type instance-variableN;

   type methodname1(parameter-list)
   {
      //bodyofmethod
   }
   type methodname2(parameter-list)
   {
      //bodyofmethod
   }
   //...
   type methodnameN(parameter-list)
   {
      //bodyofmethod
   }
}
```

Example :

```
class Student
{
    private int rollNumber;
    private String name;
    Student()      //constructor
    {
       rollNumber=0;name=null;
    }
    Student(int rollNumber,String name)
                              //paramterised constructor
    {
        this.rollNumber=rollNumber;
        this.name=name;
    }
    void display()
    {
       System.out.println("Rollnumber="+rollNumber);
       System.out.println("Name="+name);
    }
}
```

Creating objects:

Syntax
ClassName referenceName;
referenceName = new ClassName()

OR

ClassName referenceName = new ClassName();

Example :
Student s1 = new Student();
Student s2 = new Student(10,"ABC");

Constructor:

Constructor is used to initialize the instance variables.

It is called at the creation of new object

Types of Constructors
1. Default Constructor
2. Parameterised Constructor

Example :
Student s1 = new **Student();**
Student s2 = new **Student(100,"XYZ");**

Overriding to String method of the Object class:

The toString method gives a string representation of an object. To over-ride the toString method for a user defined class,

Syntax :

```
public String toString()
{
        //return a string representation n of the object
}
```

Example

```
class Student
   {
          private int rollNumber;
          private String name;
           public String toString()
             {
                    return "RollNumber = " +rollNumber + "Name = " +name;
             }
   }
```

Declaring an array of references:

Syntax :
ClassName[ ] arrayName = new ClassName[size];

Example :
Student[ ] studentArray = new Student[10];

Creating an array of objects :

```
for each reference in the array
  {
        Create an object using new
  }
```

Example :

```
Student[] studentArray = new Student[10];

  for(i=0;i<10;i++)
        studentArray[i] = new Student();
```

Wrapper classes : The wrapper class in Java provides the mechanism to convert primitive into object and object into primitive.

Method	Purpose
Byte.parseByte	Returns byte equivalent of a String
Short.parseShort	Returns the short equivalent of a String
Integer.parseInt	Returns the int equivalent of a String
Long.parseLong	Returns the long equivalent of a String
Float.parseFloat	Returns the float equivalent of a String
Double.parseDouble	Returns the double equivalent of a String

Example : //Program to demonstrate Wrapping

```
public class WrapperExample
{
   public static void main(String args[])
   {
                        //Converting int into Integer
    int a=20;
          Integer i = Integer.valueOf(a);//converting int into Integer
          Integer j=a;   //autoboxing, now compiler will write Integer.valueOf(a)
                           internally
          System.out.println(a+" "+i+" "+j);
    }
}
```

Output:
20 20 20

Example : //Program to demonsrtrate unwrapping

```
public class WrapperExample2
{
    public static void main(String args[])
     {
                //Converting Integer to int
       Integer a=new Integer(3);
       int i = a.intValue();         //converting Integer to int
       int j = a;        //unboxing, now compiler will write a.intValue() internally
       System.out.println(a+" "+i+" "+j);
    }
}
```

Output:
3 3 3

Simple I/O

To read a String from the console, use the following code:

Syntax :

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

Or

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

For this,you will have write the following statement at the beginning:
import java.io.*;

Packages:

A package is a collection of related classes and interfaces. It provides a mechanism for compartmentalizing classes. The Java API is organized as a collection of several predefined packages. The java.lang package is the default package included in all java programs. The commonly used packages are:

Creating a package :

To create a user defined package, the package statement should be written in the source code file. This statement should be written as the first line of the program. Save the program in a directory of the same name as the package.

Syntax :

```
package packageName;
```

Accessing a package

To access classes from a package, use the import statement.

Syntax :

```
import packageName.*; //importsallclasses
import packageName.className; //importsspecifiedclass
```

Note that the package can have a hierarchy of subpackages. In that case, the pack age name should be qualified using its parentpackages.

Example :

project.sourcecode.java
Here,the package named project contains one subpackage named source code which contains a subpackage named java.

Access Rules :

The access rules for member so for class are given in the table below.

Accessible to	**public**	**protected**	**private**	**none**
Same class	Yes	Yes	Yes	Yes
Class in same package	Yes	Yes	No	Yes
Sub class (in other package)	Yes	Yes	No	No
Non sub class in other package	Yes	No	No	No

Self-Activity

Execute all Sample Programs
Sample Program 1 : To demonstrate Constructors and this keyword.

```
class MyRectangle
{
   int length;
   int breadth;
       MyRectangle()  //Default constructor
       {
         length = 0;  breadth= 0;
       }
       MyRectangle (int length, int  breadth) // parameterised constructor
       {
        this.length = length;
        this.breadth = breadth;
        }
          int area()
          {
            return length*breadth;
          }
    }
  public class RectangleTest
      {
         public static void main (String [] args)
           {
                        MyRectangle  r1 = new MyRectangle();
                        System.out.println("Area = " +r1.area());
                        MyRectangle  r2 = new MyRectangle(10,20);
                        System.out.println("Area = " +r2.area());
           }
      }
```

Sample program 2: To create objects, demonstrate use of toString and static keyword.

```
class Student
{
    int rollNumber;
    String name;
    static String classTeacher;

      Student (int r, String n)
      {
        rollNumber = r;
        name = n;
      }
      static void assignTeacher (String name)
        {
              classTeacher = name;
        }
        public StringtoString()
      {
         return "[" +  rollNumber + "," + name + "," +classTeacher + "]";
      }
     public static void main(String[]args)
      {
            Student s1 = new Student(1,"A");
            Student s2 = new Student(2,"B");
            Student.assignTeacher("ABC");
            System.out.println(s1);
            System.out.println(s2);
      }
}
```

Sample program 3 : To read Student roll number and name from the console and display them (Using BufferedReader).

```
import java.io.*;
class ConsoleInput
{
   public static void main(String[] args)
    {
          int  rollNumber;
          String name;
          BufferedReaderbr = new BufferedReader (newInputStreamReader(System.in));
          System.out.println("Entertherollnumber:");
          rollNumber = Integer.parseInt(br.readLine());
          System.out.println("Enter the name:");
               name = br.readLine();
               System.out.println("RollNumber = " +rollNumber);
               System.out.println("Name = "+name);
}
}
```

Sample program 4: To read Student roll number and name from the Console and display them (Using Scannerclass).

```
import java.util.Scanner;
class ScannerTest
 {
      public static void main(Stringargs[]) throwsException
       {
          Scanner sc = new Scanner(System.in);
          System.out.println("Enter your rollno and name:");
          int  rollno = sc.nextInt();
          String name = sc.next();
          System.out.println("Rollno :" + rollno + "Name : " +name);
         sc.close();
       }
 }
```

Sample program 5 : To demonstrate commandline arguments which accepts two integers and add them.

```
public class CLA
 {
   public static void main(String[] args)
   {
        int a,b,c;
        a = Integer.parseInt(args[0]);
        b = Integer.parseInt(args[1]);
        c = a+b;
       System.out.println("Addition:"+c);
     }
   }
```

Sample Program6 : To demonstrate usage of String methods.

```
class StrMtd
  {
      public static void main(String args[])
     {
        String s1="Hello Java";
        String s2 = "Hello";
        String s3 = "BYEBYE";
        System.out.println(s1.trim());
        System.out.println(s1.substring(3,8));
        System.out.println(s2.concat("world"));
        System.out.println(s2.replace('e','k'));
        System.out.println(s3.toLowerCase());
      }
 }
```

7. Sample program to create and use packages.

```
//A.java
package P1;
public class A
 {
    public void display()
    {
           System.out.println("IndisplayofA");
     }
}

//B.java
package P1.P2;  //P2 is a subpackage of P1
public class B
 {
    public void display()
     {
        System.out.println("IndisplayofB");
     }
}

//PackageTest.java
import P1.*;
import P1.P2.*;
class PackageTest
 {
public static void main (String args[])
{
            A obj1 = new A();
            obj1.display();
            B obj2 = new B();

Obj2.display();
}
}
```

Create folder P1.Save A.java in folder P1.Create folder P2 in side P1.Save B.java in P2.

Lab Assignments

Set A

a) Create an employee class(id,name,deptname,salary). Define a default and parameterized constructor. Use 'this' keyword to initialize instance variables. Keep a count of objects created. Create objects using parameterized constructor and display the object count after each object is created.(Use static member and method). Also display the contents of each object.

b) Define Student class(roll_no, name, percentage) to create n objects of the

Student class. Accept details from the user for each object. Define a static method "sortStudent" which sorts the array on the basis of percentage.

c) Write a java program to accept 5 numbers using command line arguments sort and display them.

d) Write a java program that take input as a person name in the format of first, middle and last name and then print it in the form last, first and middle name, where in the middle name first character is capital letter.

Set B

a) Write a Java program to create a Package "SY" which has a class SYMarks (members – ComputerTotal, MathsTotal, and ElectronicsTotal). Create another package TY which has a class TYMarks (members – Theory, Practicals). Create n objects of Student class (having rollNumber, name, SYMarks and TYMarks). Add the marks of SY and TY computer subjects and calculate the Grade ('A' for >= 70, 'B' for >= 60 'C' for >= 50 , Pass Class for > =40 else 'FAIL') and display the result of the student in proper format.

b) Define a class CricketPlayer (name,no_of_innings,no_of_times_notout, totatruns, bat_avg). Create an array of n player objects .Calculate the batting average for each player using static method avg(). Define a static sort method which sorts the array on the basis of average. Display the player details in sorted order.

Set C

a) Write a package for String operation which has two classes Con and Comp. Con class has to concatenates two strings and comp class compares two strings. Also display proper message on execution.

b) Create four member variables for Customer class. Assign public, private, protected and default access modifiers respectively to these variables. Try to access these variables from other classes (Same package and Different package)

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Assignment 3: Inheritance and Interfaces

Objectives

- To improve inheritance in java.
- To define abstract class.
- To define and use interfaces and functional interfaces

Reading

You should read the following topics before starting this exercise:

- Concept of inheritance.
- Use of extends keyword.
- Concept of abstract class.
- Defining an interface.
- Use of implements keyword.

Ready Reference

Inheriting a class :

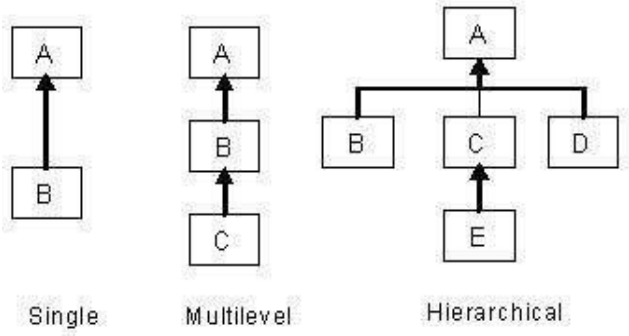
The **syntax** to create sub class is:

```
class SubClassName extends SuperClassName
{
     //class body
}
```

Example :

```
class Manager extends Employee
{
    //code
}
```

Types of Inheritance



Access in subclass

The following member scan be accessed in a subclass:

i. public or protected superclass members.

ii. Members with no specifier if subclass is in same package.

The"super"keyword

It is used for three purposes:

i. Invoking superclass constructor-super(arguments)
ii. Accessing superclass members–super.member
iii. Invoking superclass methods–super.method(arguments)

Example:

```
classA
{
        protected int num;
        A(int num)
        {
          this.num=num;
        }
}
class B extends A
{
      int num;
      B(int a, int b)
       {
            super(a);          //should be the first line in the subclass constructor
            this.num=b;
      }
     void display()
     {
         System.out.println("In A, num = "+super.num);
         System.out.println("In B, num = "+num);
    }
}
```

Overriding methods

Redefining superclass methods in a subclass is called overriding. The signature of the subclass method should be the same as the superclass method.

Syntax

```
class A
{
   void method1(int num)
    {
          //code
     }
 }
 class B extends A
  {
       void method1(int x)
       {
            //code
      }
}
```

Dynamic binding

When over-riding is used ,the method call is resolved during run-time i.e. depending on the object type, the corresponding method will be invoked.

Example :

```
A ref;
ref = new A();
ref.method1(10);       //calls method of class
A ref = new B();
ref.method1(20);      //calls method of classB
```

Abstract class

An abstract class is a class which cannot be instantiated. It is only used to create subclasses. A class which has abstract methods must be declared abstract.
An abstract class can have data members, constructors, method definitions and method declarations.

Syntax :

```
abstract class ClassName
{
        ...
}
```

Abstract method

An abstract method is a method which has no definition. The definition is provided by the subclass.

Syntax :

```
abstract returnType method(arguments);
```

Interface

An interface is a pure abstract class i.e. it has only abstract methods and final variables.
An interface can be implemented by multiple classes.

Syntax :

```
interface InterfaceName
{
        //abstract methods
        //final variables
}
```

Example:

```
interface MyInterface
{
    void method1();
    void method2();
     int size=10;       //finalandstatic
}
class MyClass implements MyInterface
{
//definemethod1andmethod2
}
```

Marker Interface :
An interface that does not contain methods, fields, and constants is known as marker interface. In other words, an empty interface is known as marker interface or tag interface. It delivers the run-time type information about an object. It is the reason that the JVM and compiler have additional information about an object.
The Serializable and Cloneable interfaces are the example of marker interface. In short, it indicates a signal or command to the JVM. The declaration of marker interface is the same as interface in Java but the interface must be empty.
Example:

```
public interface Serializable
{
    …..
}
```

Functional Interfaces
A functional interface in Java is an interface that contains only a single abstract (unimplemented) method. A functional interface can contain default and static methods which do have an implementation, in addition to the single unimplemented method. Functional Interface is also known as Single Abstract Method Interfaces or SAM Interfaces. It is a new feature in Java, which helps to achieve functional programming approach.
Syntax :

```
public interface MyFunctionalInterface()
{
      // abstract method
       public void functionalMethod();
 }
```

Example :
The below interface is a functional interface in Java, since it only contains a single non-implemented method execute()

```
public interface MyFunctionalInterface2
  {
   public void execute();

   public default void print(String text)
   {
     System.out.println(text);
   }

   public static void print(String text, PrintWriter writer) throws IOException
   {
        writer.write(text);
   }
}
```

Functional Interfaces Can Be Implemented by a Lambda Expression

lambda Expression
A Java lambda expression is thus a function which can be created without belonging to any class. Java lambda expressions are commonly used to implement simple event listeners / callbacks, or in functional programming with the Java Streams API.

Syntax :
lambda operator -> body

where lambda operator can be :
Zero parameter :
() -> System.out.println("Zero parameter lambda");
One parameter :
(t) -> System.out.println("One parameter: " + t);

Multiple parameters :
(t1, t2) -> System.out.println("Multiple parameters: "
+ t1 + ", " + t2);

OR

```
MyFunctionalInterfacefunctionalInterface = () ->
{
      // basic functionality logic goes here
}
```

Example :

```
MyFunctionalInterface lambda = () ->
{
      System.out.println("Executing...");
}
```

A Java lambda expression implements a single method from a Java interface. In order to know what method the lambda expression implements, the interface can only contain a single unimplemented method i.e. functional interface.

Self-Activity
Execute all the sample programs
Sample program 1 : To demonstrate Inheritance concept.

```
class Parent
{
   public void p1()
   {
       System.out.println("Parent method");
   }
}
public class Child extends Parent
 {
```

```
    public void c1()
  {
      System.out.println("Child method");
  }
  public static void main(String[] args)
  {
    Child cobj = new Child();
    cobj.c1();  //method of Child class
    cobj.p1();  //method of Parent class
  }
}
```

Sample program 2 : To demonstrate Abstract class.

```
abstract class Bank
{
    abstract int getRateOfInterest();
}
class SBI extends Bank
{
    int getRateOfInterest()
    {
      return 7;
    }
}
class PNB extends Bank
{
    int getRateOfInterest()
    {
      return 8;
    }
}

class TestBank
{
  public static void main(String args[])
  {
    Bank b;
    b=new SBI();
    System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
    b=new PNB();
    System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
  }
}
```

Sample Program 3 : To illustrate Interfaces.

```
interface Result
 {
   final static float pi-3.14f;
   float compute(float x, float y);
 }
```

```
class Addition implements Result
 {
   public float compute(float x, float y)
    {
         return (x+y);
     }
   }
class CircleArea implements Result
 {
   public float compute(float x, float y)
    {
         return (pi*x*x);
     }
   }
class InterfaceDemo
{
  public static void main(String args[])
   {
     Addition a = new Addition();
     CircleArea c = new CircleArea();
     Result res;
     res = a;
     System.out.println("Result of Addition ="+res.compute(15.2f,10));
     res = c;
    System.out.println("Result of CircleArea="+res.compute(10,2));
 }
}
```

Sample program 4 : To demonstrate inheritance and interfaces.

```
interface Shape
  {
    double area();
   }
  class Circle implements Shape
  {
     double radius;
     Circle(double radius)
     {
        this.radius=radius;
     }
     public double area()
     {
       return  java.util.Math.PI*radius*radius;
     }
    class CylinderextendsCircle
    {
      double height;
     Cylinder(double radius ,double height)
    {
```

```
        super(radius);
       this.height=height;
    }
   public double area()     //overriding
   {
        return  java.util.Math.PI*radius*radius*height;
    }
}
public class Test
{
  public static void main (String []args)
   {
    Shape s;
    s = new Circle(5.2);
    System.out.println("Area of circle ="+s.area());
    s = new Cylinder(5,2.5);
    System.out.println("Area of cylinder="+s.area());
   }
}
```

Sample program 5 : program to display of square of given number using Function Interface. (lambda expression)

```
@FunctionalInterface
interface PrintNumber
{
        public void print(int num1);
}

public class PrintSquare
 {
    public static void main(String[] a)
         {
                PrintNumber p = n -> System.out.println("Square is: "+ n*n);
                p.print(25);
        }
}
```

NOTE : (JAVA 8 or above version)

Lab Assignments

Set A

a) Write a program for multilevel inheritance such that country is inherited from continent. State is inherited from country. Display the place, state, country and continent.

b) Define an abstract class Staff with protected members id and name. Define a parameterized constructor. Define one subclass OfficeStaff with member department. Create n objects of OfficeStaff and display all details.

c) Define an interface "Operation" which has methods area(),volume().Define a constant PI having a value 3.142.Create a class cylinder which implements this interface (members – radius, height) Create one object and calculate the area and volume.

d) Write a program to find the cube of given number using function interface.

Set B

a) Create an abstract class "order" having members id,description.Create two subclasses "Purchase Order" and "Sales Order" having members customer name and Vendor name respectively.Define methods accept and display in all cases. Create 3 objects each of Purchase Order and Sales Order and accept and display details.

b) Write a program to using marker interface create a class product(product_id, product_name, product_cost, product_quantity) define a default and parameterized constructor. Create objects of class product and display the contents of each object and Also display the object count.

Set C

a) Create an interface Department containing attributes deptName and deptHead. It also has abstract methods for printing the attributes. Create a class hostel containing hostelName, hostelLocation and numberOfRooms. The class contains method printing the attributes. Then write Student class extending the Hostel class and implementing the Department interface. This class contains attributes studentName, regNo, electiveSubject and avgMarks. Write suitable printData method for this class. Also, implement the abstract methods of the Department interface. Write a driver class to test the Student class. The program should be menu driven containing the options:

   i. Admit new student
   ii. Migrate a student
   iii. Display details of a student

   For the third option, a search is to be made on the basis of the entered registration Number.

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Assignment 4: Exception Handling and File Handling

Objectives

- Demonstrate exception handling mechanism in java
- Defining user defined exception classes.
- Performing Input/Output operations using console and files.

Reading

You should read the following topics before starting this exercise:

- Concept of Exception and Exception class hierarchy.
- Use of try, catch, throw, throws and finally keywords
- Defining user defined exception classes
- Concept of streams and Types of streams
- Byte and Character stream classes and File class .

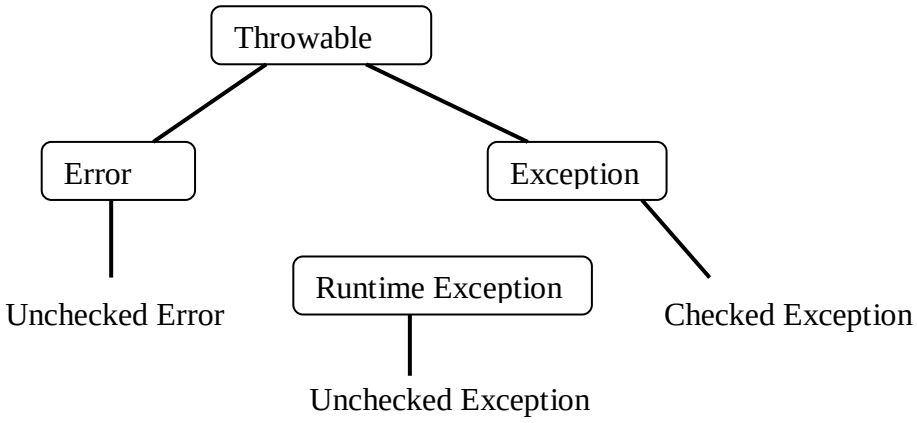
Ready Reference

Exception : An exception is an abnormal condition that arises in a code at run time.
When an exception occurs :

1. An object representing that exception is created.
2. The method may handle the exception itself.
3. If the method cannot handle the exception, it "throws" this exception object to the method which called it.
4. The exception is "caught" and processed by some method or finally by the default java exception handler.

Predefined Exception classes

Java provides a hierarchy of Exception classes which represent an exception type



Exception Handling keywords

Exception handling in java is managed using 5 keywords :

try , catch , throw, throws, finally

Syntax :

```
try
 {
     // code that may cause an exception
  }
  catch (ExceptionType1 object)
 {
     // handle the exception
 }
  catch (ExceptionType2 object)
 {
    // handle the exception
 }
 finally
{
    // this code is always executed
 }
```

Example:

```
try
   {
       int a = Integer.parseInt(args[0]);
         ...
    }
    catch(NumberFormatException e)
    {
      System.out.println(" Exception Caught" ) ;
    }
```

Note: try-catch blocks can be nested.

throw keyword :

The throw keyword is used to throw an exception object or to rethrow an exception.

```
throw exceptionObject;
```

Example:

```
catch(NumberFormatException e)
{
   System.out.println("Caught and rethrown" ) ;
   throw e;
 }
```

We can explicitly create an exception object and throw it.

For Example:

```
throw new NumberFormatException();
```

throws keyword:

If the method cannot handle the exception, it must declare a list of exceptions it may cause.

This list is specified using the throws keyword in the method header.
All checked exceptions muct be caught or declared.

Syntax:

```
returnType methodName(arguments) throws ExceptionType1
      [,ExceptionType2...]
 {
     //method body
  }
```

Example:

```
void acceptData() throws IOException
{
   //code
}
```

Exception Types: There are two types of Exceptions,

1. Checked exceptions
2. Unchecked exceptions

Checked exceptions must be caught or rethrown. Unchecked exceptions do not have to be caught.

Unchecked Exceptions:

Exception	**Meaning**
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid cast.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalStateException	Environment or application is in incorrect state.
IllegalThreadStateException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
SecurityException	Attempt to violate security.
StringIndexOutOfBounds	Attempt to index outside the bounds of a string.
UnsupportedOperationException	An unsupported operation was encountered.

Checked Exceptions:

Exception	**Meaning**
ClassNotFoundException	Class not found.
CloneNotSupportedException	Attempt to clone an object that does not implement the Cloneable

	interface.
IllegalAccessException	Access to a class is denied.
InstantiationException	Attempt to create an object of an abstract class or interface.
InterruptedException	One thread has been interrupted by another thread.
NoSuchFieldException	A requested field does not exist.
NoSuchMethodException	A requested method does not exist.

User Defined Exceptions:

A user defined exception class can be created by extending the Exception class.

```
class  UserDefinedException  extends  Exception
 {
   //code
 }
```

When that exception situation occurs, an object of this exception class can be created and thrown.

For example, if we are accepting an integer whose valid values are only positive, then we can throw an "InvalidNumberException" for any negative value entered.

```
class NegativeNumberException extends Exception
 {
      NegativeNumberException(int n)
       {
            System.out.println("Negative input " + n);
       }
 }
...
 public static void main(String[] args)
  {
       int num = Integer.parseInt(args[0]);
       if( num < 0)
            throw new NegativeNumberException(num);
       else
            //process num
  }
```

java.io.File class

This class supports a platform-independent definition of file and directory names. It also provides methods to list the files in a directory, to check the existence, readability, writeability, type, size, and modification time of files and directories, to make new directories, to rename files and directories, and to delete files and directories.

Constructors :

```
public File(String path);
public File(String path, String name);
public File(File dir, String name);
```

Example :

File f1=new File("/home/java/a.txt");

Methods

- boolean canRead()- Returns True if the file is readable.
- boolean canWrite()- Returns True if the file is writeable.
- String getName()- Returns the name of the File with any directory names omitted.
- boolean exists()- Returns true if file exists
- String getAbsolutePath()- Returns the complete filename.

Otherwise, if the File is a relative file specification, it returns the relative filename appended to the current working directory.

- String getParent() - Returns the directory of the File. If the File is an absolute specification.
- String getPath() - Returns the full name of the file, including the directory name.
- boolean isDirectory() - Returns true if File Object is a directory
- boolean isFile() - Returns true if File Object is a file
- long lastModified() - Returns the modification time of the file (which should be used for comparison with other file times only, and not interpreted as any particular time format).
- long length() - Returns the length of the file.
- boolean delete() - deletes a file or directory. Returns true after successful deletion of a file.
- boolean mkdir () - Creates a directory.
- boolean renameTo(File dest) - Renames a file or directory. Returns true after successful renaming

Example :- Checking file existence

```
import java.io.File;
class FileTest
{
    public static void main(String args[ ])
    {
        File f1=new File ("data.txt");
        if (f1.exists())
            System.out.println ("File Exists");
        else
            System.out.println ("File Does Not Exists");
    }
}
```

Directories

A directory is a File that contains a list of other files & directories. When you create a File object and it is a directory, the isDirectory() method will return true. In this case list method can be used to extract the list of other files & directories inside.

The forms of list() method is

public String[ ] list()
public String[ ] list(FilenameFilter filter)

Example : Using a list( )

```
String dirname = "/javaprg/demo";
File f1 = new File (dirname);
if (f1.isDirectory())
 {
      String s[] = f1.list();
      for (int i=0; i<s.length; i++)
       {
           File f = new File (dirname+"/"+s[i]);
            if (f.isDirectory())
                System.out.println(s[i]+ " is a directory");
           else
                System.out.println(s[i]+ " is a File");
       }
 }
```

Streams

A stream is a sequence of bytes. When writing data to a stream, the stream is called an output stream. When reading data from a stream, the stream is called an input stream. If a stream has a buffer in memory, it is a buffered stream. Binary Streams contain binary data. Character Streams have character data and are used for storing and retrieving text.

The two main types of Streams are ByteStream and CharacterStream.

There are four top level abstract stream classes: InputStream, OutputStream, Reader, and Writer.

1. InputStream. A stream to read binary data.
2. OutputStream. A stream to write binary data.
3. Reader. A stream to read characters.
4. Writer. A stream to write characters.

ByteStream Classes

a. InputStream Methods-

1. int read () - Returns an integer representation of next available byte of input.-1 is returned at the stream end.
2. int read (byte buffer[ ]) - Read up to buffer.length bytes into buffer & returns actual number of bytes that are read. At the end returns –1.

3. int read(byte buffer[ ], int offset, int numbytes) - Attempts to read up to numbytes bytes into buffer starting at buffer[offset]. Returns actual number of bytes that are read. At the end returns –1.
4. void close() - to close the input stream
5. void mark(int numbytes) - places a mark at current point in input stream & remain valid till number of bytes are read.
6. void reset() - Resets pointer to previously set mark/ goes back to stream beginning.
7. long skip(long numbytes) - skips number of bytes.
8. int available() - Returns number of bytes currently available for reading.

b. OutputStream Methods-

1. void close() - to close the OutputStream
2. void write (int b) - Writes a single byte to an output stream.
3. void write(byte buffer[ ]) - Writes a complete array of bytes to an output stream.
4. void write (byte buffer[ ], int offset, int numbytes) - Writes a sub range of numbytes bytes from the array buffer, beginning at buffer[offset].
5. void flush() - clears the buffer.

The following table lists the Byte Stream classes

Stream Class	**Meaning**
BufferedInputStream	Buffered input stream.
BufferedOutputStream	Buffered output stream.
ByteArrayInputStream	Input stream that reads from a byte array.
ByteArrayOutputStream	Output stream that writes to a byte array.
DataInputStream	An input stream that contains methods for reading the Java standard data types.
DataOutputStream	An output stream that contains methods for writing the Java standard data types.
FileInputStream	Input stream that reads from a file.
FileOutputStream	Output stream that writes to a file.
FilterInputStream	Implements InputStream.
FilterOutputStream	Implements OutputStream.
InputStream	Abstract class that describes stream input.
OutputStream	Abstract class that describes stream output.
PipedInputStream	Input pipe.
PipedOutputStream	Output pipe.
PrintStream	Output stream that contains print( ) and println( ).
PushbackInputStream	Input stream that supports one-byte “unget,” which returns a byte to the input stream.
RandomAccessFile	Supports random access file I/O.
SequenceInputStream	Input stream that is a combination of two or more input streams that will be read.

CharacterStream Classes

Reader : Reader is an abstract class that defines Java’s method of streaming character input. All methods in this class will throw an IOException.

Methods in this class-

1. int read () - Returns an integer representation of next available character from invoking stream. -1 is returned at the stream end.
2. int read (char buffer[ ]) - Read up to buffer.length chacters to buffer & returns actual number of characters that are successfully read. At the end returns –1.
3. int read(char buffer[ ], int offset, int numchars) - Attempts to read up to numchars into buffer starting at buffer[offset]. Returns actual number of characters that are read. At the end returns –1.
4. void close() - to close the input stream.
5. void mark(int numchars) - places a mark at current point in input stream & remain valid till number of characters are read.
6. void reset() - Resets pointer to previously set mark/ goes back to stream beginning.
7. long skip(long numchars) - skips number of characters.
8. int available() - Returns number of bytes currently available for reading.

Writer : Is an abstract class that defines streaming character output. All the methods in this class returns a void value & throws an IOException.

Methods in this class-

1. void close() - to close the OutputStream
2. void write (int ch) - Writes a single character to an output stream.
3. void write(char buffer[ ]) - Writes a complete array of characters to an output stream.
4. void write (char buffer[ ], int offset, int numchars) - Writes a sub range of numchars from the array buffer, beginning at buffer[offset].
5. void write(String str) - Writes str to output stream.
6. void write(String str, int offset, int numchars) - Writes a subrange of numchars from string beginning at offset.
7. void flush() - clears the buffer.

The following table lists the Character Stream classes

Stream Class	**Meaning**
BufferedReader	Buffered input character stream.
BufferedWriter	Buffered output character stream.
CharArrayReader	Input stream that reads from a character array.
CharArrayWriter	Output stream that writes to a character array.
FileReader	Input stream that reads from a file.
FileWriter	Output stream that writes to a file.
FilterReader	Filtered reader.
FilterWriter	Filtered writer.
InputStreamReader	Input stream that translates bytes to characters.
LineNumberReader	Input stream that counts lines.
OutputStreamWriter	Output stream that translates characters to bytes.
PipedReader Input	pipe PipedWriter Output pipe.
PrintWriter Output	stream that contains print( ) and println( ).
PushbackReader	Input stream that allows characters to be returned to the input stream.
Reader	Abstract class that describes character stream input.
StringReader	Input stream that reads from a string.
StringWriter	Output stream that writes to a string.
Writer	Abstract class that describes character stream output.

Random Access File

Random access files permit nonsequential, or random, access to a file's contents. To access a file randomly, you open the file, seek a particular location, and read from or write to that file. When opening a file using a RandomAccessFile, you can choose whether to open it read-only or read write.

RandomAccessFile (File file, String mode) throws FileNotFoundException

RandomAccessFile (String filePath, String mode) throws FileNotFoundExcep tion

The value of mode can be one of these:

"r" Open for reading only.
"rw" Open for reading and writing.

Methods :

1. position – Returns the current position
2. position(long) – Sets the position
3. read(ByteBuffer) – Reads bytes into the buffer from the stream
4. write(ByteBffer) – Writes bytes from the buffer to the stream
5. truncate(long) – Truncates the file (or other entity) connected to the stream

Example:

```
File f = new File("data.dat");          //Open the file for both reading and writing
RandomAccessFile rand = new RandomAccessFile(f,"rw");
rand.seek(f.length());                  //Seek to end of file
rand.writeBytes("Append this line at the end");  //Write end of file
rand.close();
System.out.println("Write-Successful");
```

Self Activity

Sample program 1 : To demonstrate exceptions.

```
class NegativeNumberException extends Exception
{
     NegativeNumberException(int n)
      {
          System.out.println("Negative input : " + n);
       }
 }
public class ExceptionTest
{
   public static void main( String args[] )
    {
       int num, i, sum=0;
       try
      {
          num = Integer.parseInt(args[0]);
          if(num < 0)
               throw new NegativeNumberException(num);
```

```
        for(i=0; i<num; i++)
              sum = sum+i;
       }
      catch(NumberFormatException e)
      {
          System.out.println("Invalid format");
       }
      catch(NegativeNumberException e)
      {
       }
       finally
       {
           System.out.println("The sum is : "+sum);
        }
      } // end main
  } // end class
```

Note : Compile and run the program for different inputs like abc, -3 and 10

Sample program 2: /* Program to count occurrences of a string within a text file*/

```
import java.io.*;
import java.util.*;
public class TextFileReadApp
{
        public static void main (String arg[])
        {
             File f = null;     // Get the file from the argument line.
              if (arg.length > 0)
              f = new File (arg[0]);
              if (f == null || !fe.exists ())
              {
                   System.exit(0);
              }
               String string_to_find = arg[1];
               int num_lines = 0;
               try
               {
                  FileReader file_reader = new FileReader (f);
                  BufferedReader buf_reader = new BufferedReader (file_reader);
                                                    // Read each line and search string
                  do
                  {
                     String line = buf_reader.readLine ();
                      if (line == null)
                          break;
                      if (line.indexOf(string_to_find) != -1)
                          num_lines++;
                    } while (true);
                   buf_reader.close ();
                 }
                 catch (IOException e)
```

```
                {
                    System.out.println ("IO exception =" + e );

                  }
                 System.out.println ("No of lines containing " +
                                        string_to_find + " = " + num_lines);
                } // main
    } //class TextFileReadApp
```

Sample program 3 :/* Program to write and read primitive types to a file */

```
import java.io.*;
class PrimitiveTypes
 {
    public static void main(String args[]) throws IOException
     {
          FileOutputStream fos = new FileOutputStream("info.dat");
          DataOutputStream dos = new DataOutputStream(fos);
          dos.writeInt(25);
          dos.writeBoolean(true);
          dos.writeChar('A');
          dos.writeDouble(5.45);
          fos.close();
           FileInputStream fis = new FileInputStream("info.dat") ;
           DataInputStream dis = new DataInputStream(fis);
           int num = dis.readInt();
           boolean b = dis.readBoolean();
           char ch = dis.readChar();
           double dbl = dis.readDouble();
           System.out.println("Int- "+num +"\nBoolean- "+b);
           System.out.println("\nCharacter- "+ch+"\nDouble- "+dbl);
           fis.close();
       } //main
} //class
```

Sample program 4 : /* Program to read integers from a file using Scanner class*/

```
import java.io.*;
import java.util.*;
class ReadIntegers
 {
    public static void main(String args[]) throws IOException
     {
        FileReader file = new FileReader("numbers.txt");
        Scanner sc = new Scanner(file);
        int sum=0, num;
        while(sc.hasNext())
        {
            num = sc.nextInt();
            System.out.println("Number = "+ num);
            sum = sum+num;
        }
      System.out.println("The sum = "+ sum);     file.close();
     }//main
```

```
} //class
```

Lab Assignments

Set A

a) Define a class patient (patient_name, patient_age, patient_oxy_level,patient_HRCT_report). Create an object of patient. Handle appropriate exception while patient oxygen level less than 95% and HRCT scan report greater than 10, then throw user defined Exception "Patient is Covid Positive(+) and Need to Hospitalized" otherwise display its information.

b) Write a program to read a text file "sample.txt" and display the contents of a file in reverse order and also original contents change the case (display in upper case).

c) Accept the names of two files and copy the contents of the first to the second. First file having Book name and Author name in file. Second file having the contents of First file and also add the comment 'end of file' at the end.

Set B

a) Write a program to read book information (bookid, bookname, bookprice, bookqty) in file "book.dat". Write a menu driven program to perform the following operations using Random access file:
   i. Search for a specific book by name.
   ii. Display all book and total cost

b) Define class EmailId with members ,username and password. Define default and parameterized constructors. Accept values from the command line Throw user defined exceptions – "InvalidUsernameException" or "InvalidPasswordException" if the username and password are invalid.

c) Define a class MyDate (day, month, year) with methods to accept and display a MyDate object. Accept date as dd, mm, yyyy. Throw user defined exception "InvalidDateException" if the date is invalid.
   Examples of invalid dates : 03 15 2019, 31 6 2000, 29 2 2021

Set C

a) Write a menu driven program to perform the following operations on a set of integers as shown in the following figure. A load operation should generate 10 random integers (2 digit) and display the number on screen. The save operation should save the number to a file "number.txt". The short menu provides various operations and the result is displayed on the screen.

b) Write a java program to accept Employee name from the user and check whether it is valid or not. If it is not valid then throw user defined Exception "Name is Invalid" otherwise display it.(Name should contain only characters)

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Assignment 5: GUI Designing, Event Handling

Objectives

- To demonstrate GUI creation using Swing package and Layout managers.
- Understand the Event Handling mechanism in java.
- Using Event classes, Event Listeners and Adapters.
- Create java application using AWT and Swing.

Reading

You should read the following topics before starting this exercise

- AWT and Swing concepts.
- Layout managers in java
- Containers and Components
- Adding components to containers
- Event sources, listeners and delegation event model
- Adapter classes

Ready Reference

Graphical User Interface elements are implemented in two java packages – AWT and Swing. Swing is the newer package and swing classes are based on AWT classes.

Swing Architecture:
The design of the Swing component classes is based on the Model-View-Controller architecture, or MVC.

1. The model stores the data.
2. The view creates the visual representation from the data in the model.
3. The controller deals with user interaction and modifies the model and/or the view.

Swing Classes:
The following table lists some important Swing classes and their description.

Class	Description
Box	Container that uses a BoxLayout.
JApplet	Base class for Swing applets.
JButton	Selectable component that supports text/image display.
JCheckBox	Selectable component that displays state to user.
JCheckBoxMenuItem	Selectable component for a menu; displays state to user.
JColorChooser	For selecting colors.
JComboBox	For selecting from a drop-down list of choices.
JComponent	Base class for Swing components.
JDesktopPane	Container for internal frames.
JDialog	Base class for pop-up subwindows.
JEditorPane	For editing and display of formatted content.
JFileChooser	For selecting files and directories.

JFormattedTextField	For editing and display of a single line of formatted text.
JFrame	Base class for top-level windows.
JInternalFrame	Base class for top-level internal windows.
JLabel	For displaying text/images.
JLayeredPane	Container that supports overlapping components.
JList	For selecting from a scrollable list of choices.
JMenu	Selectable component for holding menu items; supports text/image display.
JMenuBar	For holding menus.
JMenuItem	Selectable component that supports text/image display.
JOptionPane	For creating pop-up messages.
JPanel	Basic component container.
JPasswordField	For editing and display of a password.
JPopupMenu	For holding menu items and popping up over components.
JProgressBar	For showing the progress of an operation to the user.
JRadioButton	Selectable component that displays state to user; included in ButtonGroup to ensure that only one button is selected.
JRadioButtonMenuItem	Selectable component for menus; displays state to user; included in ButtonGroup to ensure that only one button is selected.
JRootPane	Inner container used by JFrame, JApplet, and others.
JScrollBar	For control of a scrollable area.
JScrollPane	To provide scrolling support to another component.
JSeparator	For placing a separator line on a menu or toolbar.
JSlider	For selection from a numeric range of values.
JSpinner	For selection from a set of values, from a list, a numeric range, or a date range.
JSplitPane	Container allowing the user to select the amount of space for each of two components.
JTabbedPane	Container allowing for multiple other containers to be displayed; each container appears on a tab.
JTable	For display of tabular data.
JTextArea	For editing and display of single-attributed textual content.
JTextField	For editing and display of single-attributed textual content on a single line.
JTextPane	For editing and display of multi-attributed textual content.
JToggleButton	Selectable component that supports text/image display; selection triggers component to stay “in”.
JToolBar	Draggable container.
JToolTip	Internally used for displaying tool tips above components.
JTree	For display of hierarchical data.
JViewport	Container for holding a component too big for its display area.
JWindow	Base class for pop-up windows.

Layout Manager

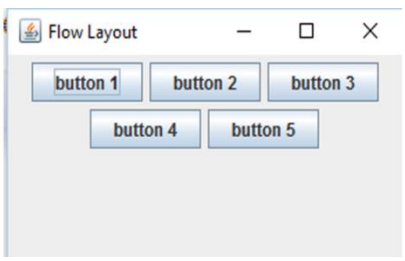
The job of a layout manager is to arrange components on a container. Each container has a layout manager associated with it. To change the layout manager for a container, use the setLayout() method.

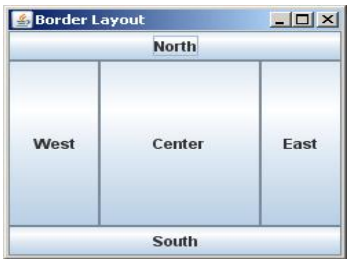
Syntax :

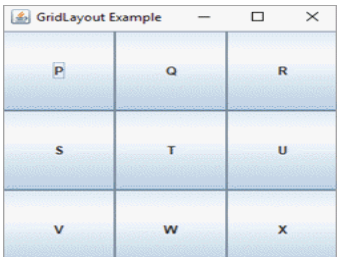
setLayout(LayoutManager obj)

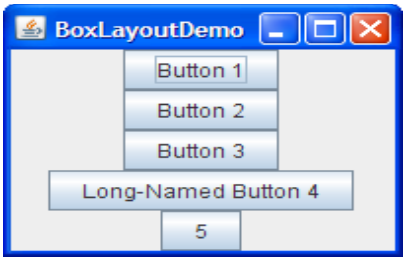
The predefined managers are listed below:

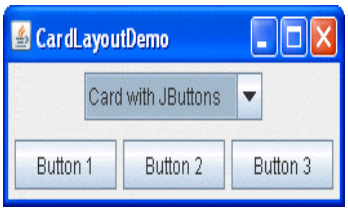
1. FlowLayout	2.BorderLayout	3.GridLayout
4. BoxLayout	5.CardLayout	6.GridBagLayout

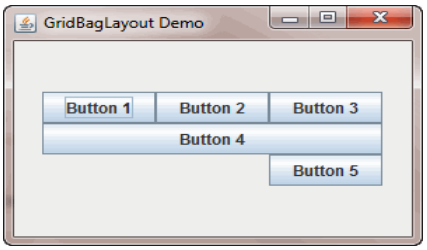












Examples:

```
JPanel p1 = new JPanel();
p1.setLayout(new FlowLayout());
p1.setLayout(new BorderLayout());
p1.setLayout(new GridLayout(3,4));
```

Important Containers:

1.JFrame –
This is a top-level container which can hold components and containers like panels.

Constructors

JFrame()
JFrame(String title)

Important Methods

setSize(int width, int height) - Specifies size of the frame in pixels.
setLocation(int x, int y) - Specifies upper left corner
setVisible(boolean visible) -Set true to display the frame
setTitle(String title) - Sets the frame title
setDefaultCloseOperation(int mode) -Specifies the operation when frame is closed.

The modes are:

JFrame.EXIT_ON_CLOSE
JFrame.DO_NOTHING_ON_CLOSE
JFrame.HIDE_ON_CLOSE
JFrame.DISPOSE_ON_CLOSE

pack() -Sets frame size to minimum size required to hold components.

2.JPanel -

This is a middle-level container which can hold components and can be added to other containers like frame and panels.

Constructors

public javax.swing.JPanel(java.awt.LayoutManager, boolean);
public javax.swing.JPanel(java.awt.LayoutManager);
public javax.swing.JPanel(boolean);
public javax.swing.JPanel();

Important Components :

1. **Label :** With the JLabel class, you can display unselectable text and images.
   Constructors-
   JLabel(Icon i)
   JLabel(Icon I , int n)
   JLabel(String s)
   JLabel(String s, Icon i, int n)
   JLabel(String s, int n)
   JLabel()
   The int argument specifies the horizontal alignment of the label's contents within its drawing area; defined in the SwingConstants interface (which JLabel implements): LEFT (default), CENTER, RIGHT, LEADING, or TRAILING.

   Methods-

1. Set or get the text displayed by the label.
   void setText(String) String getText()
2. Set or get the image displayed by the label.
   void setIcon (Icon) Icon getIcon()
3. Set or get the image displayed by the label when it's disabled. If you don't specify a disabled image, then the look-and-feel creates one by manipulating the default image.
   void setDisabledIcon(Icon) Icon getDisabledIcon()
4. Set or get where in the label its contents should be placed. For vertical alignment: TOP, CENTER (the default), and BOTTOM.
   void setHorizontalAlignment(int)
   void setVerticalAlignment(int)
   int getHorizontalAlignment()
   int getVerticalAlignment()

2. **Button :** A Swing button can display both text and an image. The underlined letter in each button's text shows the mnemonic which is the keyboard alternative.
   Constructors-
   JButton(Icon I)
   JButton(String s)
   JButton(String s, Icon I)
   Methods-
   void setDisabledIcon(Icon)
   void setPressedIcon(Icon)
   void setSelectedIcon(Icon)
   void setRolloverIcon(Icon)
   String getText()

void setText(String)
Event- ActionEvent

3. **Check boxes :**
   Class - JCheckBox
   Constructors-
   JCheckBox(Icon i)
   JCheckBox(Icon i,booean state)
   JCheckBox(String s)
   JCheckBox(String s, boolean state)
   JCheckBox(String s, Icon i)
   JCheckBox(String s, Icon I, boolean state)

   Methods-
   void setSelected(boolean state)
   String getText()
   void setText(String s)
   Event- ItemEvent

4. **Radio Buttons** :
   Class- JRadioButton
   Constructors-
   JRadioButton (String s)
   JRadioButton(String s, boolean state)
   JRadioButton(Icon i)
   JRadioButton(Icon i, boolean state)
   JRadioButton(String s, Icon i)
   JRadioButton(String s, Icon i, boolean state)
   JRadioButton()

To create a button group- ButtonGroup()
Adds a button to the group, or removes a button from the group.
void add(AbstractButton)
void remove(AbstractButton)

5. **Combo Boxes** :
   Class- JComboBox

   Constructors-
   JComboBox()

   Methods-
   void addItem(Object)
   Object getItemAt(int)
   Object getSelectedItem()
   int getItemCount()
   Event- ItemEvent
6. **List**
   Constructor- JList(ListModel)

List models-

1. SINGLE_SELECTION - Only one item can be selected at a time. When the user selects an item, any previously selected item is deselected first.
2. SINGLE_INTERVAL_SELECTION- Multiple, contiguous items can be selected. When the user begins a new selection range, any previously selected items are deselected first.
3. MULTIPLE_INTERVAL_SELECTION- The default. Any combination of items can be selected. The user must explicitly deselect items.

Methods-

```
boolean isSelectedIndex(int)
void setSelectedIndex(int)
void setSelectedIndices(int[])
void setSelectedValue(Object, boolean)
void setSelectedInterval(int, int)
int getSelectedIndex()
int getMinSelectionIndex()
int getMaxSelectionIndex()
int[] getSelectedIndices()
Object getSelectedValue()
Object[] getSelectedValues()
```

Example-

```
listModel = new DefaultListModel();
listModel.addElement("India");
listModel.addElement("Japan");
listModel.addElement("France");
listModel.addElement("Denmark");
list = new JList(listModel);
```

Event- ActionEvent

7. Text:

classes All text related classes are inherited from JTextComponent class

a. JTextField

Creates a text field. The int argument specifies the desired width in columns. The String argument contains the field's initial text. The Document argument provides a custom document for the field.

Constructors-

```
JTextField()
JTextField(String)
JTextField(String, int)
JTextField(int)
JTextField(Document, String, int)
```

b. JPasswordField

Creates a password field. When present, the int argument specifies the desired width in columns. The String argument contains the field's initial text. The Document argument provides a custom document for the field.

Constructors-

JPasswordField()
JPasswordField(String)
JPasswordField(String, int)
JPasswordField(int)
JPasswordField(Document, String, int)

Methods-

1. Set or get the text displayed by the text field.
   void setText(String) String getText()
2. Set or get the text displayed by the text field.
   char[] getPassword()
3. Set or get whether the user can edit the text in the text field.
   void setEditable(boolean) boolean isEditable()
4. Set or get the number of columns displayed by the text field. This is really just a hint for computing the field's preferred width.
   void setColumns(int); int getColumns()
5. Get the width of the text field's columns. This value is established implicitly by the font.
   int getColumnWidth()
6. Set or get the echo character i.e. the character displayed instead of the actual characters typed by the user.
   void setEchoChar(char) char getEchoChar()

Event- ActionEvent

c. JTextArea

Represents a text area which can hold multiple lines of text

Constructors-

JTextArea (int row, int cols)
JTextArea (String s, int row, int cols)

Methods-

void setColumns (int cols)
void setRows (int rows)
void append(String s)
void setLineWrap (boolean)

8.Dialog Boxes:

Types-

1. **Modal-** wont let the user interact with the remaining windows of application
   until first deals with it. Ex- when user wants to read a file, user must specify file name before prg. can begin read operation.
2. Modeless dialog box- Lets the user enters information in both, the dialog box & remainder of application ex- toolbar.

Swing has a JOptionPane class, that lets you put a simple dialog box.
Methods in JOption Class

1. static void showMessageDialog()- Shows a message with ok button.
2. static int showConfirmDialog()- shows a message & gets users options from set of options.

3. static int showOptionDialog- shows a message & get users options from set of options.
4. String showInputDialog()- shows a message with one line of user input.

9. Menu:

Creating and Setting Up Menu Bars

Constructor or Method	Purpose
JMenuBar()	Creates a menu bar.
JMenu add(JMenu)	Creates a menu bar.
void setJMenuBar(JMenuBar) JMenuBar getJMenuBar()	Sets or gets the menu bar of an applet, dialog, frame, internal frame, or root pane.

Creating and Populating Menus

Constructor or Method	Purpose
JMenu() JMenu(String)	Creates a menu. The string specifies the text to display for the menu.
JMenuItem add(JMenuItem) JMenuItem add(Action) JMenuItem add(String)	Adds a menu item to the current end of the menu. If the argument is an Action object, then the menu creates a menu item. If the argument is a string, then the menu automatically creates a JMenuItem object that displays the specified text.
void addSeparator()	Adds a separator to the current end of the menu.
JMenuItem insert(JMenuItem, int) JMenuItem insert(Action, int) void insert(String, int) void insertSeparator(int)	Inserts a menu item or separator into the menu at the specified position. The first menu item is at position 0, the second at position 1, and so on. The JMenuItem, Action, and String arguments are treated the same as in the corresponding add methods.
void remove(JMenuItem) void remove(int) void removeAll()	Removes the specified item(s) from the menu. If the argument is an integer, then it specifies the position of the menu item to be removed.

Implementing Menu Items

Constructor or Method	Purpose
JMenuItem() JMenuItem(String) JMenuItem(Icon) JMenuItem(String, Icon) JMenuItem(String, int)	Creates an ordinary menu item. The icon argument, if present, specifies the icon that the menu item should display. Similarly, the string argument specifies the text that the menu item should display. The integer argument specifies the keyboard mnemonic to use. You can specify any of the relevant VK constants defined in the KeyEvent class. For example, to specify the A key, use KeyEvent.VK_A.
JCheckBoxMenuItem() JCheckBoxMenuItem(String) JCheckBoxMenuItem(Icon)	Creates a menu item that looks and acts like a check box. The string argument, if any, specifies the text that the menu item should

JCheckBoxMenuItem(String, Icon) JCheckBoxMenuItem(String, boolean) JCheckBoxMenuItem(String, Icon, boolean)	display. If you specify true for the boolean argument, then the menu item is initially selected (checked). Otherwise, the menu item is initially unselected.
JRadioButtonMenuItem() JRadioButtonMenuItem(String) JRadioButtonMenuItem(Icon) JRadioButtonMenuItem(String, Icon) JRadioButtonMenuItem(String, boolean) JRadioButtonMenuItem(Icon, boolean) JRadioButtonMenuItem(String, Icon, boolean)	Creates a menu item that looks and acts like a radio button. The string argument, if any, specifies the text that the menu item should display. If you specify true for the boolean argument, then the menu item is initially selected. Otherwise, the menu item is initially unselected.
void setState(boolean) boolean getState() (in JCheckBoxMenuItem)	Set or get the selection state of a check box menu item.
void setEnabled(boolean)	If the argument is true, enable the menu item. Otherwise, disable the menu item.

Event handling is an important part of GUI based applications. Events are generated by event sources. A mouse click, Window closed, key typed etc. are examples of events. All java events are sub-classes of java.awt.AWTEvent class.

Java has two types of events:

1. **Low-Level Events:** Low-level events represent direct communication from user. A low level event is a key press or a key release, a mouse click, drag, move or release, and so on.

Following are low level events.

Event	**Description**
ComponentEvent	Indicates that a component object (e.g. Button, List, TextField) is moved, resized, rendered invisible or made visible again.
FocusEvent	Indicates that a component has gained or lost the input focus.
KeyEvent	Generated by a component object (such as TextField) when a key is pressed, released or typed.
MouseEvent	Indicates that a mouse action occurred in a component. E.g. mouse is pressed, releases, clicked (pressed and released), moved or dragged.
ContainerEvent	Indicates that a container's contents are changed because a component was added or removed.
WindowEvent	Indicates that a window has changed its status. This low level event is generated by a Window object when it is opened, closed, activated, deactivated, iconified, deiconified or when focus is transferred into or out of the Window.

2. **High-Level Events:** High-level (also called as semantic events) events encapsulate the

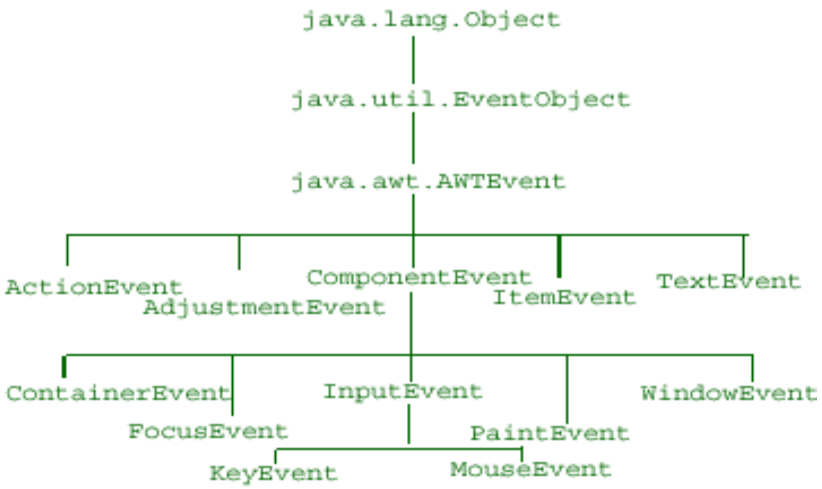
meaning of a user interface component. These include following events.

Event	**Description**
ActionEvent	Indicates that a component-defined action occurred. This high-level event is generated by a component (such as Button) when the component-specific action occurs (such as being pressed).
AdjustmentEvent	The adjustment event is emitted by Adjustable objects like scrollbars.
ItemEvent	Indicates that an item was selected or deselected. This high-level event is generated by an ItemSelectable object (such as a List) when an item is selected or deselected by the user.
TextEvent	Indicates that an object's text changed. This high-level event is generated by an object (such as TextComponent) when its text changes.

The following table lists the events, their corresponding listeners and the method to add the listener to the component.

Event	**Event Source**	**Event Listener**	**Method to add listener to event source**
Low – Level Events			
ComponentEvent	Component	ComponentListener	addComponentListener()
FocusEvent	Component	FocusListener	addFocusListener()
KeyEvent	Component	KeyListener	addKeyListener()
MouseEvent	Component	MouseListener MouseMotionListener	addMouseListener() addMouseMotionListener()
ContainerEvent	Container	ContainerListener	addContainerListener()
WindowEvent	Window	WindowListener	addWindowListener()
High-level Events			
ActionEvent	Button List MenuItem TextField	ActionListener	addActionListener()
ItemEvent	Choice CheckBox CheckBoxMenuItem List	ItemListener	addItemListener()
AdjustmentEvent	Scrollbar	AdjustmentListener	addAdjustmentListener()
TextEvent	TextField TextArea	TextListener	addTextLIstener()

Event class hierarchy



Listener Methods:

Methods	**Description**
ComponentListener	
componentResized(ComponentEvent e)	Invoked when component's size changes.
componentMoved(ComponentEvent e)	Invoked when component's position changes.
componentShown(ComponentEvent e)	Invoked when component has been made visible.
componentHidden(ComponentEvent e)	Invoked when component has been made invisible.
FocusListener	
focusGained(FocusEvent e) Invoked	Invoked when component gains the keyboard focus.
focusLost(FocusEvent e)	when component loses the keyboard focus.
KeyListener	
keyTyped(KeyEvent e)	Invoked when a key is typed.
keyPressed(KeyEvent e)	Invoked when a key is pressed.
keyReleased (KeyEvent e)	Invoked when a key is released.
MouseListener	
mouseClicked(MouseEvent e)	Invoked when a mouse button is clicked (i.e. pressed and released) on a component.
mousePressed(MouseEvent e)	Invoked when a mouse button is pressed on a component.
mouseReleased(MouseEvent e)	Invoked when a mouse button is released on a component.
mouseEntered(MouseEvent e)	Invoked when a mouse enters a component.
mouseExited(MouseEvent e)	Invoked when a mouse exits a component.
MouseMotionListener	
mouseDragged(MouseEvent e)	Invoked when a mouse button is pressed on a component and then dragged.
mouseMoved(MouseEvent e)	Invoked when a the mouse cursor is moved on to a component but mouse button is not pressed.
ContainerListener	

componentAdded(ContainerEvent e)	Invoked when a component is added to the container.
componentRemoved(ContainerEvent e)	Invoked when a component is removed from the container.
WindowListener	
windowOpened(WindowEvent e)	Invoked the first time a window is made visible
windowClosing(WindowEvent e)	Invoked when the user attempts to close the window from the window's system menu.
windowClosed(WindowEvent e)	Invoked when a window has been closed as the result of calling dispose on the window.
windowIconified(WindowEvent e)	Invoked when a window is changed from a normal to a minimized state.
windowDeiconified(WindowEvent e)	Invoked when a window is changed from minimized to normal state.
windowActivated(WindowEvent e)	Invoked when the window is set to be the active window.
windowDeactivated(WindowEvent e)	Invoked when the window is no longer the active window.
ActionListener	
actionPerformed(ActionEvent e)	Invoked when an action occurs.
ComponentListsner	
itemStateChanged(ActionEvent e)	Invoked when anitem has been selected oe deselected by the user.
AdjustmentListener	
adjustmentValueChanged(ActionEvent e)	Invoked when the value of the adjustable has changed.
TextListener	
textValueChanged(ActionEvent e)	Invoked when the value of the text has changed.

Adapter Classes:

All high level listeners contain only one method to handle the high-level events. But most low level event listeners are designed to listen to multiple event subtypes (i.e. the MouseListener listens to mouse-down, mouse-up, mouse-enter, etc.). AWT provides a set of abstract "adapter" classes, which implements each listener interface. These allow programs to easily subclass the Adapters and override only the methods representing event types they are interested in, instead of implementing all methods in listener interfaces.

The Adapter classes provided by AWT are as follows:

java.awt.event.ComponenentAdapter
java.awt.event.ContainerAdapter
java.awt.event.FocusAdapter
java.awt.event.KeyAdapter
java.awt.event.MouseAdapter
java.awt.event.MouseMotionAdapter
java.awt.event.WindowAdapter

Applet

Applets are small java programs which are executed and displayed in a java compatible web browser.

Creating an applet

All applets are subclasses of the java.applet.Applet class. You can also create an applet by

extending the javax.swing.JApplet class.

The syntax is:

```
class MyApplet extends Applet
{
  //applet methods
}
```

Applet methods:

Method	**Purpose**
init( )	Automatically called to perform initialization of the applet. Executed only once.
start( )	Called every time the applet moves into sight on the Web browser to allow the applet to start up its normal operations.
stop( )	Called every time the applet moves out of sight on the Web browser to allow the applet to shut off expensive operations.
destroy( )	Called when the applet is being unloaded from the page to perform final release of resources when the applet is no longer used.
paint()	Called each time the applets output needs to be redrawn.

Running an applet

1. Compile the applet code using javac
2. Use the java tool – appletviewer to view the applet (embed the APPLET tag in comments in the code)
3. Use the APPLET tag in an HTML page and load the applet in a browser

Using appletviewer:

1. Write the HTML APPLET tag in comments in the source file.
2. Compile the applet source code using javac.
3. Use appletviewer ClassName.class to view the applet.

Using browser:

1. Create an HTML file containing the APPLET tag.
2. Compile the applet source code using javac.
3. In the web browser, open the HTML file.

The APPLET tag

```
< APPLET  [CODEBASE  = appletURL]
  CODE  = appletClassFile
  [ALT  = alternateText]
  [ARCHIVE = archiveFile]
  [NAME  = appletInstanceName]
  WIDTH  = pixels   HEIGHT  = pixels
  [ALIGN  = alignment]
  [VSPACE = pixels]
  [HSPACE = pixels]
>
  [< PARAM NAME = AttributeName VALUE = AttributeValue />]
  </APPLET>
```

Attribute	Value	Meaning
align	left right top bottom middle baseline	Specifies the alignment of an applet according to surrounding elements.
alt	text	Specifies an alternate text for an applet.
archive	URL	Specifies the location of an archive file.
code	URL	Specifies the file name of a Java applet .
codebase	URL	Specifies a relative base URL for applets specified in the code attribute
height	pixels	Specifies the height of an applet.
hspace	pixels	Defines the horizontal spacing around an applet.
name	name	Defines the name for an applet (to use in scripts).
vspace	pixels	Defines the vertical spacing around an applet.
width	pixels	Specifies the width of an applet

The mandatory attributes are CODE, HEIGHT and WIDTH.

Examples:

1. <applet code=MyApplet width=200 height=200 archive="files.jar">
   </applet>
2. <applet code=Simple.class width=100 height=200 codebase=”example/”>
   </applet>

Passing parameters to applets

The PARAM tag allows us to pass information to an applet when it starts running. A parameter is a NAME – VALUE pair. Every parameter is identified by a name and it has a value.

```
< PARAM NAME = AttributeName VALUE = AttributeValue />
```

Example:

```
<APPLET  NAME = "MyApplet.class" WIDTH = 100 HEIGHT = 100>
<PARAM NAME = "ImageSource" VALUE = "project/images/">
<PARAM NAME = "BackgroundColor" VALUE = "0xc0c0c0">
<PARAM NAME = "FontColor" VALUE = "Red">
</APPLET>
```

The Applet can retrieve information about the parameters using the getParameter() method.

```
String getParameter(String parameterName);
```

Example:

```
String dirName = getParameter(“ImageSource”);
Color c = new Color( Integer.parseInt(getParameter(“BackgroundColor”)));
```

paint(), repaint() and update()

The paint() method redraws the applet.
The repaint() method is used to force redrawing of the applet.
The update() method redraws only a portion of the applet.

Self Activity

Sample program 1: Program to demonstrate Button and text field

```
import java.awt.event.*;
import javax.swing.*;
import java.awt.*;
 public class JButtonDemo extends JFrame implements ActionListener
{
     JTextField jtf;
     JButton jb;
      public JButtonDemo()
      {
         setLayout(new FlowLayout());
          jtf = new JTextField(15);
          add (jtf);
          jb = new JButton (“Click Me”);
          jb.addActionListener (this);
          add(jb);
          setSize(200,200);
          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          setVisible(true);
         }
       public void actionPerformed(ActionEvent ae)
       {
                jtf.setText (ae.getActionCommand());
        }
        public static void main(String[] args)
        {
                new JButtonDemo();
        }
       }
```

Sample program 2 : Program to demonstrate Combobox

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class JCdemo extends JFrame implements ItemListener
{
   JTextField  jtf;
   JCheckBox jcb1, jcb2;
   public JCdemo()
   {
       setLayout(new FlowLayout());
       jcb1 = new JCheckBox("Swing Demos");
       jcb1.addItemListener(this);
       add(jcb1);
       jcb2 = new JCheckBox("Java Demos");
       jcb2.addItemListener(this);
       add(jcb2);
```

```
                jtf = new JTextField(35);
                add(jtf);
                setSize(200,200);
                setVisible(true);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                }
                public void itemStateChanged (ItemEvent ie)
                {
                        String text = " ";
                         if(jcb1.isSelected())
                            text = text + jcb1.getText() + " ";
                         if(jcb2.isSelected())
                             text = text + jcb2.getText();
                         jtf.setText(text);
                  }
              public static void main(String[] args)
                 {
                    new JCdemo();
                 }
          }
```

Sample program 3: Program to demonstrate Radio Button

```
          import java.awt.*;
          import javax.swing.*;
          import java.awt.event.*;
          public class JRdemo extends JFrame implements ActionListener
           {
               JTextField  jtf;
               JRadioButton jrb1,jrb2;
               ButtonGroup bg;
                public JRdemo()
                 {
                       setLayout(new FlowLayout());
                       bg = new ButtonGroup();
                       jrb1 = new JRadioButton("A");
                       jrb1.addActionListener(this);
                       bg.add(jrb1);
                       add(jrb1);
                       jrb2 = new JRadioButton("B");
                       jrb2.addActionListener(this);
                       bg.add(jrb2);
                       add(jrb2);
                       jtf = new JTextField(5);
                       add(jtf);
                       setSize(200,200);
                       setVisible(true);
                       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                  }
                 public void actionPerformed (ActionEvent ae)
                  {
                        jtf.setText(ae.getActionCommand());
```

```
                }

        public static void main(String[] args)
            {
                new JRdemo();
             }
    }
```

Sample program 4 : Program to handle mouse movements and key events on a frame

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class EventTest extends JFrame
 {
    JLabel l = new JLabel();
    EventTest()
     {
        setLayout(new FlowLayout());
        add(l);
        addKeyListener(new KeyAdapter()
        {
           public void keyTyped(KeyEvent ke)
           {
                l.setText("You typed " + ke.getKeyChar());
            }
          });

        addMouseMotionListener(new MouseMotionAdapter()
            {
            public void mouseMoved(MouseEvent me)
                {
                   l.setText("Mouse moved : X = "+ me.getX() + "Y = " +
                        me.getY());
                 }
          });

          setSize(200,200);
          setVisible(true);
          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     }
      public static void main(String[] args)
       {
          new EventTest();
       }
    }
```

Sample program 5 : Program to display a message in an applet

```
import java.awt.*;
import java.applet.*;
/* <applet code="MyApplet.class" width=200 height=100>
   </applet>
*/
```

```
public class MyApplet extends Applet
{
    public void paint(Graphics g)
        {
            g.drawString("My First Applet", 20,20);
         }
}
```

Save this as MyApplet.java. Compile and Execute it using command – appletviewer MyApplet.class

Sample program 6 : Applet with components

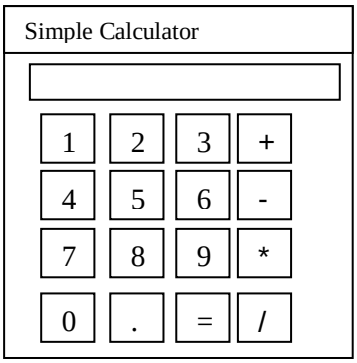
```
import java.awt.*;
import javax.swing.* ;
import java.applet.*;
/* <applet code="MyApplet.class" width=200 height=100>
    </applet>
*/
   public class MyApplet extends Applet
    {
         JPanel p;
         JTextField t;
         JButton b;
         public void init()
         {
                p = new JPanel();
                p.setLayout(new FlowLayout());
                t = new JTextField(20);
                b = new JButton("Click");
                p.add(t);
                p.add(b);
               add(p);
         }
     }
```

Save this as MyApplet.java Compile the file. Execute it using command – appletviewer MyApplet.class.

Lab Assignments

Set A

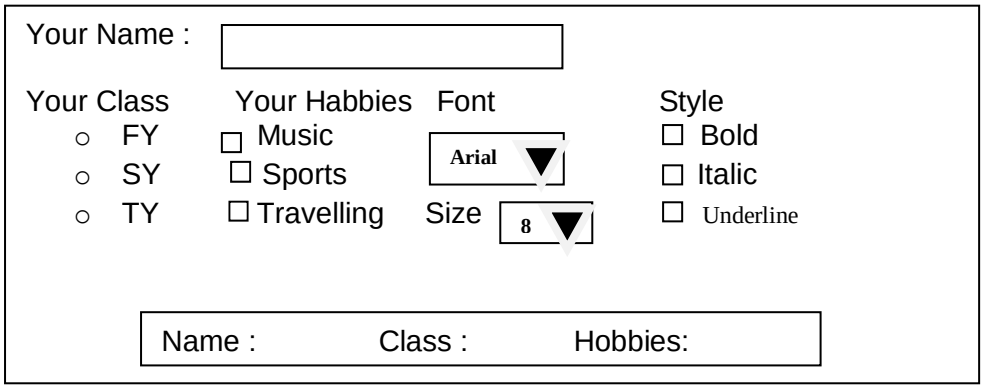
a) Write a java program that works as a simple calculator. Use a grid layout to arrange buttons for the digits and for the +, -, *, % operations. Add a text field to display the result.



b) Design a screen to handle the Mouse Events such as MOUSE_MOVED and MOUSE_CLICK and display the position of the Mouse_Click in a TextField.

Set B

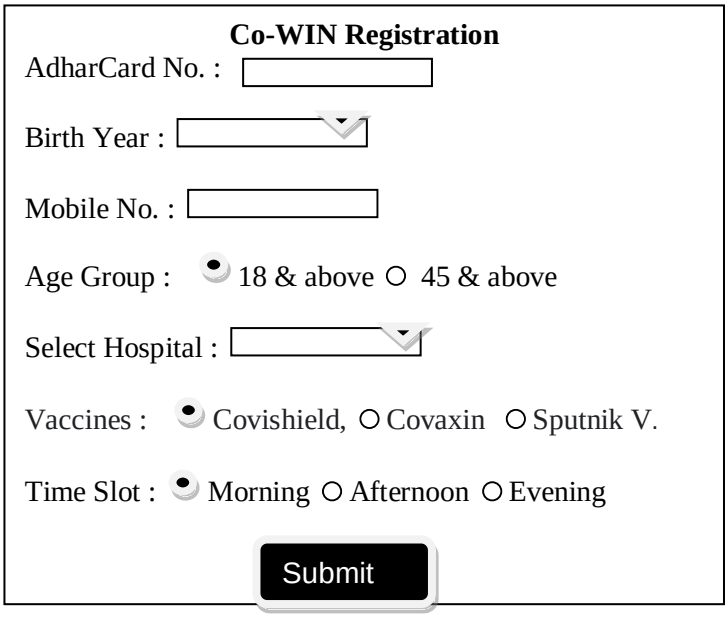
a) Create the following GUI screen using appropriate layout managers. Accept the name, class , hobbies of the user and apply the changes and display the selected options in a text box.



b) Write a Java program to design a screen using Awt that will take a user name and password. If the user name and password are not same, raise an Exception with appropriate message. User can have 3 login chances only. Use clear button to clear the TextFields.

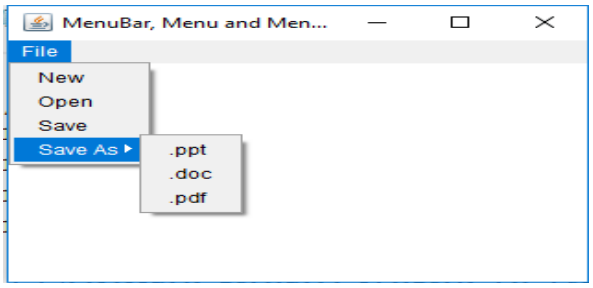
Set C

a) Write a java program to create the following GUI for user registration form



If above conditions are met, display "Registration Successful" otherwise "Registration Failed" after the user clicks the Submit button.

b) Write a program to display the following menus and sub-menus.



Additional Programs for practice

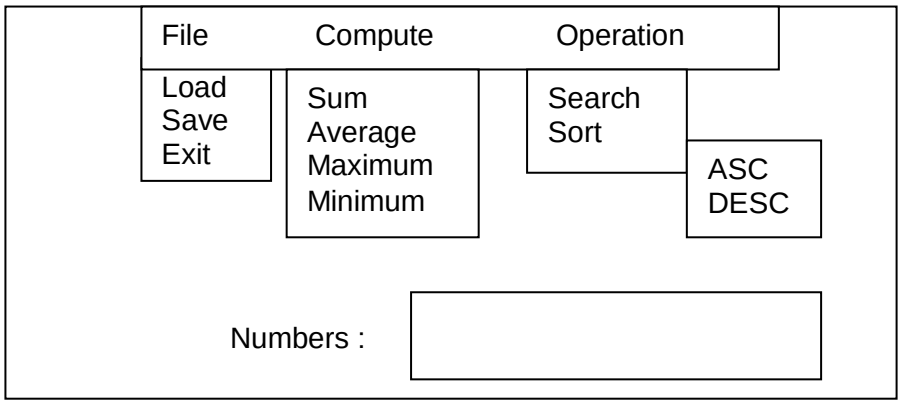
a) Write Java program to design three text boxes and two buttons using swing . Enter different strings in first and second textbox. On clicking the First command button, concatenation of two strings should be displayed in third text box and on clicking second command button , reverse of string should display in third text box.

b) Create an Applet which displays a message in the center of the screen. The message indicates the events taking place on the applet window. Handle events like mouse click, mouse moved, mouse dragged, mouse pressed, and key pressed. The message should update each time an event occurs. The message should give details of the event such as which mouse button was pressed, which key is pressed etc. (Hint: Use

repaint(), KeyListener, MouseListener, MouseEvent method getButton, KeyEvent methods getKeyChar)

c) Write a java program to create the following GUI for user registration form. Perform the following validations: i. Password should be minimum 6 characters containing atleast one uppercase letter, one digit and one symbol. ii. Confirm password and password fields should match. iii. The Captcha should generate two random 2 digit numbers and accept the sum from the user. If above conditions are met, display "Registration Successful" otherwise "Registration Failed" after the user clicks the Submit button.

d) Write a menu driven program to perform the following operations on a set of integers. The Load operation should generate 50 random integers (2 digits) and display the numbers on the screen. The save operation should save the numbers to a file "numbers.txt". The Compute menu provides various operations and the result is displayed in a message box. The Search operation accepts a number from the user in an input dialog and displays the search result in a message dialog. The sort operation sorts the numbers and displays the sorted data on the screen.



Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge