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

Laboratory Course II
Programming in Java - CS348

Semester I

(From Academic Year 2015)

Name _____ **Roll No.** _____

College _____ **Division** _____

Academic Year _____

PREPARED BY:

PROF. MS. POONAM PONDE (NOWROSJEE WADIA COLLEGE)

PROF. JEEVAN LIMAYE (FERGUSSON COLLEGE)

FIRST EDITION AUTHORS:

Ms. Poonam Ponde

Ms. Seema Jawale

Ms. Kalpana Joshi

Ms. Jayshri Patil

Ms. Ranjana Shevkar

ABOUT THE WORK BOOK

- **OBJECTIVES OF THIS BOOK**

This lab-book is intended to be used by T.Y.B.Sc(Computer Science) students for Laboratory course – II (Programming in Java) , Semester I.
The objectives of this book are

a. Covers the complete scope of the syllabus.

b. Bringing uniformity in the way course is conducted across different colleges.

c. Continuous assessment of the students.

d. Providing ready references for students while working in the lab.

- **How to use this book?**

This book is mandatory for the completion of the laboratory course. It is a measure of the performance of the student in the laboratory for the entire duration of the course.

- **Instructions to the students**

1. Students should carry this book during practical sessions.

2. Print outs of source code and outputs is optional

3. Student should read the topics mentioned in **Reading section** of this book before coming for the practical session.

4. Students should solve those exercises which are selected by Practical in-charge as a part of journal activity. However, students are free to solve additional exercises for more practice.

5. Each assignment will be assessed on a scale of 0 to 5 as indicated below.

i) Not done	0
ii) Incomplete	1
iii) Late Complete	2
iv) Needs improvement	3
v) Complete	4
vi) Well Done	5

- **Difficulty Levels**

Self Activity: Students should solve these exercises for practice only.
SET A - Easy: All exercises are compulsory.
SET B - Medium: All exercises are compulsory.

- **Instruction to the Instructors**

1) Make sure that students follow the instruction as given above.

2) After a student completes a specific set, the instructor has to verify the outputs and sign in the space provided after the activity.

3) Evaluate each assignment on a scale of 5 as specified above by ticking appropriate box.

4) The value should also be entered on assignment completion page of the respective Lab course.

5) Students should be encouraged to use an IDE like Eclipse for their assignments and project work.

Roll No :
Name :

Assignment Completion Sheet

Sr. No	Assignment Name	Marks
1	Java Tools and IDE, Simple java programs	
2	Array of Objects and Packages	
3	Inheritance and Interfaces	
4	Exception Handling	
5	I/O and File Handling	
6	GUI Designing, Event Handling and Applets	
	Total out of 30	
	Total out of 5	

Signature of Incharge:

Examiner I:

Examiner II:

Date:

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)**
- **Defining simple classes and creating objects.**

Reading

You should read the following topics before starting this exercise

1. Creating, compiling and running a java program.
2. The java virtual machine.
3. Java tools like javac, java, javadoc, javap and jdb.
4. Java keywords
5. 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 author-information	Author-information contains author name, and / or author email address or any other appropriate information.
@version	@version version-information	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

(4) 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

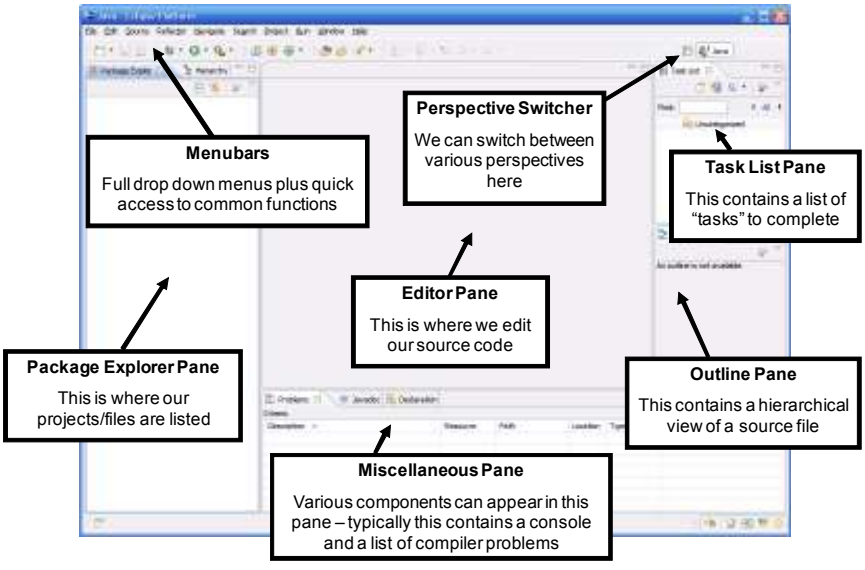
```
 CLASSPATH=.:/usr/local/classes.jar:/home/user1/myclasses
export CLASSPATH
```

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)

1. Sample program

```
/* 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

2. Sample program

```
/* 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.

Answer the following questions :

1. How will you pass command line argument using IDE?

_____

_____

2. Write the output if the command line argument passed is “ABC”.

_____

Lab Assignments

SET A

1. Using javap, view the methods of the following classes from the lang package: java.lang.Object , java.lang.String and java.util.Scanner.

2. Compile sample program 2. Type the following command and view the bytecodes.

```
javap -c MyClass
```

SET B

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

   Current date is : 31/07/2015
   Current date is : 07-31-2015
   Current date is : Friday July 31 2015
   Current date and time is : Fri July 31 16:25:56 IST 2015
   Current date and time is : 31/07/15 16:25:56 PM +0530
   Current time is : 16:25:56
   Current week of year is : 31
   Current week of month : 5
   Current day of the year is : 212

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

2. 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.

Signature of the instructor ☐ Date ☐

Assignment Evaluation

0: Not done ☐	2: Late Complete ☐	4: Complete ☐
1: Incomplete ☐	3: Needs improvement ☐	5: Well Done ☐

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:
1. Structure of a class in java.
2. Declaring class reference.
3. Creating an object using new.
4. Declaring an array of references.
5. Creating an array of objects.
6. 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) {
    // body of method
  }
  type methodname2(parameter-list) {
    // body of method
  }
  // ...
  type methodnameN(parameter-list) {
    // body of method
  }
}
```

Example

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

Creating objects:

```
ClassName referenceName;
referenceName = new ClassName();
```

OR

```
ClassName referenceName = new ClassName();
```

Example:

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

Overriding toString 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, use the syntax:

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

Example

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

Declaring an array of references:

```
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();
```

To convert the argument from String to any type, use Wrapper classes.

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

Simple I/O

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

```
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:

java.lang	Language support classes such as Math, Thread, String
java.util	Utility classes such as LinkedList, Vector , Date.
java.io	Input/Output classes
java.awt	For graphical user interfaces and event handling.
javax.swing	For graphical user interfaces
java.net	For networking
java.applet	For creating applets.

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.

```
package packageName;
```

Accessing a package

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

```
import packageName.*; //imports all classes
import packageName.className; //imports specified class
```

Note that the package can have a hierarchy of subpackages. In that case, the package name should be qualified using its parent packages. *Example:* project.sourcecode.java
Here, the package named project contains one subpackage named sourcecode which contains a subpackage named java.

Access Rules

The access rules for members of a class are given in the table below.

Accessible to	public	protected	none	private
Same class	Yes	Yes	Yes	Yes
Class in same package	Yes	Yes	Yes	No
Subclass (in other package)	Yes	Yes	No	No
Non subclass in Other package	Yes	No	No	No

Self Activity

1. Sample program 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 String toString()   {
      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);
   }
}
```

2. Sample program 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)    throws IOException
  {
    int rollNumber;
    String name;
    BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
    System.out.println("Enter the roll number: ");
    rollNumber = Integer.parseInt(br.readLine());
    System.out.println(" Enter the name: ");
    name = br.readLine();
    System.out.println(" Roll Number = "  + rollNumber);
    System.out.println(" Name = "  + name);
  }
}
```

3. Sample program to read Student roll number and name from the console and display them (Using Scanner class).

```
import java.util.Scanner;
class ScannerTest{
     public static void main(String args[])throws Exception
     {
       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();
     }
}
```

4. Sample program to create and use packages

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

```
public class B {
      public void display() {
            System.out.println("In display of B");
      }
}
//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 inside P1. Save B.java in P2.

Lab Assignments

SET A

1. Define a Student class (roll number, name, percentage). Define a default and parameterized constructor. 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.

2. Modify the above program 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.

SET B

1. Create a package named **Series** having three different classes to print series:
   a. Prime numbers b. Fibonacci series c. Squares of numbers
Write a program to generate 'n' terms of the above series.

2. 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.

Signature of the instructor ☐ Date ☐

Assignment Evaluation

0: Not done ☐ 2: Late Complete ☐ 4: Complete ☐

1: Incomplete ☐ 3: Needs improvement ☐ 5: Well Done ☐

Assignment 3: Inheritance and Interfaces

Objectives

- **To implement inheritance in java.**
- **To define abstract classes.**
- **To define and use interfaces.**
- **Use predefined interfaces like Cloneable**

Reading

You should read the following topics before starting this exercise:

1. Concept of inheritance.
2. Use of extends keyword.
3. Concept of abstract class.
4. Defining an interface.
5. Use of implements keyword.

Ready Reference

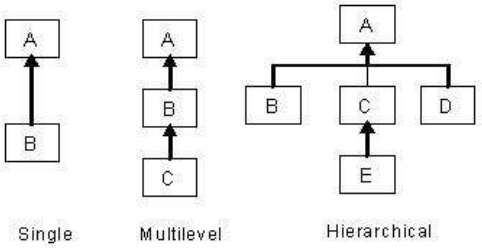
Inheriting a class : The syntax to create a subclass is :

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

Example:

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

Types of Inheritance



Access in subclass

The following members can 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:

```
class A
{ 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.

```
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 class B
```

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.

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

Abstract method

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

```
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.

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

Example:

```
interface MyInterface
{
  void method1();
  void method2();
  int size= 10; //final and static
}
class MyClass implements MyInterface {
  //define method1 and method2
}
```

Self Activity

1. Sample program 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 Cylinder extends Circle
{
   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());
  }
}
```

Lab Assignments

SET A

1. Define a class Employee having private members – id, name, department, salary. Define default and parameterized constructors. Create a subclass called "Manager" with private member bonus. Define methods accept and display in both the classes. Create n objects of the Manager class and display the details of the manager having the maximum total salary (salary+bonus)

2. Create an abstract class Shape with methods calc_area and calc_volume. Derive three classes Sphere(radius) , Cone(radius, height) and Cylinder(radius, height), Box(length, breadth, height) from it. Calculate area and volume of all. (Use Method overriding).

3. Write a Java program to create a super class **Vehicle** having members Company and price. Derive 2 different classes LightMotorVehicle (members – mileage) and HeavyMotorVehicle (members – capacity-in-tons). Accept the information for n vehicles and display the information in appropriate form. While taking data, ask the user about the type of vehicle first.

SET B

1. Define an abstract class "Staff" with members name and address. Define two sub-classes of this class – "FullTimeStaff" (department, salary) and "PartTimeStaff" (number-of-hours, rate-per-hour). Define appropriate constructors. Create n objects which could be of either FullTimeStaff or PartTimeStaff class by asking the user's choice. Display details of all "FullTimeStaff" objects and all "PartTimeStaff" objects.

2. Create an interface "CreditCardInterface" with methods : viewCreditAmount(), useCard(), payCredit() and increaseLimit(). Create a class SilverCardCustomer (name, cardnumber (16 digits), creditAmount – initialized to 0, creditLimit - set to 50,000) which implements the above interface. Inherit class GoldCardCustomer from SilverCardCustomer having the same methods but creditLimit of 1,00,000. Create an object of each class and perform operations. Display appropriate messages for success or failure of transactions. (Use method overriding)

i. useCard() method increases the creditAmount by a specific amount upto creditLimit
ii. payCredit() reduces the creditAmount by a specific amount.
iii. increaseLimit() increases the creditLimit for GoldCardCustomers (only 3 times, not more than 5000Rs. each time)

Signature of the instructor ☐ Date ☐

Assignment Evaluation

0: Not done ☐ 2: Late Complete ☐ 4: Complete ☐

1: Incomplete ☐ 3: Needs improvement ☐ 5: Well Done ☐

Assignment 4: Exception Handling

Objectives

- **Demonstrate exception handling mechanism in java**
- **Defining user defined exception classes**

Reading

You should read the following topics before starting this exercise:

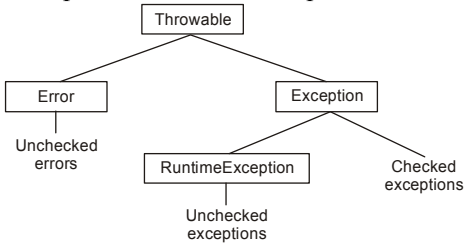
1. Concept of Exception
2. Exception class hierarchy.
3. Use of try, catch, throw, throws and finally keywords
4. Defining user defined exception classes

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("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, **Checked exceptions** and **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
}
```

Self Activity

1. Sample program 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
```

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

Lab Assignments

SET A

1. Define a class CricketPlayer (name, no_of_innings, no_times_notout, total_runs, bat_avg). Create an array of n player objects. Calculate the batting average for each player using a static method avg(). Handle appropriate exception while calculating average. Define a static method "sortPlayer" which sorts the array on the basis of average. Display the player details in sorted order.
2. Define a class SavingAccount (acNo, name, balance). Define appropriate constructors and operations withdraw(), deposit() and viewBalance(). The minimum balance must be 500. Create an object and perform operations. Raise user defined InsufficientFundsException when balance is not sufficient for withdraw operation.

SET B

1. 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 : 12 15 2015, 31 6 1990, 29 2 2001

Signature of the instructor ☐ Date ☐

Assignment Evaluation

0: Not done ☐ 2: Late Complete ☐ 4: Complete ☐

1: Incomplete ☐ 3: Needs improvement ☐ 5: Well Done ☐

Assignment 5: I/O and File Handling

Objectives

- **Performing Input/Output operations using console and files.**

Reading

You should read the following topics before starting this exercise:

1. Concept of streams
2. Types of streams
3. Byte and Character stream classes.
4. The File class

Ready Reference

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

1. boolean canRead()- Returns True if the file is readable.
2. boolean canWrite()- Returns True if the file is writeable.
3. String getName()- Returns the name of the File with any directory names omitted.
4. boolean exists()- Returns true if file exists
5. 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.
6. String getParent()- Returns the directory of the File. If the File is an absolute specification.
7. String getPath()- Returns the full name of the file, including the directory name.
8. boolean isDirectory()- Returns true if File Object is a directory
9. boolean isFile()- Returns true if File Object is a file
10. 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).
11. long length()- Returns the length of the file.
12. boolean delete()- deletes a file or directory. Returns true after successful deletion of a file.
13. boolean mkdir ()- Creates a directory.
14. boolean renameTo (File dest)- Renames a file or directory. Returns true after successful renaming

Example 1:- Checking file existance

```
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 & 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)
```

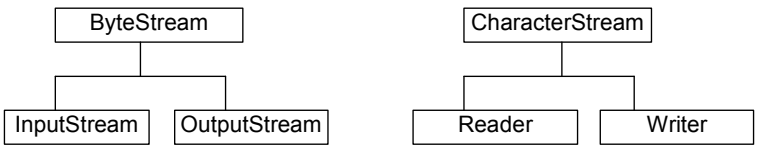
Example :- Using a list( )

```
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

1. 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.

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

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

RandomAccessFile

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

1. Sample program

```
/* 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
```

Compile this program and pass two command line arguments: filename and string to search.

2. Sample program

```
/* 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();
  }
}
```

3. Sample program

```
/* 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();
  }
}
```

Lab Assignments

SET A

1. Write a program to accept a string as command line argument and check whether it is a file or directory. If it is a directory, list the contents of the directory, count how many files the directory has and delete all files in that directory having extension .txt. (Ask the user if the files have to be deleted). If it is a file, display all information about the file (path, size, attributes etc).

2. Write a menu driven program to perform the following operations on a text file "phone.txt" which contains name and phone number pairs. The menu should have options:
   i. Search name and display phone number
   ii. Add a new name-phone number pair.

SET B

1. Write a program to read item information (id, name, price, qty) in file "item.dat". Write a menu driven program to perform the following operations using Random access file:
   i. Search for a specific item by name. ii. Find costliest item. ii. Display all items and total cost

Additional Programs for practice

1. Accept the names of two files and copy the contents of the first to the second. Add Author name and Date in comments in the beginning of file. Add the comment 'end of file' at the end.

2. Write a Java program to accept an option, string and file name using command line argument. Perform following operations:
   a. If no option is passed then print all lines in the file containing the string.
   b. If the option passed is –c then print the count of lines containing the string.
   c. If the option passed is –v then print the lines not containing the string.

Signature of the instructor ☐ Date ☐

Assignment Evaluation

0: Not done ☐ 2: Late Complete ☐ 4: Complete ☐

1: Incomplete ☐ 3: Needs improvement ☐ 5: Well Done ☐

Assignment 6: GUI Designing, Event Handling and Applets

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.**
- **Creating java applets which run in a web browser.**

Reading

You should read the following topics before starting this exercise

1. AWT and Swing concepts.
2. Layout managers in java
3. Containers and Components
4. Adding components to containers
5. Event sources, listeners and delegation event model
6. Adapter classes
7. Applet tag, Applet class, applet methods

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

Class	Description
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

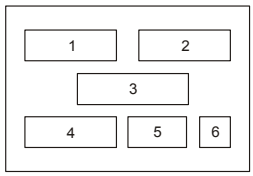
The job of a layout manager is to arrange components on a container. Each container has a layout manager associated with it. To change the layout manager for a container, use the **setLayout()** method.

Syntax

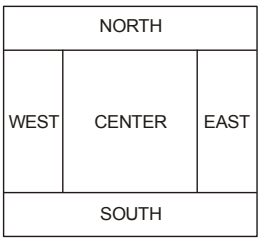
```
setLayout(LayoutManager obj)
```

The predefined managers are listed below:

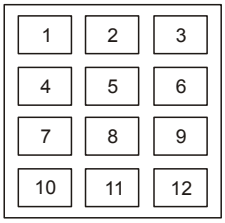
1. FlowLayout 2.BorderLayout 3.GridLayout
4. BoxLayout 5.CardLayout 6.GridBagLayout



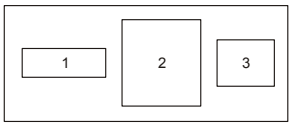
FlowLayout

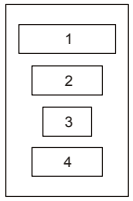


BorderLayout

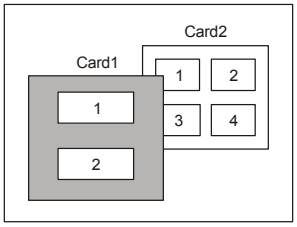


GridLayout

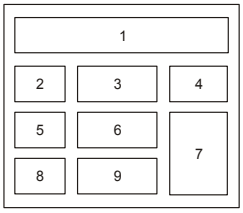




BoxLayout



CardLayout



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	
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	
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) JCheckBoxMenuItem(String, Icon) JCheckBoxMenuItem(String, boolean) JCheckBoxMenuItem(String, Icon, boolean)	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 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.

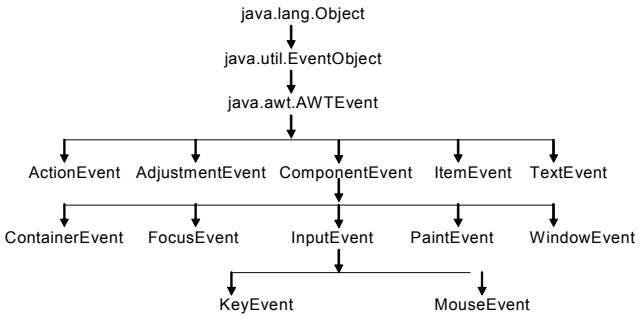
Event	Description
ActionEvent	Indicates that a component-defined action occurred. This high-level event is generated by a component (such as Button) when the component-specific action occurs (such as being pressed).
AdjustmentEvent	The adjustment event is emitted by Adjustable objects like scrollbars.
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 when component gains the keyboard focus.
focusLost(FocusEvent e)	Invoked 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

1. Sample program

```
/* 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();
             }
      }
```

2. Sample program

```
/* 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();
      }
}
```

3. Sample program

```
/* 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(); }
}
```

4. Sample program

```
/* 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();
  }
}
```

5. Sample program

```
/* 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

6. Sample program

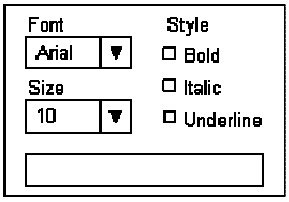
```
/* Applet with components*/
import java.awt.*;
import javax.swing.* ;
import java.applet.*;
/*
<applet code="MyApplet.class" width=200 height=100>
</applet>
*/
public class MyApplet extends Applet
{
  JPanel p;   JTextField t;   JButton b;
  public void init()
  {
    p = new JPanel();
    p.setLayout(new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

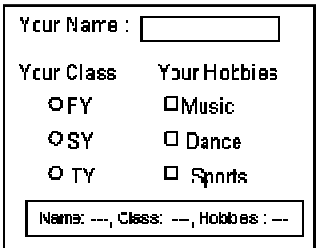
Lab Assignments

SET A

1. Write a program to create the following GUI and apply the changes to the text in the TextField.



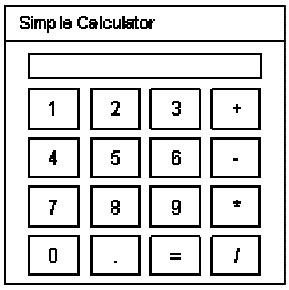
2. Create the following GUI screen using appropriate layout managers. Accept the name, class , hobbies of the user and display the selected options in a text box.



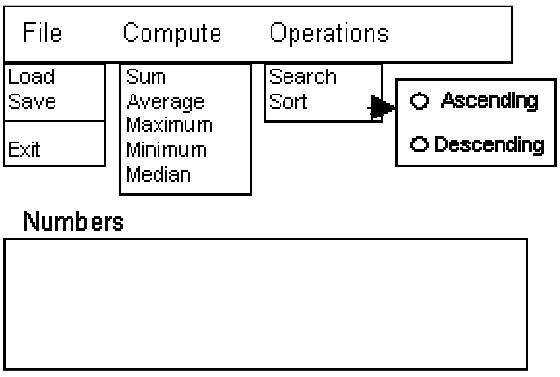
3. Create an Applet which displays a message in the center of the screen. The message indicates the events taking place on the applet window. Handle events like mouse click, mouse moved, mouse dragged, mouse pressed, and key pressed. The message should update each time an event occurs. The message should 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)

SET B

1.Write a java program to implement a simple arithmetic calculator. Perform appropriate validations.



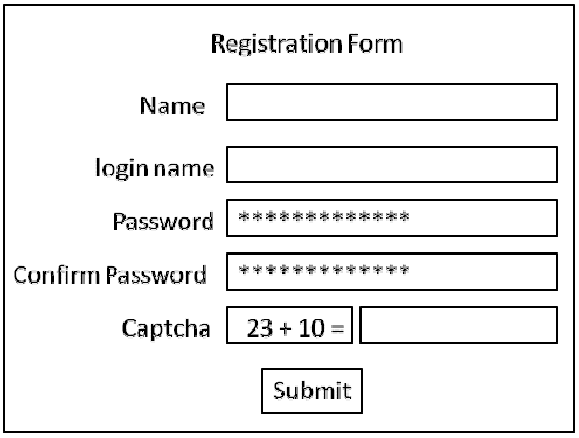
2. Write a menu driven program to perform the following operations on a set of integers. The Load operation should generate 50 random integers (2 digits) and display the numbers on the screen. The save operation should save the numbers to a file "numbers.txt". The Compute menu provides various operations 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.



3. Write a java program to create the following GUI for user registration form. Perform the following validations:

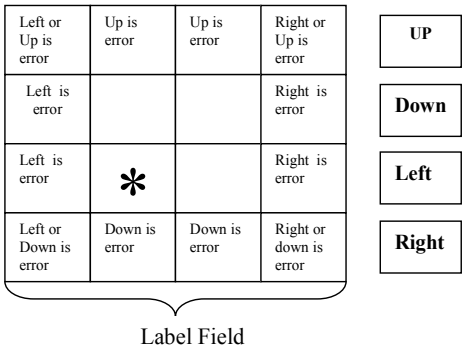
i. Password should be minimum 6 characters containing atleast one uppercase letter, one digit and one symbol.
ii. Confirm password and password fields should match.
iii. The Captcha should generate two random 2 digit numbers and accept the sum from the user.

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

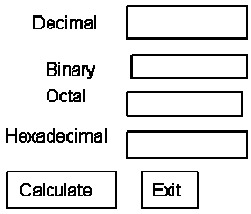


Additional programs for practice

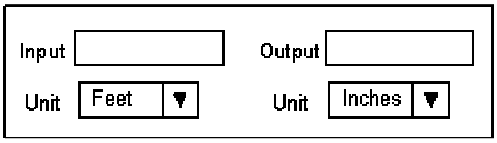
1.Create an application in Java using swing that will move star towards up, down, left and right. Display appropriate message if it crosses the boundary. Design the screen as shown:



2. Create a GUI and program for number conversion from decimal to binary, octal and hexadecimal when the user clicks on "Calculate".



3. Create a conversion applet which accepts value in one unit and converts it to another. The input and output unit is selected from a list. Perform conversion to and from Feet, Inches, Centimeters, Meters and Kilometers.



Signature of the instructor ____ Date ____

Assignment Evaluation

0: Not done ☐ 2: Late Complete ☐ 4: Complete ☐

1: Incomplete ☐ 3: Needs improvement ☐ 5: Well Done ☐