Lab Manual

CSC—241

Object Oriented Programming

Department of Computer Science
Islamabad Campus

CUI

Lab Contents:

This lab emphasizes the concepts of object-oriented techniques used in developing computer-based system. The topics include: Problem solving in Object Oriented Paradigm, Defining classes and objects in JAVA, Controlling access to members- Encapsulation, Passing and returning non-primitive values from methods, Composition/Containership (Has-a relationship), Inheritance, Method Overriding and Abstract Class, Polymorphism, Interfaces, Array List and Generics, File Handling, Graphical User Interface – Layout Managers , Graphical User Interface- Event Driven Programming.

Student Outcomes (SO)

S.#	**Description**
1	Apply knowledge of computing fundamentals, knowledge of a computing specialization, and mathematics, science, and domain knowledge appropriate for the computing specialization to the abstraction and conceptualization of computing models from defined problems and requirements
2	Identify, formulate, research literature, and solve complex computing problems reaching substantiated conclusions using fundamental principles of mathematics, computing sciences, and relevant domain disciplines
3	Design and evaluate solutions for complex computing problems, and design and evaluate systems, components, or processes that meet specified needs with appropriate consideration for public health and safety, cultural, societal, and environmental considerations
4	Create, select, adapt and apply appropriate techniques, resources, and modern computing tools to complex computing activities, with an understanding of the limitations
5	Function effectively as an individual and as a member or leader in diverse teams and in multi-disciplinary settings.
9	Recognize the need, and have the ability, to engage in independent learning for continual development as a computing professional

Intended Learning Outcomes

Sr.#	**Description**	**Blooms Taxonomy Learning Level**	**SO**
CLO -4	Implement a small module utilizing Object-Oriented design.	*Applying*	2-4
CLO-5	Develop a GUI based project for a real-world problem in a team environment.	*Creating*	2-5,9

Lab Assessment Policy

The lab work done by the student is evaluated using Psycho-motor rubrics defined by the course instructor, viva-voce, project work/performance. Marks distribution is as follows:

Assignments	Lab Mid Term Exam	Lab Terminal Exam	Total
25	25	50	100

Note: Midterm and Final term exams must be computer based.

List of Labs

Lab 01
Problem Solving in Object Oriented Paradigm

Objective:

Objective of this lab is to understand the Object-Oriented paradigm.

Activity Outcomes:

The student will be able to understand the Object-oriented paradigm.
The student will be able to understand difference between class and object.

Instructor Note:

As pre-lab activity, read Chapter 9 from the text book "Introduction to Java Programming", Y. Daniel Liang, Pearson, 2019.

1) Useful Concepts

The world around us is made up of objects, such as people, automobiles, buildings, streets, and so forth. Each of these objects has the ability to perform certain actions, and each of these actions has some effect on some of the other objects in the world.

OOP is a programming methodology that views a program as similarly consisting of objects that interact with each other by means of actions.

Object-oriented programming has its own specialized terminology. The objects are called, appropriately enough, objects. The actions that an object can take are called methods. Objects of the same kind are said to have the same type or, more often, are said to be in the same class.

For example, in an airport simulation program, all the simulated airplanes might belong to the same class, probably called the Airplane class. All objects within a class have the same methods. Thus, in a simulation program, all airplanes have the same methods (or possible actions), such as taking off, flying to a specific location, landing, and so forth. However, all simulated airplanes are not identical. They can have different characteristics, which are indicated in the program by associating different data (that is, some different information) with each particular airplane object. For example, the data associated with an airplane object might be two numbers for its speed and altitude.

Things that are called procedures, methods, functions, or subprograms in other languages are all called methods in Java. In Java, all methods (and for that matter, any programming constructs whatsoever) are part of a class.

Syntax :

```
class ClassName {
   //datefileds
   //methods
}
```

2) Solved Lab Activities

Sr.No	***Allocated Time***	***Level of Complexity***	***CLO Mapping***
Activity 1	***20 mins***	***Low***	***CLO-4***
Activity 2	***20 mins***	***Low***	***CLO-4***
Activity 3	***20 mins***	***Low***	***CLO-4***

Activity 1:

Consider the concept of a CourseResult. The CourseResult should have data members like the student's name, course name and grade obtained in that course.
This concept can be represented in a class as follows:

Solution:

```
class CourseResult {

    public String studentname;
    public String coursename;
    public String grade;

    public void display() {
        System.out.println("Student Name is:" +
studentname + "Course Name is:" + coursename
                + "Grade is:" + grade);
    }
}

public class CourseResultRun {

    public static void main(String[] args) {
        CourseResult c1 = new CourseResult();
        c1.studentname = "Ali";
        c1.coursename = "OOP";
        c1.grade = "A";
        c1.display();
        CourseResult c2 = new CourseResult();
        c2.studentname = "Saba";
        c2.coursename = "ICP";
        c2.grade = "A+";
        c2.display();
    }
}
```

Note that both objects; c1 and c2 have three data members, but each object has different values for their data members.

Activity 2:

The example below represents a Date class. As date is composed of three attributes, namely month, year and day; so, the class contains three Data Members. Now every date object will have these three attributes, but each object can have different values for these three

Solution:

```
class Date {

    public String month;
    public int day;
    public int year; //a four digit number.

    public void displayDate() {
        System.out.println(month + " " + day + ", " + year);
    }
}

public class DateDemo {

    public static void main(String[] args) {
        Date date1, date2;
        date1 = new Date();
        date1.month = "December";
        date1.day = 31;
        date1.year = 2012;
        System.out.println("date1:");
        date1.displayDate();
        date2 = new Date();
        date2.month = "July";
        date2.day = 4;
        date2.year = 1776;
        System.out.println("date2:");
        date2.displayDate();
    }
}
```

Activity 3:

Consider the concept of a Car Part. After analyzing this concept we may consider that it can be described by three data members: modelNumber, partNumber and cost.
The methods should facilitate the user to assign values to these data members and show the values for each object.
This concept can be represented in a class as follows:

```
import java.util.Scanner;

class CarPart {

    private String modelNumber;
    private String partNumber;
    private String cost;

    public void setparameter(String x, String y, String z) {
        modelNumber = x;
        partNumber = y;
        cost = z;
    }

    public void display() {
        System.out.println("Model  Number: " + modelNumber + "Part 
Number: " + partNumber+ "Cost: " + cost);
    }
}
public class CarPartRunner {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        CarPart car1 = new CarPart();
        System.out.println("What is Model Number?");
        System.out.println("What is Part Number?");
        System.out.println("What is Cost?");
        String x = sc.nextLine();
        String y = sc.nextLine();
        String z = sc.nextLine();
        car1.setparameter(x, y, z);
        car1.display();
        }
}
```

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficulty and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

A Student is an object in a university management System. Analyze the concept and identify the data members that a Student class should have. Also analyze the behavior of student in a university management System and identify the methods that should be included in Student class.

Lab Task 2

Time is an intangible concept. Analyze the concept and identify the data members and methods that should be included in Time class.

Lab Task 3

Car is an object that helps us in transportation. Analyze the concept and identify the data members and methods that should be included in Car class.

Lab Task 4

Rectangle is an object that represents a specific shape. Analyze the concept and identify the data members and methods that should be included in Rectangle class.

Lab 02
Defining classes and objects in JAVA

Objective:

Objective of this lab is to understand the importance of classes and construction of objects using constructors

Activity Outcomes:

- The student will be able to declare a classes and objects.
- The student will be able to declare member functions and member variables of a class.
- The student will be able to declare overloaded constructors.

Instructor Note:

As pre-lab activity, read Chapter 9 from the text book “Introduction to Java Programming”, Y. Daniel Liang, Pearson, 2019.

1) Useful Concepts

- **Data Abstraction**

  Abstraction is the process of recognizing and focusing on important characteristics of a situation or object and leaving/filtering out the un-wanted characteristics of that situation or object. For example a person will be viewed differently by a doctor and an employer.

  A doctor sees the person as patient. Thus he is interested in name, height, weight, age, blood group, previous or existing diseases etc of a person.

  An employer sees a person as an employee. Therefore, employer is interested in name, age, health, degree of study, work experience etc of a person.

- **Class and Object:**

  The fundamental idea behind object-oriented languages is to combine into a single unit both data and the functions that operate on that data. Such a unit is called an object. A class serves as a plan, or blueprint. It specifies what data and what functions will be included in objects of that class. An object is often called an ―instance‖ of a class.

- **Instance Variables and Methods:**

  Instance variables represent the characteristics of the object and methods represent the behavior of the object. For example length & width are the instance variables of class Rectangle and Calculatearea() is a method.

  Instance variables and methods belong to some class, and are defined inside the class to which they belong.

  Syntax:

```
public class Class_Name
{
Instance_Variable_Declaration_1
Instance_Variable_Declaration_2
. . .
Instance_Variable_Declaration_Last

Method_Definition_1 Method_Definition_2
. . .
Method_Definition_Last
}
```

- **Constructors:**

  It is a special function that is automatically executed when an object of that class is created. It has no return type and has the same name as that of the class. It is normally defined in classes to initialize data members. A constructor with no parameters is called a no-argument constructor. A constructor may contain arguments which can be used for initiation of data members.

Syntax:

```
class_name( )
{
      Public class_name()
      {
      //body
      }

      Public class_name(type var1, type var2)
      {
      //body
      }
}
```

If your class definition contains absolutely no constructor definitions, then Java will automatically create a no-argument constructor. If your class definition contains one or more constructor definitions, then Java does not automatically generate any constructor; in this case, what you define is what you get. Most of the classes you define should include a definition of a no-argument constructor.

2) Solved Lab Activities

Sr.No	***Allocated Time***	***Level of Complexity***	***CLO Mapping***
Activity 1	***10 mins***	***Low***	***CLO-4***
Activity 2	***20 mins***	***Low***	***CLO-4***
Activity 3	***20 mins***	***Low***	***CLO-4***

Activity 1:

The following example shows the declaration of class Rectangle. It has two data members that represent the length and width of rectangle. The method calculateArea will return the area of rectangle. The runner class will create an object of Rectangle class and area function will be called.

Solution:

```
class Rectangle {

    public int length, width;

    public int Calculatearea() {
        return (length * width);
    }
}

public class runner {

    public static void main(String args[]) {
        Rectangle rect = new Rectangle();
        rect.length = 10;
        rect.width = 5;
        System.out.println(rect.Calculatearea());
    }
}
```

Activity 2:

The following example demonstrates the use of constructors

Solution:

```
class Rectangle {

    public int length, width;

    public Rectangle() {
        length = 5;
        width = 2;
    }

    public Rectangle(int l, int w) {
        length = l;
        width = w;
    }

    public int Calculatearea() {
        return (length * width);
    }
}
public class runner {
        public static void main(String args[]) {
        Rectangle rect = new Rectangle();
        System.out.println(rect.Calculatearea());
        Rectangle rect1 = new Rectangle(10, 20);
        System.out.println(rect1.Calculatearea());
    }
}
```

Activity 3:

The following example shows the declaration of class Point. It has two data members that represent the x and y coordinate. Create two constructors and a function to move the point. The runner class will create an object of Point class and move function will be called.

Solution:

```
class Point {

    private int x;
    private int y;

    public Point() {
        x = 1;
        y = 2;
    }

    public Point(int a, int b) {
        x = a;
        y = b;
    }

    public void setX(int a) {
        x = a;
    }

    public void setY(int b) {
        y = b;
    }
    public void display() {
        System.out.println("x coordinate = " + x + " y coordinate = "
+ y);
    }

    public void movePoint(int a, int b) {
        x = x + a;
        y = y + b;
        System.out.println("x coordinate after moving = " + x + " y
coordinate after moving = " + y);
    }

}


public class runner {

       public static void main(String args[]) {

        Point p1 = new Point();
        p1.movePoint(2, 3);

        p1.display();

        Point p2 = new Point();
        p2.movePoint(2, 3);
        p2.display();

    }

}
```

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficulty and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Create a class circle class with radius as data member. Create two constructors (no argument, and two arguments) and a method to calculate Circumference.

Lab Task 2

Create a class Account class with balance as data member. Create two constructors (no argument, and two arguments) and methods to withdraw and deposit balance.

Lab Task 3

Create a class –Distance‖ with two constructors (no argument, and two argument), two data members (feet and inches). Also create display function which displays all data members.

Lab Task 4

Write a class Marks with three data members to store three marks. Create two constructors and a method to calculate and return the sum.

Lab Task 5

Write a class Time with three data members to store hr, min and seconds. Create two constructors and apply checks to set valid time. Also create display function which displays all data members.

Lab 03
Controlling access to class members – Encapsulation

Objective:

The objective of this lab is to teach the students, concept of encapsulation and access modifiers

Activity Outcomes:

At the end of this lab student will be familiar with the accessing rules of class data members and member functions

Instructor Note:

As pre-lab activity, read Chapter 9 from the text book “Introduction to Java Programming”, Y. Daniel Liang, Pearson, 2019.

1) Useful Concepts

Encapsulation

Information hiding means that you separate the description of how to use a class from the implementation details, such as how the class methods are defined. The programmer who uses the class can consider the implementation details as hidden, since he or she does not need to look at them. Information hiding is a way of avoiding information overloading. It keeps the information needed by a programmer using the class within reasonable bounds. Another term for information hiding is abstraction.
Encapsulation means grouping software into a unit in such a way that it is easy to use because there is a well-defined simple interface. So, encapsulation and information hiding are two sides of the same coin.

Access Modifiers

Java allows you to control access to classes, methods, and fields via access modifiers. The access to classes, constructors, methods and fields are regulated using access modifiers i.e. a class can control what information or data can be accessible by other classes. To take advantage of encapsulation, you should minimize access whenever possible.

Syntax:

```
class class_name {
     access_specifie type member1;
      access_specifie type member1;
     ...... .
}
```

The following table describes the access modifiers provided by JAVA.

TABLE I Access Modifiers

Modifier	Description
(no modifier)	member is accessible within its package only
Public	member is accessible from any class of any package

Protected	member is accessible in its class package and by its subclasses
Private	member is accessible only from its class

The accessibility rules are depicted in the following table

TABLE II Accessibility Rules

Protection	Accessed inside Class	Accessed in subclass	Accessed in any Class
Private	Yes	No	No
Protected	Yes	Yes	No
Public	Yes	Yes	yes

Accessors and Mutators

We should always make all instance variables in a class private and should define public methods to provide access to these members. Accessor methods allow you to obtain the data. For example, the method getMonth() returns the number of the month. Mutator methods allow you to change the data in a class object.

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	***10 mins***	***Low***	***CLO-4***
Activity 2	***20 mins***	***Low***	***CLO-4***
Activity 3	***20 mins***	***Low***	***CLO-4***

Activity 1:

The following example shows the declaration of class Circle. It has one data members radius. The data member is declared private and access is provided by declaring set and get methods.

Solution:

```
class Circle {

    private int radius;
    public Circle() {
        radius = 7;
    }

    public Circle(int r) {
        radius = r;
    }

    public void setRadius(int r) {
        radius = r;
    }
    public int getRadius() {
        return radius;
    }
    public void display() {
        System.out.println("radius = " + radius);
    }

    public double CalculateCircumference() {
        double a = 3.14 * radius * radius;
        return a;
    }
}

public class Runner {

    public static void main(String args[]) {
        Circle c1 = new Circle();
        c1.setRadius(5);
        System.out.println("Circumference of
Circle 1 is: " + c1.CalculateCircumference());
        int r = c1.getRadius();
        Circle c2 = new Circle(r);
        c2.setRadius(5);
        System.out.println("Circumference of
Circle 2 is: " + c2.CalculateCircumference());

    }
}
```

Activity 2*:*

The following example shows the declaration of class Rectangle. It has two data members that represent the length and width of rectangle. Both data member are declared private and access is provided by declaring set and get methods for both data members.

Solution:

```
class Rectangle {

    private int length, width;

    public Rectangle() {
        length = 5;
        width = 2;
    }

    public Rectangle(int l, int w) {
        length = l;
        width = w;
    }

    public void setLength(int l) //sets the value
of length
    {
        length = l;
    }

    public void setWidth(int w) //sets the value
of width
    {
        width = w;
    }

    public int getLength() //gets the value of
length
    {
        return length;
    }
    public int getWidth() //gets the value of
width
    {
        return width;
    }

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

public class Runner {

    public static void main() {
        Rectangle rect = new Rectangle();
        rect.setLength(5);
        rect.setWidth(10);
        System.out.println("Area of Rectangle is:
" + rect.area());
        System.out.println("Width of Rectangle
is: " + rect.getWidth());
    }
}
```

Activity 3:

The following example shows the declaration of class Point. It has two data members that represent the x and y coordinate of a point. Both data member are declared private and access is provided by declaring set and get methods for both data members.

Solution:

```
class Point {

    private int x;
    private int y;

    public Point() {
        x = 0;
        y = 0;
    }

    public Point(int a, int b) {
        x = a;
        y = b;
    }

    public void setX(int a) {
        x = a;
    }

    public void setY(int b) {
        y = b;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }
    public void display() {
        System.out.println("x coordinate = " + x
+ " y coordinate = " + y);
    }
    public void movePoint(int a, int b) {
        x = x + a;
        y = y + b;
    } }
public class runner {

    public static void main() {
        Point p1 = new Point();
        p1.setX(10);
        p1.setY(7);
        p1.display();
        Point p2 = new Point(10, 11);
        p2.movePoint(2, 3);
        p2.display();
    }
}
```

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficulty and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Create an Encapsulated class Marks with three data members to store three marks. Create set and get methods for all data members. Test the class in runner

Lab Task 2

Create an Encapsulated class Account class with balance as data member. Create two constructors and methods to withdraw and deposit balance. In the runner create two accounts. The second account should be created with the same balance as first account. (Hint: use get function)

Lab Task 3

Create an Encapsulated class Student with following characteristics:

Data Members:

String Name
Int [] Result_array[5] // Result array contains the marks for 5 subjects

Methods:
Student (String, int[])   // argument Constructor
Average ()   // it calculates and returns the average based on the marks in the array.

Runner:

Create two objects of type Student and call the Average method.

Compare the Average of both Students and display which student has higher average. Create a third student with name as object 1 and result array as object 2

Lab Task 4

Suppose you operate several hot dog stands distributed throughout town. Define an Encapsulated class named HotDogStand that has an instance variable for the hot dog stand's ID number and an instance variable for how many hot dogs the stand has sold that day.
Create a constructor that allows a user of the class to initialize both values. Also create a method named justSold that increments by one the number of hot dogs the stand has sold. The idea is that this method will be invoked each time the stand sells a hot dog so that you can track the total number of hot dogs sold by the stand.
Write a main method to test your class with at least three hot dog stands that each sell a variety of hot dogs. Use get function to display the hot dogs sold for each object.
.

Lab 04
Passing and returning non primitive values from methods

Objective:

The objective of this lab is to teach the students, how the objects can be passed to and returned from the functions.

Activity Outcomes:

After completion of this Lab students will be able to :

- pass objects to methods
- return objects from methods

Instructor Note:

As pre-lab activity, read Chapter 10 from the text book "Introduction to Java Programming", Y. Daniel Liang, Pearson, 2019.

1) Useful Concepts

In Java, all primitives are passed by value. This means a copy of the value is passed into the method. Objects can be passed natively, just like primitives. It is often misstated that Object parameters are passed by Reference. While it is true that the parameter is a reference to an Object, the reference itself is passed by Value.

Syntax for passing objects to method:

```
     public void MethodName (reference_variable_type reference_variable) {

     ......

     }
```

Syntax for returning objects from method:

```
     public reference_variable_type MethodName
     (reference_variable_type reference_variable) {

          ......

          return reference_variable;

          }
```

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	***10 mins***	***Medium***	***CLO-4***
Activity 2	***20 mins***	***Medium***	***CLO-4***
Activity 3	***20 mins***	***Medium***	***CLO-4***

Activity 1:

Passing object as parameter and change value of its data member.

Solution:

```
class ObjectPass {

    public int value;
    public static void
increment(ObjectPass a) {
        a.value++;
    }
}

public class ObjectPassTest {

    public static void main(String[]
args) {
        ObjectPass p = new
ObjectPass();
        p.value = 5;
        System.out.println("Before
calling: " + p.value); // output is 5
        ObjectPass.increment(p);
        System.out.println("After
calling: " + p.value); // output is 6
    }
}
```

Now it is like the pass was by reference! but the thing is what we pass exactly is a handle of an object, and in the called method a new handle created and pointed to the same object. Now when more than one handles tied to the same object, it is known as **aliasing**. This is the default way Java does when passing the handle to a called method, create alias.

Activity 2*:*

The following activity demonstrates the creation of a method that accepts and returns object.

Solution:

```

class Complex {

    private double real;
    private double imag;

    public Complex() {
        real = 0.0;
        imag = 0.0;
    }

    public Complex(double r, double im) {
        real = r;
        imag = im;
    }

    public Complex Add(Complex b) {
        Complex c_new = new Complex(real +
b.real, imag + b.imag);
        return c_new;
    }

    public void Show() {
        System.out.println(real + imag);
    }
}

public class ComplexTest {

    public static void main(String args[]) {
        Complex C1 = new Complex(11, 2.3);
        Complex C2 = new Complex(9, 2.3);
        Complex C3 = new Complex();
        C3 = C1.Add(C2);
        C3.Show();
    }
}
```

Activity 3:

The following activity demonstrates the creation of a method that accepts two objects.

Solution:

```
class Point {

    private int X;
    private int Y;

    public Point() {
        X = 5;
        Y = 6;
    }

    public Point(int a, int c) {
        X = a;
        Y = c;
    }

    public void setX(int a) {
        X = a;
    }

    public void setY(int c) {
        Y = c;
    }

    public int getX() {
        return X;
    }

    public int getY() {
        return Y;
    }

    public Point Add(Point Pa, Point Pb) {
        Point p_new = new Point(X + Pa.X + Pb.X,
Y + Pa.Y + Pb.Y);
        return p_new;
    }

    public void display() {
        System.out.println(X);
        System.out.println(Y);
    }
}

public class PointTest {
    public static void main(String[] args) {
        Point p1 = new Point(10, 20);
        Point p2 = new Point(30, 40);
        Point p3 = new Point();
        Point p4 = p1.Add(p2, p3);
        p4.display();
    }
}
```

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficulty and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Create a class ― Distance‖ with two constructors (no argument, and two argument), two data members (feet and inches) . Create setter, getter and display method. Create a method that adds two Distance Objects and returns the added Distance Object.

Lab Task 2

Create an Encapsulated class Book. Its data members are

- *author (String)*
- *chapterNames[100] (String[])*

Create two overloaded constructors, one with no argument and one with two arguments.

Create a method compareBooks that compares the author of two Books and returns true if both books have same author and false otherwise. (This method must manipulate two Book objects)

Create a method compareChapterNames that compares the chapter names of two Books and returns the book with larger chapters. Display the author of the book with greater chapters in main.

Create a runner class that declares two objects of type Book. One object should be declared using no argument constructor and then the parameters should be set through the set() methods. The second object should be declared with argument constructor. Finally the CompareBooks()and compareChapterNames method should be called and the result should be displayed in the runner class.

Lab Task 3

Define a class called Fraction. This class is used to represent a ratio of two integers. Create two constructors, set, get and display function. Include an additional method, ***equals****, that takes as input another Fraction and returns true if the two fractions are identical and false if they are not.*

Lab 05
Composition / Containership (Has-a relationship)

Objective:

The purpose of lab is to make students understand the concept of has-a relationship an object-oriented programming**.**

Activity Outcomes:

Students will be able to understand that complex objects can be modeled as composition of other objects.

Students will be able to implement programs related to composition.

Instructor Note:

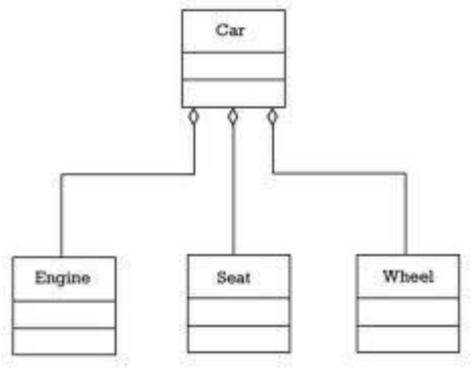
As pre-lab activity, read Chapter 10 from the text book "Introduction to Java Programming", Y. Daniel Liang, Pearson, 2019.

1) Useful Concepts

In real-life, complex objects are often built from smaller, simpler objects. For example, a car is built using a metal frame, an engine, some tires, a transmission, a steering wheel, and a large number of other parts. A personal computer is built from a CPU, a motherboard, some memory, etc. Even you are built from smaller parts: you have a head, a body, some legs, arms, and so on. This process of building complex objects from simpler ones is called composition (also known as object containership).

More specifically, composition is used for objects that have a —has-a relationship to each other. A car has-a metal frame, has-an engine, and has-a transmission. A personal computer has-a CPU, a motherboard, and other components. You have-a head, a body.

```
class wheel
{
}

class car
{
  wheel w1;
}
```



2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	***20 mins***	***Medium***	***CLO-4***
Activity 2	***25 mins***	***Medium***	***CLO-4***

Activity 1:

Composition is about expressing relationships between objects. Think about the example of a manager. A manager has the properties e.g Title and club dues. A manager has an employment record. And a manager has an educational record. The phrase "has a" implies a relationship where the manager owns, or at minimum, uses, another object. It is this "has a" relationship which is the basis for composition. This example can be programmed as follows:

Solution:

```
class studentRecord {

    private String degree;

    public studentRecord() {
    }

    public void setDegree(String deg) {
        degree = deg;

    }

    public String getDegree() {
        return degree;
    }

}

class employeeRecord {

    private int emp_id;
    private double salary;

    public employeeRecord() {
    }

    public void setEmp_id(int id) {
        emp_id = id;
    }

    public int getEmp_id() {
        return emp_id;
    }

    public void setSalary(int sal) {
        salary = sal;
    }
    public double getSalary() {
        return salary;
    }

}

class Manager {

    private String title;
    private double dues;
    private employeeRecord emp;
    private studentRecord stu;

    public Manager(String t, double d,
employeeRecord e, studentRecord s) {
        title = t;
        dues = d;
        emp = e;
        stu = s;
    }

    public void display() {

        System.out.println("Title is : " +
title);
        System.out.println("Dues are : " + dues);

        System.out.println("Emplyoyee record is
as under:");
        System.out.println("EmployeeId is : " +
emp.getEmp_id());
        System.out.println("EmployeeId is : " +
emp.getSalary());

        System.out.println("Student record is as
under: ");

        System.out.println("Degree is : " +
stu.getDegree());
    }

}

public class Runner {

    public static void main(String args[]) {
        studentRecord s = new studentRecord();
        s.setDegree("MBA");
        employeeRecord e = new employeeRecord();
        e.setEmp_id(1);
        e.setSalary(25000);
        Manager m1 = new
Manager("financeManager", 5000, e, s);
        m1.display();
    }
}
```

Activity 2:

The program below represents an employee class which has two Date objects as data members.

Solution:

```
class Date {

    private int day;
    private int month;
    private int year;

    public Date(int theMonth, int theDay, int
theYear) {
        day = checkday(theDay);
        month = checkmonth(theMonth);
        year = theYear;
    }

    private int checkmonth(int testMonth) {
        if (testMonth > 0 && testMonth <= 12) {
            return testMonth;
        } else {
            System.out.println("Invalid month" +
testMonth + "set to 1");
            return 1;
        }
    }

    private int checkday(int testDay) {
        int daysofmonth[] = {0, 31, 28, 31, 30,
31, 30, 31, 31, 30, 31, 30, 31};

        if (testDay > 0 && testDay <=
daysofmonth[month]) {
            return testDay;
        } else if (month == 2 && testDay == 29 &&
(year % 400 == 0 || (year % 4 == 0 && year % 100
                != 0))) {
            return testDay;
        } else {
            System.out.println("Invalid date" +
testDay + "set to 1");
        }
        return 1;
    }

    public int getDay() {
        return day;
    }

    public int getMonth() {
        return month;
    }

    public int getYear() {
        return year;
    }
    public void display() {
        System.out.println(day + " " + month + "
" + year);
    }

}

class employee {

    private String name;
    private String fname;
    private Date birthdate;
    private Date hiredate;

    employee() {

    }

    employee(String x, String y, Date
birthofDate, Date dateofHire) {
        name = x;
        fname = y;
        birthdate = birthofDate;
        hiredate = dateofHire;

    }

    public void setname(String x) {
        name = x;
    }

    public String getname() {
        return name;

    }

    public void setfname(String x) {
        fname = x;
    }
    public String getfname() {
        return fname;
    }

    public void setbirthdate(Date b) {
        birthdate = b;

    }

    public Date getbirthdate() {
        return birthdate;

    }

    public void sethiredate(Date h) {
        hiredate = h;
    }

    public Date gethiredate() {
        return hiredate;
    }

    public void display() {

        System.out.println("Name: " + name + "
Father Name: " + fname);
        birthdate.display();
        hiredate.display();

    }
}

public class Employrun {

    public static void main(String[] args) {
        Date b = new Date(1, 12, 1990);
        Date h = new Date(5, 6, 2016);
        employee e1 = new employee("xxx", "yyyy",
b, h);
          e1.display();
     }

}
```

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficulty and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Create an Address class, which contains street#, house#, city and code. Create another class Person that contains an address of type Address. Give appropriate get and set functions for both classes. Test class person in main.

Lab Task 2

Create a class Book that contains an author of type Person (Note: Use the Person class created in the first exercise). Other data members are bookName and publisher. Modify the address of the author in runner class.

Lab Task 3

Design a class Point with two data members x-cord and y-cord. This class should have an arguments constructor, setters, getters and a display function.
Now create another class ―Line‖, which contains two Points ―startPoint‖ and ―endPoint‖. It should have a function that finds the length of the line.

Hint: formula is: sqrt((x2-x1)2 + (y2-y1)2)

Create two line objects in runner and display the length of each line.

Lab Task 4

Create a class named Pizza that stores information about a single pizza. It should contain the following:

Private instance variables to store the size of the pizza (either small, medium, or large), the number of cheese toppings, the number of pepperoni toppings, and the number of ham toppings.
Constructor(s) that set all of the instance variables.
Public methods to get and set the instance variables.
A public method named calcCost( ) that returns a double that is the cost of the pizza. Pizza cost is determined by:
Small: $10 + $2 per topping Medium: $12 + $2 per topping Large: $14 + $2 per topping

public method named getDescription( ) that returns a String containing the pizza size, quantity of each topping.

Write test code to create several pizzas and output their descriptions. For example, a large pizza with one cheese, one pepperoni and two ham toppings should cost a total of $22.
Now Create a PizzaOrder class that allows up to three pizzas to be saved in an order. Each pizza saved should be a Pizza object. Create a method calcTotal() that returns the cost of order.
In the runner order two pizzas and return the total cost.

Lab 06

Inheritance

Objective:

The objective of this lab is to familiarize the students with various concepts and terminologies of inheritance using Java.

Activity Outcomes:

This lab teaches you the following topics:

- Declaration of the derived classes along with the way to access of base class members.
- Protected Access modifier and working with derived class constructors.

Instructor Note:

As pre-lab activity, read Chapter 11 from the text book "Introduction to Java Programming", Y. Daniel Liang, Pearson, 2019.

1) Useful Concepts

a. Inheritance

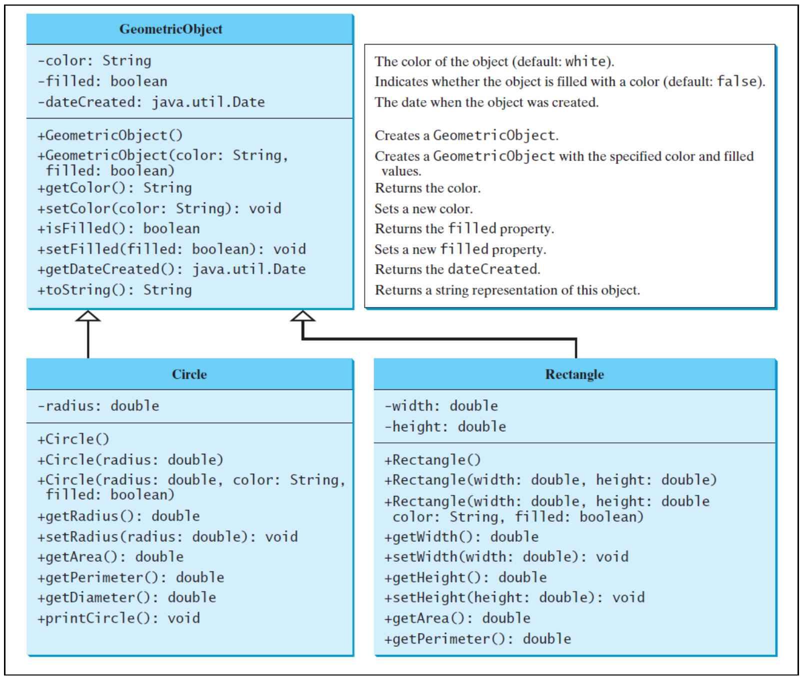
Inheritance is a way of creating a new class by starting with an existing class and adding new members. The new class can replace or extend the functionality of the existing class. The existing class is called the base class and the new class is called the derived class.

b. Protected Access Specifier

Protected members are directly accessible by derived classes but not by other users. A class member labeled **protected** is accessible to member functions of derived classes as well as to member functions of the same class.

c. Derived class constructor

Constructors are not inherited, even though they have public visibility. However, the super reference can be used within the child's constructor to call the parent's constructor. In that case, the call to parent's constructor must be the first statement.


GeometricObject
-color: String
-filled: boolean
-dateCreated: java.util.Date
+GeometricObject()
+GeometricObject(color: String, filled: boolean)
+getColor(): String
+setColor(color: String): void
+isFilled(): boolean
+setFilled(filled: boolean): void
+getDateCreated(): java.util.Date
+toString(): String
The color of the object (default: white).
Indicates whether the object is filled with a color (default: false).
The date when the object was created.
Creates a GeometricObject.
Creates a GeometricObject with the specified color and filled values.
Returns the color.
Sets a new color.
Returns the filled property.
Sets a new filled property.
Returns the dateCreated.
Returns a string representation of this object.
Circle
-radius: double
+Circle()
+Circle(radius: double)
+Circle(radius: double, color: String, filled: boolean)
+getRadius(): double
+setRadius(radius: double): void
+getArea(): double
+getPerimeter(): double
+getDiameter(): double
+printCircle(): void
Rectangle
-width: double
-height: double
+Rectangle()
+Rectangle(width: double, height: double)
+Rectangle(width: double, height: double color: String, filled: boolean)
+getWidth(): double
+setWidth(width: double): void
+getHeight(): double
+setHeight(height: double): void
+getArea(): double
+getPerimeter(): double

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	*25 mins*	*Medium*	*CLO-4*
Activity 2	*25 mins*	*Medium*	*CLO-4*

Activity 1:

This example will explain the method to specify the derived class. It explains the syntax for writing the constructor of derived class.

Solution:

```
public class person {

protected String name ; protected String id ; protected int phone ;

public person() {
name = "NaginaNazar" ; id = "sp14bcs039" ; phone = 12345 ;
}

public person(String a , String b , int c)
{ name = a ; id = b ; phone = c ;}

 public void setName(String a){ name = a ;}

public void setId(String j){id = j ;}

public void setPhone(int a) { phone = a ;}

public String getName() {return name ;}

public String getid() {return id ;}

public int getPhone() {return phone ;}

public void display( )
{
```

```
System.out.println("Name : " + name + "ID : " + id + "Phone : " +
phone ) ;}}
---------------------------------------------------------------------
public class student extends person {
private String rollNo;
private int marks ;

public student() {
super() ;
rollNo = "sp14bcs039" ; marks = 345 ;
}
public student(String a , String b , int c , String d , int e)
{ super(a,b,c) ;
rollNo = d ; marks = e ;
}

public void setRollNo(String a){ rollNo = a ;}

public void setMarks(int a ){ marks = a ;}

public String getRollNo() { return rollNo ;}

public int getMarks() {return marks ;}

public void display( )
{
super.display();
System.out.println("Roll # : " + rollNo + "\nMarks : " + marks) ;
}
}

---------------------------------------------------------------------
public class Runner
{
public static void main(String []args)
{
student s = new student ("Ahmed", "s-09", 123, "sp16-bcs-98",50);
s.display();
} }
```

Output

```
Name : Ahmed

ID : s-09

Phone : 123

Roll # : sp16-bcs-98

Marks : 50
```

Activity 2:

This example demonstrates another scenario of inheritance. The super class can be extended by more than one class.

```
public class Employee {

protected String name;
protected String phone;
protected String address;
protected int allowance;

public Employee(String name, String phone, String address, int
allowance)
{
this.name = name; this.phone = phone; this.address = address;
this.allowance = allowance;
}
}
--------------------------------------------------------------------
public class Regular extends Employee
{
private int basicPay;

public Regular(String name, String phone, String address, int
allowance, int basicPay)
{
super(name, phone, address, allowance);
this.basicPay = basicPay;
}
public void Display(){
System.out.println("Name: " + name + "Phone Number: " + phone
+"Address: " + address + "Allowance:  " + allowance + "Basic Pay:  "
+ basicPay);
}
}
----------------------------------------------------------------------
public class Adhoc extends Employee
{
private int numberOfWorkingDays; private int wage;

public Adhoc(String   name, String     phone,     String     address,
int  allowance, int numberOfWorkingDays, int wage)
{
super(name, phone, address, allowance);
this.numberOfWorkingDays = numberOfWorkingDays;
this.wage = wage;
}

public void Display()
{
System.out.println("Name: " + name + "Phone Number: " + phone
+"Address: " + address +   "Allowance:     "    +   allowance   +
"Number    Of   Working   Days: " + numberOfWorkingDays + "Wage: " +
wage);
}
}
----------------------------------------------------------------------
public class Runner
{
public static void main(String []args){
Regular regularObj = new
Regular("Ahmed","090078601","Islamabad",15000,60000);
regularObj.Display();
Adhoc adhocObj = new
Adhoc("Ali","03333333333","Rawalpindi",500,23,1500);
adhocObj.Display();
}
}
```

Output

```
Name: Ahmed Phone Number: 090078601 Address: Islamabad
Allowance:  15000Basic Pay:  60000

Name: Ali Phone Number: 03333333333 Address: Rawalpindi
Allowance:      500 Number Of Working Days: 23 Wage: 1500
```

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficulty and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

(The Person, Student, Employee, Faculty, and Staff classes)
Design a class named Person and its two subclasses named Student and Employee. Design two more classes; Faculty and Staff and extend them from Employee. The detail of classes is as under:

A person has a name, address, phone number, and email address.
A student has a status (String)
An employee has an office, salary, and date hired. Use the Date class to create an object for date hired.
A faculty member has office hours and a rank.
A staff member has a title.
Create display method in each class

Lab Task 2

Imagine a publishing company that markets both book and audio-cassette versions of its works. Create a class publication that stores the title and price of a publication. From this class derive two classes:
i. book, which adds a page count and
ii. tape, which adds a playing time in minutes.
Each of these three classes should have set() and get() functions and a display() function to display its data. Write a main() program to test the book and tape class by creating instances of them, asking the user to fill in their data and then displaying the data with display().

Lab Task 3

Write a base class Computer that contains data members of wordsize(in bits), memorysize (in megabytes), storagesize (in megabytes) and speed (in megahertz). Derive a Laptop class that is a kind of computer but also specifies the object's length, width, height, and weight. Member functions for both classes should include a default constructor, a constructor to inialize all components and a function to display data members.r.

Lab 07

Method Overriding and Abstract Classes

Objective:

The objective of this lab is to familiarize the students with various concepts and terminologies of method overriding and concept of abstract classes.

Activity Outcomes:

This lab teaches you the following topics:

- Method overriding where a base class method version is redefined in the child class with exact method signatures.
- Abstract classes along with the access of base class members.

Instructor Note:

As pre-lab activity, read Chapter 11 from the text book “Introduction to Java Programming”, Y. Daniel Liang, Pearson, 2019.

1) Useful Concepts

a. Method Overriding

The definition of an inherited method can be changed in the definition of a derived class so that it has a meaning in the derived class that is different from what it is in the base class. This is called overriding the definition of the inherited method.

For example, the methods toString and equals are overridden (redefined) in the definition of the derived class HourlyEmployee. They are also overridden in the class SalariedEmployee. To override a method definition, simply give the new definition of the method in the class definition, just as you would with a method that is added in the derived class.

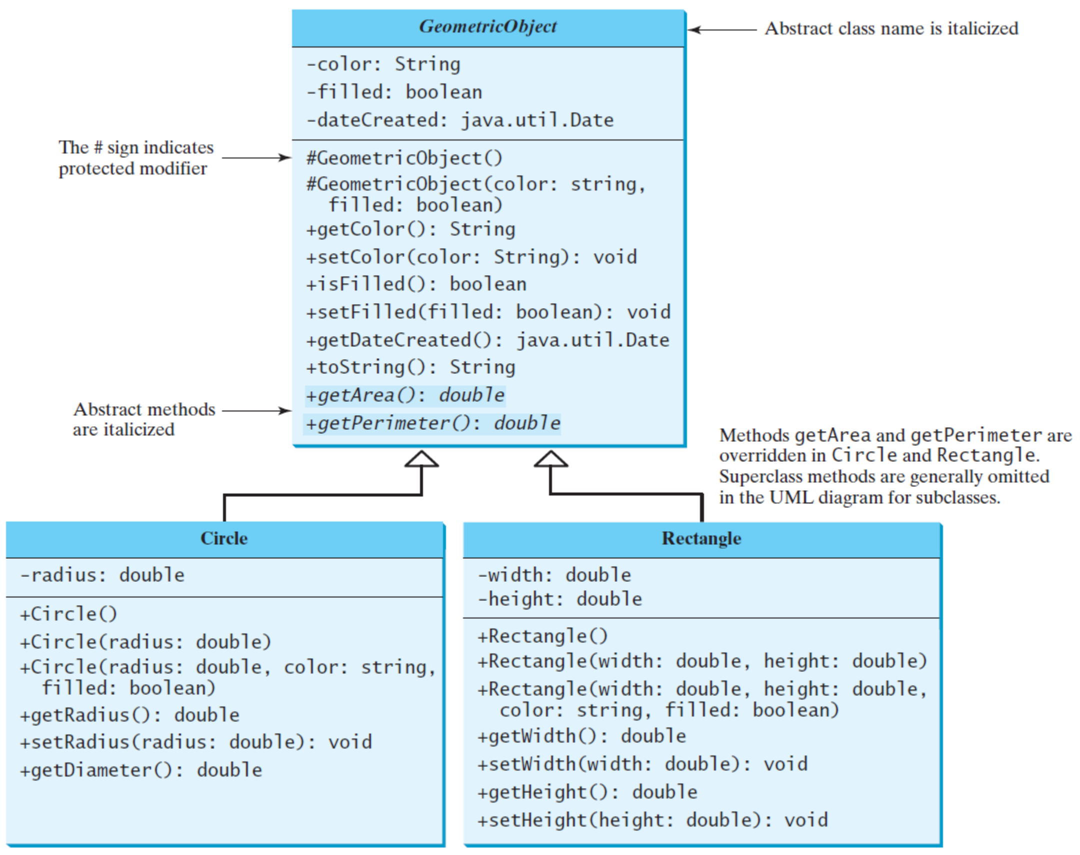
In a derived class, you can override (change) the definition of a method from the base class. As a general rule, when overriding a method definition, you may not change the type returned by the method, and you may not change a void method to a method that returns a value, nor a method that returns a value to a void method. The one exception to this rule is if the returned type is a class type, then you may change the returned type to that of any descendent class of the returned type. For example, if a function returns the type Employee, when you override the function definition in a derived class, you may change the returned type to HourlyEmployee, SalariedEmployee, or any other descendent class of the class Employee. This sort of changed return type is known as a covariant return type and is new in Java version 5.0; it was not allowed in earlier versions of Java.

b. Abstract Class

A class that has at least one abstract method is called an abstract class and, in Java, must have the modifier abstract added to the class heading. An abstract class can have any number of abstract methods. In addition, it can have, and typically does have, other regular (fully defined) methods. If a derived class of an abstract class does not give full definitions to all the abstract methods, or if the derived class adds an abstract method, then the derived class is also an abstract class and must include the modifier abstract in its heading.

In contrast with the term abstract class, a class with no abstract methods is called a concrete class.

```
public abstract class GeometricObject
{
private instanceVariables;
. . .
public abstract double getArea();
public abstract double getPerimeter();
. . .
}
```



GeometricObject
-color: String
-filled: boolean
-dateCreated: java.util.Date
#GeometricObject()
#GeometricObject(color: string, filled: boolean)
+getColor(): String
+setColor(color: String): void
+isFilled(): boolean
+setFilled(filled: boolean): void
+getDateCreated(): java.util.Date
+toString(): String
+getArea(): double
+getPerimeter(): double
Abstract class name is italicized
The # sign indicates protected modifier
Abstract methods are italicized
Methods getArea and getPerimeter are overridden in Circle and Rectangle. Superclass methods are generally omitted in the UML diagram for subclasses.
Circle
-radius: double
+Circle()
+Circle(radius: double)
+Circle(radius: double, color: string, filled: boolean)
+getRadius(): double
+setRadius(radius: double): void
+getDiameter(): double
Rectangle
-width: double
-height: double
+Rectangle()
+Rectangle(width: double, height: double)
+Rectangle(width: double, height: double, color: string, filled: boolean)
+getWidth(): double
+setWidth(width: double): void
+getHeight(): double
+setHeight(height: double): void

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	***10 mins***	***Medium***	***CLO-4***
Activity 2	***20 mins***	***Medium***	***CLO-4***
Activity 2	***20 mins***	***Medium***	***CLO-4***

Activity 1:

The following activity demonstrates the creation of overridden methods.

Solution:

```
class A
{
int i, j;

A(int a, int b) { i = a; j = b; }
// display i and j
void show() {
System.out.println("i and j: " + i + " " + j);
}
}
-------------------------------------------------------------------
class B extends A
{
int k;
B(int a, int b, int c) { super(a, b); k = c; }

// display k – this overrides show() in A
void show() {
System.out.println("k: " + k);
}
}
--------------------------------------------------------------------
Public class OverrideRunner
{
public static void main(String args[])
{
B subOb = new B(1, 2, 3); subOb.show(); // this calls show() in B
}}
```

Output

```
k: 3
```

Activity 2:

The following activity explains the use of overriding for customizing the method of super class. The classes below include a CommisionEmployee class that has attributes of firstname, lastName, SSN, grossSales, CommisionRate. It has a constructor to initialize, set and get functions, and a function to display data members. The other class BasePlusCommisionEmployee is inherited from CommisionEmployee. It has additional attributes of Salary. It also has set and get functions and display function. The Earning method is overridden in this example.

Solution:

```
public class commissionEmployee
{
```

```
protected String FirstName; protected String LastName; protected
String SSN; protected double grossSales; protected double commonRate;

public commissionEmployee()
{
FirstName="Nagina"; LastName="Nazar"; SSN="S003";
grossSales=1234.1; commonRate=12.5;
}

public commissionEmployee (String a,String e,String b, double c,
double d){ FirstName=a;
LastName=e; SSN=b;
grossSales=c; commonRate=d;
}

public void setFN(String a){ FirstName=a;}
public void setLN(String e){ LastName=e;}
public void setSSN(String b){ SSN=b;}
public void setGS(double c){ grossSales=c;}
public void setCR(double d){ commonRate=d;}
public String getFN(){return FirstName;}
public String getSSN(){return SSN;}
public double getGS(){return grossSales;}
public double getCR(){return commonRate;}

public double earnings(){
return grossSales*commonRate;
}

public void display(){
System.out.println("first name:"+FirstName+"last name:"
+LastName+"SSN:"+SSN+" Gross Sale:"+grossSales+" and
commonRate:"+commonRate);
}
}
--------------------------------------------------------------------
public class BasePlusCommEmployee extends commissionEmployee
{
private double salary;
```

```
BasePlusCommEmployee(){ salary=48000; }

BasePlusCommEmployee(String A,String E,String B, double C, double D,
double S)
{
super(A,E,B,C,D);
salary=S;
}
//overridden method
public double earnings()
{
return super.earnings()+salary;
}

public void display(){
super.display();
System.out.println("Salary : "+salary);
}
}

Public class OverrideRunner
{
public static void main(String args[])
{
BasePlusCommEmployee b = new BasePlusCommEmployee("ali", "ahmed",
"25-kkn", 100, 5.2, 25000);
double earn = b.earnings();

System.out.println("Earning of employee is " + earn);
}
}
```

Output

```
Earning of employee is 25520.0
```

Activity 3:

A Simple demonstration of abstract.

```
public abstract class A {
abstract void callme();
// concrete methods are still allowed in abstract classes
void callmetoo() {
System.out.println("This is a concrete method.");
} }
---------------------------------------------------------------------
public  class B extends A {
void callme() {
System.out.println("B's implementation of callme.");
} }
---------------------------------------------------------------------
public class AbstractDemo {
public static void main(String[] args) {
B b = new B();
b.callme(); b.callmetoo(); } }
```

Output

B’s implementation of Call me. This is a concrete method

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficulty and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Create a class named Movie that can be used with your video rental business. The Movie class should track the Motion Picture Association of America (MPAA) rating (e.g., Rated G, PG-13, R), ID Number, and movie title with appropriate accessor and mutator methods. Also create an equals() method that overrides Object ’s equals() method, where two movies are equal if their ID number is identical. Next, create three additional classes named Action , Comedy , and Drama that are derived from Movie . Finally, create an overridden method named calcLateFees that takes as input the number of days a movie is late and returns the late fee for that movie. The default late fee is $2/day. Action movies have a late fee of $3/day, comedies are $2.50/day, and dramas are $2/day. Test your classes from a main method.

Lab Task 2

Write a program that declares two classes. The parent class is called Simple that has two data members num1 and num2 to store two numbers. It also has four member functions.

The add() function adds two numbers and displays the result. The sub() function subtracts two numbers and displays the result.
The mul() function multiplies two numbers and displays the result. The div() function divides two numbers and displays the result.

The child class is called VerifiedSimple that overrides all four functions. Each function in the child class checks the value of data members. It calls the corresponding member function in the parent class if the values are greater than 0. Otherwise it displays error message.

Lab Task 3

Create an abstract class that stores data about the shapes e.g. Number of Lines in a Shape, Pen Color, Fill Color and an abstract method draw. Implement the method draw for Circle, Square and Triangle subclasses, the better approach is to draw these figures on screen, if you can't then just use a display message in the draw function.

Lab 8

Polymorphism

Objective:

The purpose of lab is to make students understand writing of generic code using polymorphism.

Activity Outcomes:

This lab teaches you the following topics:

- Call methods of class using polymorphism
- Upcasting and Downcasting of objects.

Instructor Note:

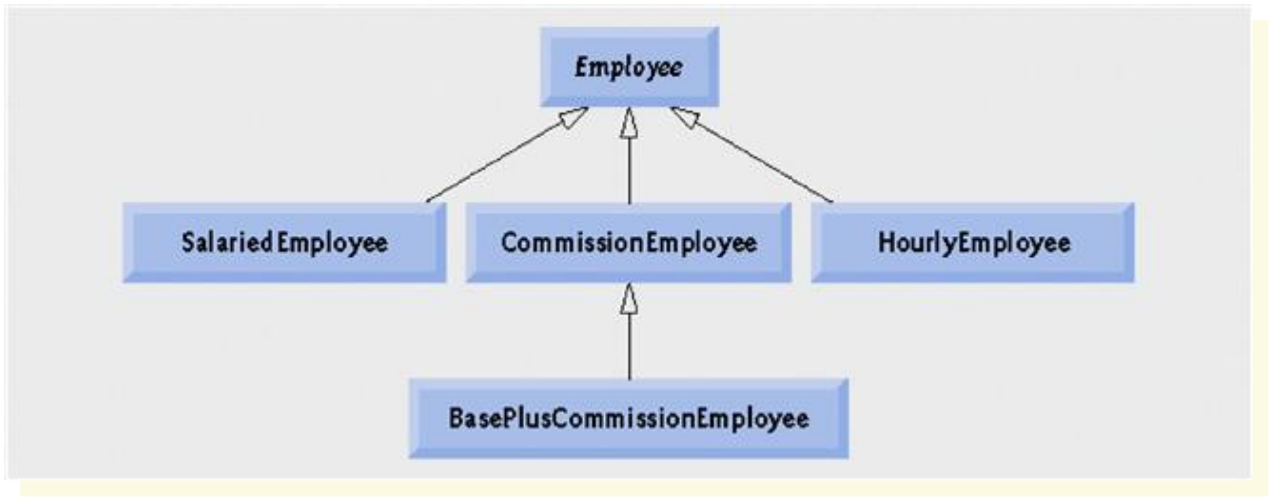
As pre-lab activity, read Chapter 11 from the text book "Introduction to Java Programming", Y. Daniel Liang, Pearson, 2019.

1) Useful Concepts

Polymorphism enables you to "program in the general" rather than "program in the specific." In particular, polymorphism enables you to write programs that process objects that share the same superclass (either directly or indirectly) as if they're all objects of the superclass; this can simplify programming.

Consider the following example of polymorphism. Suppose we create a program that simulates the movement of several types of animals for a biological study. Classes Fish, Frog and Bird represent the types of animals under investigation. Imagine that each class extends superclass Animal, which contains a method move and maintains an animal's current location as x-y coordinates. Each subclass implements method move. Our program maintains an Animal array containing references to objects of the various Animal subclasses. To simulate the animals' movements, the program sends each object the same message once per second—namely, move. Each specific type of Animal responds to a move message in its own way—a Fish might swim three feet, a Frog might jump five feet and a Bird might fly ten feet. Each object knows how to modify its x-y coordinates appropriately for its specific type of movement.

Relying on each object to know how to "do the right thing" (i.e., do what is appropriate for that type of object) in response to the same method call is the key concept of polymorphism. The same message (in this case, move) sent to a variety of objects has "many forms" of results—hence the term polymorphism



```
Employee [] employees = new Employee [4];
employees[0] = new CommisionEmployee();
employees[1] = new SalariedEmployee();
employees[2] = new HourlyEmployee();
employees[3] = new BaseCommEmployee();
```

```
for ( int i=0; i<=3;i++)
{
System.out.printf(employees[i].earnngs());
}
```

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	***25 mins***	***Medium***	***CLO-4***
Activity 2	***25 mins***	***Medium***	***CLO-4***

Activity 1:

The following example demonstrates polymorphism for a overridden method.

Solution:

```
public abstract class Employee
{
private String firstName;
private String lastName;
private String socialSecurityNumber;

public Employee( String first, String last, String ssn )
{
firstName = first;
lastName = last;
socialSecurityNumber = ssn;
}
public String toString()
{
return String.format( "%s %s\nsocial security number: %s",
firstName, lastName, socialSecurityNumber );
} // end method toString
public abstract double earnings();
}
--------------------------------------------------------------------
Public class SalariedEmployee extends Employee
{
private double weeklySalary;
```

```
// four-argument constructor
public SalariedEmployee( String first, String last, String ssn,double
salary )
{
super( first, last, ssn ); // pass to Employee constructor
weeklySalary = salary ;
}

@Override
public double earnings()
{
return weeklySalary;
}
}
----------------------------------------------------------------------
public class HourlyEmployee extends Employee
{
private double wage; // wage per hour
private double hours; // hours worked for week

// five-argument constructor
public HourlyEmployee( String first, String last, String ssn,double
hourlyWage, double hoursWorked )
{
super( first, last, ssn );
wage  = hourlyWage;
hours  = hoursWorked;
}

@Override
public double earnings()
{
if (hours  <= 40 ) // no overtime
return wage * hours ;
else
return 40 * wage + (hours - 40 ) * wage * 1.5;
} }
----------------------------------------------------------------------
public class CommissionEmployee extends Employee
{
```

```
private double grossSales; // gross weekly sales
private double commissionRate; // commission percentage

// five-argument constructor
public CommissionEmployee( String first, String last, String
ssn,double sales, double rate )
{
super( first, last, ssn );
grossSales =sales ;
commissionRate  =rate;
}

@Override
public double earnings()
{
return commissionRate  * grossSales ;
}
}
---------------------------------------------------------------------
public class BasePlusCommissionEmployee extends CommissionEmployee
{
private double baseSalary; // base salary per week

public BasePlusCommissionEmployee( String first, String last,String
ssn, double sales, double rate, double salary )
{
super( first, last, ssn, sales, rate );
baseSalary = salary; // validate and store base salary
}

public void setBaseSalary(double baseSalary)
{this.baseSalary = baseSalary;}
public double getBaseSalary() {  return baseSalary;   }
```

```
@Override
public double earnings()
{
return baseSalary + super.earnings();
} }
```

```
-----------------------------------------------------------------------
public class PayrollSystemTest

{

public static void main( String[] args )

{

SalariedEmployee salariedEmployee = new SalariedEmployee ("John",
"Smith", "111-11-1111", 800.00 );

HourlyEmployee hourlyEmployee= new HourlyEmployee( "Karen", "Price",
"222-22-2222", 16.75, 40 );

CommissionEmployee commissionEmployee = new CommissionEmployee(
"Sue", "Jones", "333-33-3333", 10000, .06 );

BasePlusCommissionEmployee basePlusCommissionEmployee = new
BasePlusCommissionEmployee("Bob", "Lewis", "444-44-4444", 5000, .04,
300 );

Employee[] employees = new Employee[ 4 ];

employees[ 0 ] = salariedEmployee;

employees[ 1 ] = hourlyEmployee;

employees[ 2 ] = commissionEmployee;

employees[ 3 ] = basePlusCommissionEmployee;

for (int i=0; i<4 ;i++)

{

System.out.println(employees[i].earnings()); //polymorphic call

} } }
```

Output

```
800.0

670.0

600.0

500.0
```

Activity 2:

The following example demonstrates downcasting

```
public class PayrollSystemTest

{

public static void main( String[] args )

{

SalariedEmployee salariedEmployee = new SalariedEmployee ("John",
"Smith", "111-11-1111", 800.00 );

HourlyEmployee hourlyEmployee= new HourlyEmployee( "Karen", "Price",
"222-22-2222", 16.75, 40 );

CommissionEmployee commissionEmployee = new CommissionEmployee(
"Sue", "Jones", "333-33-3333", 10000, .06 );

BasePlusCommissionEmployee basePlusCommissionEmployee = new
BasePlusCommissionEmployee("Bob", "Lewis", "444-44-4444", 5000, .04,
300 );

Employee[] employees = new Employee[ 4 ];

// initialize array with Employees

employees[ 0 ] = salariedEmployee;

employees[ 1 ] = hourlyEmployee;

employees[ 2 ] = commissionEmployee;

employees[ 3 ] = basePlusCommissionEmployee;

for (int i=0; i<4 ;i++)

{

if (employees [i] instanceof BasePlusCommissionEmployee)

{

BasePlusCommissionEmployee emp= (BasePlusCommissionEmployee )
employees[i];

emp.setBaseSalary( 1.10 * emp.getBaseSalary() );
System.out.println("New base salary with 10 percent increase is " +
emp.getBaseSalary() );

employees[i]=emp;

 }

System.out.println("Earning is " + employees[i].earnings());

} //end for

}

}
```

Output

```
Earning is 800.0

Earning is 670.0

Earning is 600.0

New base salary with 10 percent increase is 330.0

Earning is 530.0
```

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficulty and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Package-delivery services, offer a number of different shipping options, each with specific costs associated.

Create an inheritance hierarchy to represent various types of packages. Use Package as the super class of the hierarchy, then include classes TwoDayPackage and OvernightPackage that derive from Package.

Super class Package should include data members representing the name and address for both the sender and the recipient of the package, in addition to data members that store the weight (in ounces) and cost per ounce to ship the package. Package's constructor should initialize these data members. Ensure that the weight and cost per ounce contain positive values.

Package should provide a public member function calculateCost() that returns a double indicating the cost associated with shipping the package. Package's calculateCost() function should determine the cost by multiplying the weight by the cost per ounce.

Derived class TwoDayPackage should inherit the functionality of base class Package, but also include a data member that represents a flat fee that the shipping company charges for two-day-delivery service. TwoDayPackage'sconstructor should receive a value to initialize this data member. TwoDayPackage should redefine member function calculateCost() so that it computes the shipping cost by adding the flat fee to the cost calculated by base class Package's calculateCost() function.

Class OvernightPackage should inherit from class Package and contain an additional data member representing an additionalfee charged for overnight-delivery service.
OvernightPackage should redefine member function calculateCost() so that it computes the shipping cost by adding the additionalfee to the cost calculated by base class Package's calculateCost() function.

Write a test program that creates objects of each type of Package and tests member function calculateCost() using polymorphism .

Lab Task 2

Create an abstract class "Person", with data member "name". Create set and get methods, and an abstract Boolean method "isOutstanding()".

Derive two classes Student and Professor. Student class has data member CGPA.

Professor Class has data member numberOfPublications. Provide setters and getters and implementation of abstract function in both classes.

In student class isOutstanding() will return true if CGPA is greater than 3.5. In the Professor class isOutstanding() will return true, if numberOfPublications> 50.

In the main class create an array of Person class and call isOutstanding() function for student and professor. isOutstanding() for professor should be called after setting the publication count to 100.

Lab Task 3

Create a class hierarchy that performs conversions from one system of units to another. Your program should perform the following conversions,
i. Liters to Gallons, ii. Fahrenheit to Celsius and iii. Feet to Meters

The Super class convert declares two variables, val1 and val2, which hold the initial and converted values, respectively. It contains an abstract function "compute()".

The function that will actually perform the conversion, compute() must be defined by the classes derived from convert. The specific nature of compute() will be determined by what type of conversion is taking place.

Three classes will be derived from convert to perform conversions of Liters to Gallons (l_to_g), Fahrenheit to Celsius (f_to_c) and Feet to Meters (f_to_m), respectively. Each derived class implements compute() in its own way to perform the desired conversion.

Test these classes from main() to demonstrate that even though the actual conversion differs between l_to_g, f_to_c, and f_to_m, the interface remains constant.

Lab 9

Interfaces

Objective:

Objective of this lab is to learn how to define Interface and implement Interfaces. Students will also learn how to implement multiple interfaces in one class.

Activity Outcomes:

This lab teaches you the following topics:

- Interface.
- Method overriding with interfaces.
- Implementation of multiple interfaces in class.
- Use of interfaces with abstract classes

Instructor Note:

As pre-lab activity, read Chapter 13 from the text book “Introduction to Java Programming”, Y. Daniel Liang, Pearson, 2019.

1) Useful Concepts

a. Interface

An interface is a type that groups together a number of different classes that all include method definitions for a common set of method headings.
Java interface specifies a set of methods that any class that implements the interface must have. An interface is itself a type, which allows you to define methods with parameters of an interface type and then have the code apply to all classes that implement the interface. One way to view an interface is as an extreme form of an abstract class. However, as you will see, an interface allows you to do more than an abstract class allows you to do. Interfaces are Java's way of approximating multiple inheritance. You cannot have multiple base classes in Java, but interfaces allow you to approximate the power of multiple base classes.

Defining interface :

```
public interface Ordered {
public boolean precedes(Object other);

}
```

Implement interface and its method:

```
public class Class_name implements Interface_Name
{
public boolean precedes(Object other){ }
}

public class implements SomeInterface, AnotherInterface{}
```

b. Defining Constants in Interfaces

The designers of Java often used the interface mechanism to take care of a number of miscellaneous details that do not really fit the spirit of what an interface is supposed to be. One example of this is the use of an interface as a way to name a group of defined constants. An interface can contain defined constants as well as method headings, or instead of method headings. When a method implements the interface, it automatically gets the defined constants. For example, the following interface defines constants for months:

```
     public interface MonthNumbers {

         public static final int JANUARY = 1, FEBRUARY
         = 2,MARCH = 3, APRIL = 4, MAY  =  5,    JUNE
```

```
= 6, JULY = 7, AUGUST = 8, SEPTEMBER = 9,
OCTOBER = 10, NOVEMBER = 11, DECEMBER = 12; }
```

Any class that implements the MonthNumbers interface will automatically have the 2 constants defined in the MonthNumbers interface. For example, consider the following toy class:

```
public class DemoMonthNumbers
implements MonthNumbers
{ public static void
main(String[] args)               {
        System.out.println( "The number for January is " +
        JANUARY);                                         }
        }
```

c. The Comparable Interface

The Comparable interface is in the java.lang package and so is automatically available to your program. The Comparable interface has only the following method heading that must be given a definition for a class to implement the Comparable interface:

```
public int compareTo(Object other);
```

The method compareTo should return a negative number if the calling object “comes before” the parameter other, a zero if the calling object “equals” the parameter other, and a positive number if the calling object “comes after” the parameter other. The “comes before” ordering that underlies compareTo should be a total ordering. Most normal ordering, such as less-than ordering on numbers and lexicographic ordering on strings, is total ordering.

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	***15 mins***	***Medium***	***CLO-4***
Activity 2	***25 mins***	***Medium***	***CLO-4***

Activity 1:

Declare a Interface, RegisterForExams that contains single method register, implements the interface in two different classes (a) Student (b) Employee. Write down a Test Application to test the overridden method.

Solution:

```
public interface RegisterForExams {
public void register();
}
---------------------------------------------------------------------
public class EmplayeeTask implements RegisterForExams{

private String name; private String date; private int salary;

public EmplayeeTask()
{
name = null; date = null; salary = 0;
}

public EmplayeeTask(String name,String date,int salary)
{
this.name = name; this.date = date; this.salary = salary;
}

@Override
public void register() {
System.out.println("Employee is registered " + "Name " + name + "
salary " + salary + " date " + date);
}
}
-------------------------------------------------------------------
public class StudentTask implements RegisterForExams{
private String name; private int age; private double gpa;

public StudentTask()
{

name = null; age = 0;
gpa = 0;
}
public StudentTask(String name,int age,double gpa)
{
this.name = name;
this.age = age;

this.gpa = gpa;

}
@Override
public void register() {
System.out.println("Student is registered  " + "Student name " + name
+ " gpa " + gpa);
}}
----------------------------------------------------------------------
------
public class Runner {
public static void main(String[] args) {

EmplayeeTask e = new EmplayeeTask("Ahmed","11,02,2001",20000);
StudentTask s = new StudentTask("Ali",22,3.5);
e.register();
s.register();
}  }
```

Output

```
Employee is registered Name Ahmed salary 20000 date 11,02,2001
Student is registered Student name Ali gpa 3.5
```

Activity 2:

An Example that shows How to create your own interface and implement it in abstract class

```
interface I1 {

void methodI1(); // public static by default
}
----------------------------------------------------------------------
interface I2 extends I1 {

void methodI2(); // public static by default
}
```

```
----------------------------------------------------------------------
class A1 {
public String methodA1() {
String strA1 = "I am in methodC1 of class A1"; return strA1;
}
public String toString() {
return "toString() method of class A1";
}
}
----------------------------------------------------------------------
class B1 extends A1 implements I2 {

public void methodI1() {
System.out.println("I am in methodI1 of class B1");
}
public void methodI2() {
System.out.println("I am in methodI2 of class B1");
}
}
----------------------------------------------------------------------
class C1 implements I2 {

public void methodI1() {
System.out.println("I am in methodI1 of class C1");
}
public void methodI2() {
System.out.println("I am in methodI2 of class C1");
}
}

// Note that the class is declared as abstract as it does not
// satisfy the interface contract

abstract class D1 implements I2 {

public void methodI1() {
}
// This class does not implement methodI2() hence declared abstract.
}
----------------------------------------------------------------------
```

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

I1 i1 = new B1();
i1.methodI1(); // OK as methodI1 is present in B1
// i1.methodI2(); Compilation error as methodI2 not present in I1
// Casting to convert the type of the reference from type I1 to type
I2 ((I2) i1).methodI2();
I2 i2 = new B1();
i2.methodI1(); // OK
i2.methodI2(); // OK
// Does not Compile as methodA1() not present in interface reference
I1
// String var = i1.methodA1();
// Hence I1 requires a cast to invoke methodA1
String var2 = ((A1) i1).methodA1(); System.out.println("var2 : " +
var2);
String var3 = ((B1) i1).methodA1(); System.out.println("var3 : " +
var3);

String var4 = i1.toString();
System.out.println("var4 : " + var4);
String var5 = i2.toString();
System.out.println("var5 : " + var5);
I1 i3 = new C1();
String var6 = i3.toString();
System.out.println("var6 : " + var6); // It prints the Object
toString() method
Object o1 = new B1();
// o1.methodI1(); does not compile as Object class does not define
// methodI1()
// To solve the probelm we need to downcast o1 reference. We can do
it
// in the following 4 ways
((I1) o1).methodI1(); // 1
((I2) o1).methodI1(); // 2
((B1) o1).methodI1(); // 3
/*
*
*     B1 does not have any relationship with C1 except they are
"siblings".
*
*     Well, you can't cast siblings into one another.
*
*/
//  ((C1)o1).methodI1(); Produces a ClassCastException
}
}
```

Output

```
I am in methodI1 of class B1

I am in methodI1 of class B1

I am in methodI2 of class B1

var2 : I am in methodC1 of class A1

var3 : I am in methodC1 of class A1

var4 : toString() method of class A1

var5 : toString() method of class A1

var6 : javaapplication12.C1@15db9742

I am in methodI1 of class B1

I am in methodI1 of class B1

I am in methodI1 of class B1
```

Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficulty and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

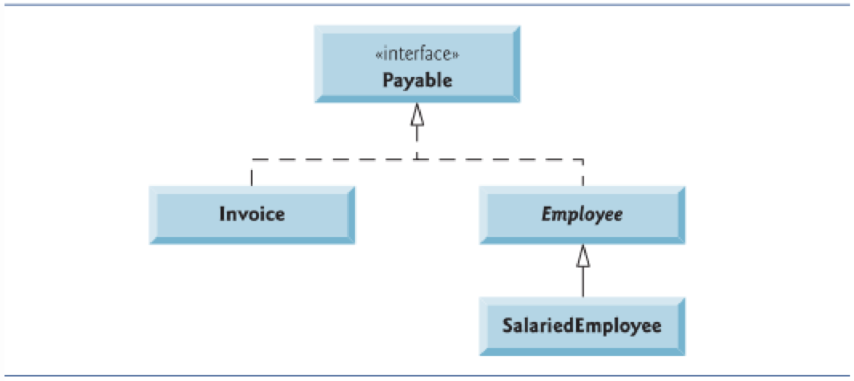
Lab Task 1

Public interface Shape
{
double getArea();
}

Create two classes Circle and Rectangle. Both must implement the interface Shape.
Note: You can assume appropriate data members for circle and rectangle

Lab Task 2

Implement the following hierarchy



Payable:

Double getPaymenyAmount();

Invoice:

```
private String partNumber;
private String partDescription;
private int quantity;
private double pricePerItem;
```

Employee:

```
private String firstName;
private String lastName;
private String socialSecurityNumber;
```

Salaried Employee:

```
private double weeklySalary;
```

In the runner , call the getPaymentAmount() method polymorphically.

Lab Task 3

Below is the skeleton for a class named "InventoryItem" . Each inventory item has a name and a unique ID number:
class InventoryItem
{
private String name;
private int uniqueItemID;
}

Flesh out the class with appropriate accessors, constructors, and mutatators. This class will implement the following interface:

```
Public interface compare
{
            boolean compareObjects(Object o);
}
```

Lab Task 4

Below is a code skeleton for an interface called "Enumeration" and a class called "NameCollection ". Enumeration provides an interface to sequentially iterate through some type of collection. In this case, the collection will be the class NameCollection that simply stores a collection of names using an array of strings.

```
interface Enumeration
{
// return true if a value exists in the next index
public boolean hasNext(int index);

// returns the next element in the collection as an Object
public Object getNext(int index);

}
//NameCollection implements a collection of names using  a simple array.
 class NameCollection
{
  String[] names = new String[100];
 }
```

Create constructor and abstract methods of interface in the class NameCollection.
Then write a main method that creates a NamesCollection object with a sample array of strings, and then iterates through the enumeration outputting each name using the getNext() method .

Lab 10

Array List and Generics

Objective:

Objective of this lab is to explore the Generic enable types (classes and interfaces) to be parameters when defining classes, interfaces and methods along with ArrayList data structure

Activity Outcomes:

This lab teaches you the following topics:

- Using Array Lists.
- Using Generics

Instructor Note:

As pre-lab activity, read Chapter 19 from the text book “Introduction to Java Programming”, Y. Daniel Liang, Pearson, 2019.

1) Useful Concepts

a. Generics:

At its core, the term generics means parameterized types. Parameterized types are important because they enable you to create classes, interfaces, and methods in which the type of data upon which they operate is specified as a parameter. Using generics, it is possible to create a single class; that automatically works with different types of data.

A class, interface, or method that operates on a parameterized type is called generic, as in generic class or generic method. It is important to understand that Java has always given you the ability to create generalized classes, interfaces, and methods by operating through references of type Object. Because Object is the super class of all other classes, an Object reference can refer to any type object. Thus, in pre-generics code, generalized classes, interfaces, and methods used Object references to operate on various types of objects. The problem was that they could not do so with type safety.

Generics add the type safety that was lacking. They also streamline the process, because it is no longer necessary to explicitly employ casts to translate between Object and the type of data that is actually being operated upon. With generics, all casts are automatic and implicit. Thus, generics expand your ability to reuse code and let you do so safely and easily.

Non-Generic Box Class
Begin by examining a non-generic Box class that operates on objects of any type. It needs only to provide two methods: set, which adds an object to the box, and get, which retrieves it:

```
public class Box { private Object object;

public void set(Object object) { this.object = object; } public Object get() { return object; }
}
```

Since its methods accept or return an Object, you are free to pass in whatever you want, provided that it is not one of the primitive types. There is no way to verify, at compile time, how the class is used. One part of the code may place an Integer in the box and expect to get Integers out of it, while another part of the code may mistakenly pass in a String, resulting in a runtime error.

A Generic Version of the Box Class

A generic class is defined with the following format:

```
class name<T1, T2, ..., Tn> { /* ... */ }
```

The type parameter section, delimited by angle brackets (<>), follows the class name. It specifies the type parameters (also called type variables) T1, T2, ..., and Tn.

To update the Box class to use generics, you create a generic type declaration by changing the code "public class Box" to "public class Box<T>". This introduces the type variable, T, that can be used anywhere inside the class.

With this change, the Box class becomes:

```
/**
*	Generic version of the Box class.
*	@param<T> the type of the value being boxed
*/
public class Box<T> {
// T stands for "Type" private T t;

public void set(T t) { this.t = t; } public T get() { return t; }
}
```

As you can see, all occurrences of Object are replaced by T. A type variable can be any non- primitive type you specify: any class type, any interface type, any array type, or even another type variable.

This same technique can be applied to create generic interfaces.

Type Parameter Naming Conventions

By convention, type parameter names are single, uppercase letters. This stands in sharp contrast to the variable naming conventions that you already know about, and with good reason: Without this convention, it would be difficult to tell the difference between a type variable and an ordinary class or interface name.

The most commonly used type parameter names are:

E - Element (used extensively by the Java Collections Framework) K - Key
N - Number T - Type
V - Value
S,U,V etc. - 2nd, 3rd, 4th types

Invoking and Instantiating a Generic Type

To reference the generic Box class from within your code, you must perform a generic type invocation, which replaces T with some concrete value, such as Integer:

```
Box <Integer> integerBox;
```

You can think of a generic type invocation as being similar to an ordinary method invocation, but instead of passing an argument to a method, you are passing a type argument — Integer in this case — to the Box class itself.

Like any other variable declaration, this code does not actually create a new Box object. It simply declares that integerBox will hold a reference to a "Box of Integer", which is how Box<Integer> is read.

An invocation of a generic type is generally known as a parameterized type.

To instantiate this class, use the new keyword, as usual, but place <Integer> between the class name and the parenthesis:

Box <Integer> integerBox = new Box <Integer> ();

Diamond Operator

In Java SE 7 and later, you can replace the type arguments required to invoke the constructor of a generic class with an empty set of type arguments (<> - referred as diamond operator) as long as the compiler can determine, or infer, the type arguments from the context. This pair of angle brackets, <>, is informally called the diamond. For example, you can create an instance of Box<Integer> with the following statement:

Box <Integer> integerBox = new Box <> ();

b. Java Arraylist:

The Java ArrayList is a dynamic array-like data structure that can grow or shrink in size during the execution of a program as elements are added/deleted. An Array on the other hand, has a fixed size: once we declared it to be a particular size, that size cannot be changed. To use an ArrayList, you first have to import the class:

import java.util.ArrayList;

You can then create a new ArrayList object:

ArrayListlistTest = new ArrayList( );

The Java API has a list of all the methods provided by an ArrayList.

2) Solved Lab Activites

Sr.No	***Allocated Time***	***Level of Complexity***	***CLO Mapping***
Activity 1	***10 mins***	***Medium***	***CLO-4***
Activity 2	***25 mins***	***High***	***CLO-4***
Activity 3	***25 mins***	***High***	***CLO-4***

Activity 1:

The following program shows a simple use of ArrayList. An array list is created, and then objects of type String are added to it. The list is then displayed. Some of the elements are removed and the list is displayed again.

Solution:

```
importjava.util.*;
class ArrayListDemo {
public static void main(String args[]) {
ArrayList al = new ArrayList();
System.out.println("Initial size of al: " + al.size());
// add elements to the array list
al.add("C");
al.add("A");
al.add("E");
al.add("B");
al.add("D");
al.add("F");
al.add(1, "A2");

System.out.println("Size of al after additions: " + al.size());
// display the array list
System.out.println("Contents of al: " + al);
// Remove elements from the array list al.remove("F");
al.remove(2);
System.out.println("Size of al after deletions: " + al.size());
System.out.println("Contents of al: " + al);}}
```

Output

```
Initial size of al: 0

Size of al after additions: 7

Contents of al: [C, A2, A, E, B, D, F] Size of al after deletions: 5

Contents of al: [C, A2, E, B, D]
```

Activity 2:

The following program defines two classes. The first is the generic class Gen, and the second is GenDemo, which uses Gen.

```
// A simple generic class.
// Here, T is a type parameter that
// will be replaced by a real type
// when an object of type Gen is created.

class Gen<T> {

T ob; // declare an object of type T

// Pass the constructor a reference to an object of type T.
Gen(T o) {
ob = o;
}

// Return ob.
T getob()
{ return ob;
}

// Show type of T.
Void showType() {
System.out.println("Type of T is " + ob.getClass().getName());}
}
```

```
// Demonstrate the generic class.
Class GenDemo {
public static void main(String args[]) {
// Create a Gen reference for Integers.
Gen<Integer> iOb;
// Create a Gen<Integer> object and assign its reference to iOb.
// Notice the use of autoboxing
// to encapsulate the value 88 within an Integer object.
iOb = new Gen<Integer>(88);
// Show the type of data used by iOb.
iOb.showType();
// Get the value in iOb. Notice that
// no cast is needed.
int v = iOb.getob();
System.out.println("value: " + v);
System.out.println();
// Create a Gen object for Strings.
Gen<String> strOb = new Gen<String>("Generics Test");
// Show the type of data used by strOb.
strOb.showType();
// Get the value of strOb. Again, notice that no cast is needed.
String str = strOb.getob();
System.out.println("value: " + str);}}
```

Output

```
Type of T is java.lang.Integer
value: 88
Type of T is java.lang.String
value: Generics Test
```

Activity 3:

You can declare more than one type parameter in a generic type. To specify two or more type parameters, simply use a comma-separated list. For example, the following TwoGenclass is a variation of the Gen class that has two type parameters:

```
// A simple generic class with two type
// parameters: T and V.
class TwoGen<T, V>
{
T ob1;
V ob2;
// Pass the constructor a reference to
// an object of type T and an object of type V.

TwoGen(T o1, V o2) { ob1 = o1; ob2 = o2;}
// Show types of T and V.
void showTypes()
{
System.out.println("Type of T is " + ob1.getClass().getName());
System.out.println("Type of V is " + ob2.getClass().getName());}
T getob1() { return ob1; }

V getob2() { return ob2;}
}
---------------------------------------------------------------------
// Demonstrate TwoGen.
Class SimpGen {
public static void main(String args[]) {
TwoGen<Integer, String> tgObj = new TwoGen  <Integer, String>(88,
"Generics");
// Show the types.
tgObj.showTypes();
// Obtain and show values.
int v = tgObj.getob1();
System.out.println("value: " + v);
String str = tgObj.getob2();
System.out.println("value: " + str);}
}
```

Output

```
Type of T is java.lang.Integer

Type of V is java.lang.String

value: 88

value: Generics
```

Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficulty and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Write a program that uses an ArrayList of parameter type Contact to store a database of contacts. The Contact class should store the contact's first and last name, phone number, and email address. Add appropriate accessor and mutator methods. Your database program should present a menu that allows the user to add a contact, display all contacts, search for a specific contact and display it, or search for a specific contact and give the user the option to delete it. The searches should find any contact where any instance variable contains a target search string. For example, if "elmore" is the search target, then any *contact where the first name, last name, phone number, or email address contains “elmore” should be returned for display or deletion. Use the “for-each” loop to iterate through the ArrayList*

Lab Task 2

Write a generic class, MyMathClass , with a type parameter T where T is a numeric object type (e.g., Integer, Double, or any class that extends java.lang.Number). Add a method named standardDeviation that takes an ArrayList of type T and returns as a double the standard deviation of the values in the ArrayList . Use the doubleValue () method in the Number class to retrieve the value of each number as a double. Refer to Programming Project 6.5 for a definition of computing the standard deviation. Test your method with suitable data. Your program should generate a compile-time error if your standard deviation method is invoked on an ArrayList that is defined for nonnumeric elements (e.g., Strings).

Lab Task 3

Create a generic class with a type parameter that simulates drawing an item at random out of a box. This class could be used for simulating a random drawing. For example, the box might contain Strings representing names written on a slip of paper, or the box might contain Integers representing a random drawing for a lottery based on numeric lottery picks.

Create an add method that allows the user of the class to add an object of the specified type along with an isEmpty method that determines whether or not the box is empty. Finally, your class should have a drawItem method that randomly selects an object from the box and returns it.
If the user attempts to drawn an item out of an empty box, return null . Write a main method that tests your class.

Lab Task 4

In the sport of diving, seven judges award a score between 0 and 10, where each score may be a floating-point value. The highest and lowest scores are thrown out and the remaining scores are added together. The sum is then multiplied by the degree of difficulty for that dive. The degree of difficulty ranges from 1.2 to 3.8 points. The total is then multiplied by 0.6 to determine the diver’s score.

Write a computer program that inputs a degree of difficulty and seven judges’ scores and outputs the overall score for that dive. The program should use an ArrayList of type Double to store the scores.

Lab 11

File Handling

Objective:

The purpose of lab is to make students understand ways of getting information in and out of your Java programs, using files.

Activity Outcomes:

This lab teaches you the following topics:

- Students will be able to retrieve create file.
- Students will be able to write and read data to and from a file.
- Students will learn about Object Serialization.

Instructor Note:

As pre-lab activity, read Chapter 17 from the text book “Introduction to Java Programming”, Y. Daniel Liang, Pearson, 2019.

1)Useful Concepts

To read an entire object from or write an entire object to a file, Java provides object serialization. A serialized object is represented as a sequence of bytes that includes the object's data and its type information. After a serialized object has been written into a file, it can be read from the file and deserialized to recreate the object in memory.

A class that implements the Serializable interface is said to be a serializable class. To use objects of a class with writeObject() and readObject() , that class must be serializable. But to make the class serializable, we change nothing in the class. All we do is add the phrase implements Serializable . This phrase tells the run-time system that it is OK to treat objects of the class in a particular way when doing file I/O.

Classes ObjectInputStream and ObjectOutputStream, which respectively implement the ObjectInput and ObjectOutput interfaces, enable entire objects to be read from or written to a stream.

To use serialization with files, initialize ObjectInputStream and ObjectOutputStream objects with FileInputStream and FileOutputStream objects

ObjectOutput interface method writeObject takes an Object as an argument and writes its information to an OutputStream.

A class that implements ObjectOuput (such as ObjectOutputStream) declares this method and ensures that the object being output implements Serializable.

ObjectInput interface method readObject reads and returns a reference to an Object from an InputStream. After an object has been read, its reference can be cast to the object's actual type.

. /**
Demonstrates binary file I/O of serializable class objects.
*/

```
public class ObjectIODemo
{
public static void main(String[] args)
{
try
{
ObjectOutputStreamoutputStream =
new ObjectOutputStream(new FileOutputStream("datafile"));
SomeClass oneObject = new SomeClass(1, 'A');
SomeClassan otherObject = new SomeClass(42, 'Z');
outputStream.writeObject(oneObject);
outputStream.writeObject(anotherObject);
outputStream.close();
System.out.println("Data sent to file.");
}
catch(IOException e)
{
System.out.println("Problem with file output.");
}
System.out.println(
"Now let's reopen the file and display the data.");

try
{
ObjectInputStreaminputStream =
new ObjectInputStream(new FileInputStream("datafile"));
Notice the type casts.
SomeClassreadOne = (SomeClass)inputStream.readObject( );
SomeClassreadTwo = (SomeClass)inputStream.readObject( );
System.out.println("The following were read from the
file:"); System.out.println(readOne);
System.out.println(readTwo);
}
catch(FileNotFoundException e)
{
System.out.println("Cannot find datafile.");
}
catch(ClassNotFoundException e)
{
System.out.println("Problems with file input.");
}

catch(IOException e)
{
System.out.println("Problems with file input.");
}
System.out.println("End of program.");
}
}
```

2) Solved Lab Activities

Sr.No	***Allocated Time***	***Level of Complexity***	***CLO Mapping***
Activity 1	***25 mins***	***Medium***	***CLO-4***
Activity 2	***25 mins***	***High***	***CLO-4***

Activity 1:

The following example demonstrates writing of objects to a file.

Solution:

```
import java.io.*;

public  class Person implements Serializable
{
public String name = null;
public int age = 0;

public void setAge(int a) { age = a ;}

public String getName() {return name ;}

}
----------------------------------------------------------------------
import java.io.*;
public class ObjectOutputStreamExample {

public void writeToFile() {

try
{
ObjectOutputStream objectOutputStream =
new ObjectOutputStream(new FileOutputStream("filename"));

Person p = new Person();
p.name = "Jakob Jenkov"; p.age = 40;

objectOutputStream.writeObject(p);
}
catch (FileNotFoundException ex)
{ ex.printStackTrace();
}
catch (IOException ex)
{ ex.printStackTrace();
} } }
```

Activity 2:

The following example demonstrates reading of all objects from a file.

```
import java.io.*;

public class ObjectInputStreamExample {

public void readFromFile()

{

try

{

ObjectInputStream objectInputStream = new ObjectInputStream(new
FileInputStream("filename"));

while (true)

{

Person personRead = (Person) objectInputStream.readObject();

System.out.println(personRead.name);

System.out.println(personRead.age);

}

}

catch (EOFException ex) { //This exception will be caught when EOF is
reached System.out.println("End of file reached.");

} catch (ClassNotFoundException ex) { ex.printStackTrace();

} catch (FileNotFoundException ex) { ex.printStackTrace();

} catch (IOException ex) { ex.printStackTrace();}}}
```

Output

```
Jakob Jenkov

40
```

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficulty and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Create a class Book that has name(String), publisher (String) and an author (Person). Write five objects of Book Class in a file named "BookStore".

Lab Task 2

Consider the Book class of Activity 1 and write a function that displays all Books present in file "BookStore".

Lab Task 3

Consider the Book class of Activity 1 and write a function that asks the user for the name of a Book and searches the record against that book in the file "BookStore".

Lab Task 4

Create an ATM System with Account as the Serializable class. Write ten objects of Account in a file. Now write functions for withdraw, deposit, transfer and balance inquiry.

Note:

a. Each function should ask for the account number on which specific operation should be made.

b. All changes in Account object should be effectively represented in the file.

Lab 12

Graphical User Interface - Layout Managers

Objective:

The purpose of lab is to make students understand basic concepts of Layouts of GUI in Java. The students will learn about the three basic types of layouts and understand the difference between them.

Activity Outcomes:

Students will be able to create frames with different layouts. Students will be able to create simple to medium level complex GUI

Instructor Note:

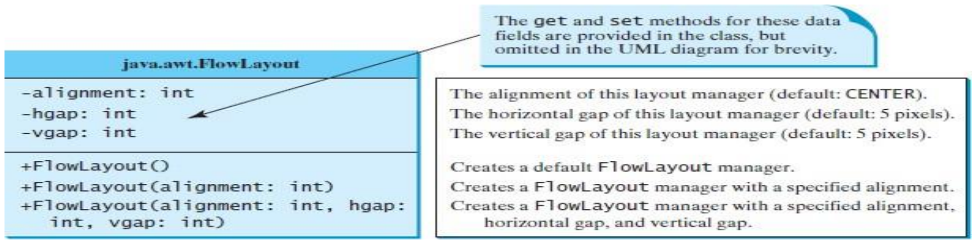
As pre-lab activity, read Chapter 15 from the text book "Introduction to Java Programming", Y. Daniel Liang, Pearson, 2019.

1) Useful Concepts

In many other window systems, the user-interface components are arranged by using hardcoded pixel measurements. For example, put a button at location (10, 10) in the window using hard-coded pixel measurements, the user interface might look fine on one system but be unusable on another. Java's layout managers provide a level of abstraction that automatically maps your user interface on all window systems.
The Java GUI components are placed in containers, where they are arranged by the container's layout manager. In the preceding program, you did not specify where to place the OK button in the frame, but Java knows where to place it, because the layout manager works behind the scenes to place components in the correct locations. A layout manager is created using a layout manager class.
Layout managers are set in containers using the SetLayout(aLayoutManager) method. For example, you can use the following statements to create an instance of XLayout and set it in a container:
LayoutManagerlayoutManager = new XLayout(); container.setLayout(layoutManager);

FlowLayout

FlowLayout is the simplest layout manager. The components are arranged in the container from left to right in the order in which they were added. When one row is filled, a new row is started. You can specify the way the components are aligned by using one of three constants: FlowLayout.RIGHT, FlowLayout.CENTER, or FlowLayout.LEFT. You can also specify the gap between components in pixels. The class diagram for FlowLayout is shown in Figure below

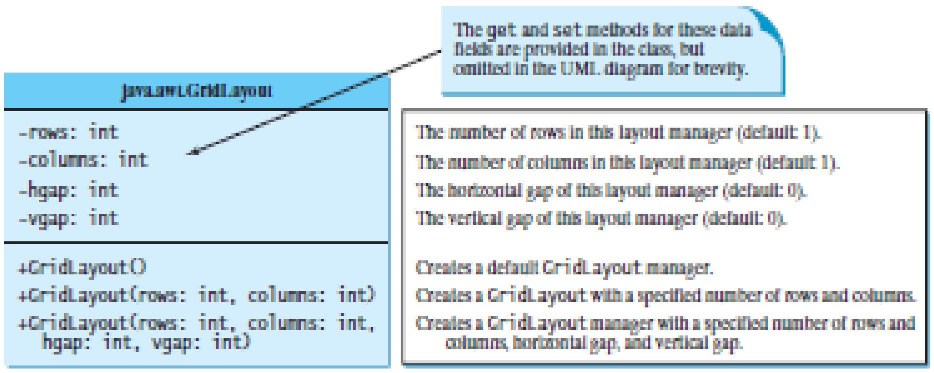


The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
java.awt.FlowLayout
-alignment: int
-hgap: int
-vgap: int
+FlowLayout()
+FlowLayout(alignment: int)
+FlowLayout(alignment: int, hgap: int, vgap: int)
The alignment of this layout manager (default: CENTER).
The horizontal gap of this layout manager (default: 5 pixels).
The vertical gap of this layout manager (default: 5 pixels).
Creates a default FlowLayout manager.
Creates a FlowLayout manager with a specified alignment.
Creates a FlowLayout manager with a specified alignment, horizontal gap, and vertical gap.

Figure 1 Flow Layout

Grid Layout

The GridLayout manager arranges components in a grid (matrix) formation. The components are placed in the grid from left to right, starting with the first row, then the second, and so on, in the order in which they are added. The class diagram for GridLayout is shown in Figure below.



The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
java.awt.GridLayout
-rows: int
-columns: int
-hgap: int
-vgap: int
+GridLayout()
+GridLayout(rows: int, columns: int)
+GridLayout(rows: int, columns: int, hgap: int, vgap: int)
The number of rows in this layout manager (default: 1).
The number of columns in this layout manager (default: 1).
The horizontal gap of this layout manager (default: 0).
The vertical gap of this layout manager (default: 0).
Creates a default GridLayout manager.
Creates a GridLayout with a specified number of rows and columns.
Creates a GridLayout manager with a specified number of rows and columns, horizontal gap, and vertical gap.

Figure 2 Grid Layout

BorderLayout

The BorderLayout manager divides a container into five areas: East, South, West, North, and Center. Components are added to a BorderLayout by using add(Component, index), where index is a constant as mentioned below:

- BorderLayout.EAST,
- BorderLayout.SOUTH,
- BorderLayout.WEST,
- BorderLayout.NORTH,
- BorderLayout.CENTER.

The class diagram for BorderLayout is shown in Figure below:

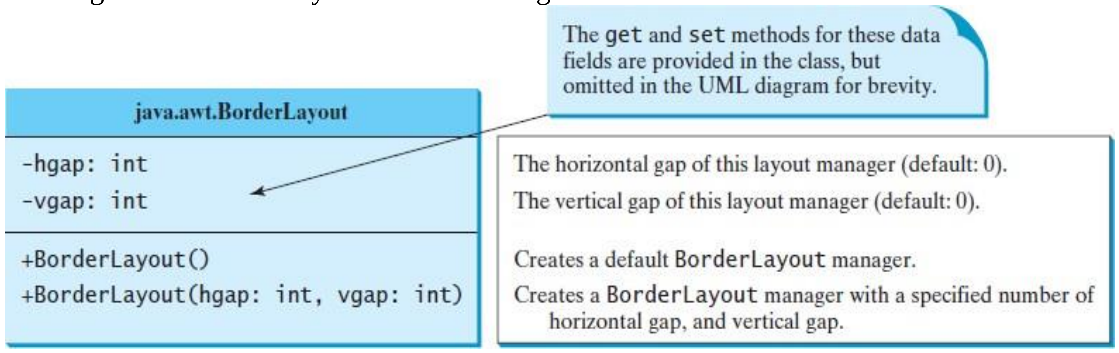

The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
java.awt.BorderLayout
-hgap: int
-vgap: int
+BorderLayout()
+BorderLayout(hgap: int, vgap: int)
The horizontal gap of this layout manager (default: 0).
The vertical gap of this layout manager (default: 0).
Creates a default BorderLayout manager.
Creates a BorderLayout manager with a specified number of horizontal gap, and vertical gap.

Figure 3 Border Layout

Panels as SubContainers

Suppose that you want to place ten buttons and a text field in a frame. The buttons are placed in grid formation, but the text field is placed on a separate row. It is difficult to achieve the desired look by placing all the components in a single container. With Java GUI programming, you can divide a window into panels. Panels act as subcontainers to group user-interface components.
You add the buttons in one panel, then add the panel into the frame. The Swing version of panel is JPanel. You can use new JPanel() to create a panel with a default FlowLayout manager or new JPanel(LayoutManager) to create a panel with the specified layout manager.
Use the add(Component) method to add a component to the panel. For example, the following code creates a panel and adds a button to it:
JPanel p = new JPanel(); p.add(new JButton("OK"));
Panels can be placed inside a frame or inside another panel. The following statement places panel p into frame f:

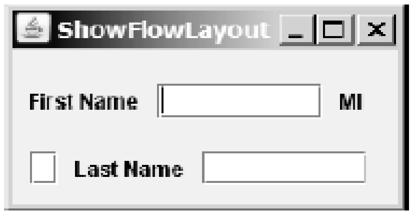
f.add(p);

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	***15 mins***	***High***	***CLO-4***
Activity 2	***15 mins***	***High***	***CLO-4***
Activity 3	***15 mins***	***High***	***CLO-4***
Activity 4	***15 mins***	***High***	***CLO-4***

Activity 1:

Create the following frame using Flow Layout.

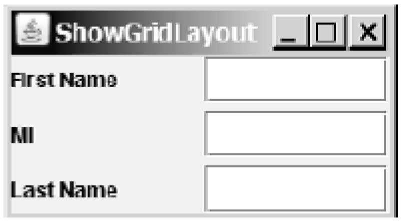


Solution:

```
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JFrame;
import java.awt.FlowLayout;

public class ShowFlowLayout extends JFrame {

    public ShowFlowLayout() {
        //Set FlowLayout, aligned left with
horizontal gap 10
        //and vertical gap 20 between components
        setLayout(new FlowLayout(FlowLayout.LEFT,
10, 20));
        add(new JLabel("First Name"));
        add(new JTextField(8));
        add(new JLabel("MI"));
        add(new JTextField(1));
        add(new JLabel("Last Name"));
        add(new JTextField(8));

    }
    /**
     * Main method
     */
    public static void main(String[] args) {
        ShowFlowLayout frame = new
ShowFlowLayout();
        frame.setTitle("Show FlowLayout");
        frame.setSize(200, 200);
        frame.setLocationRelativeTo(null);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLO
SE);
        frame.setVisible(true);
    }
}
```

Activity 2:

Create the following frame using Grid Layout

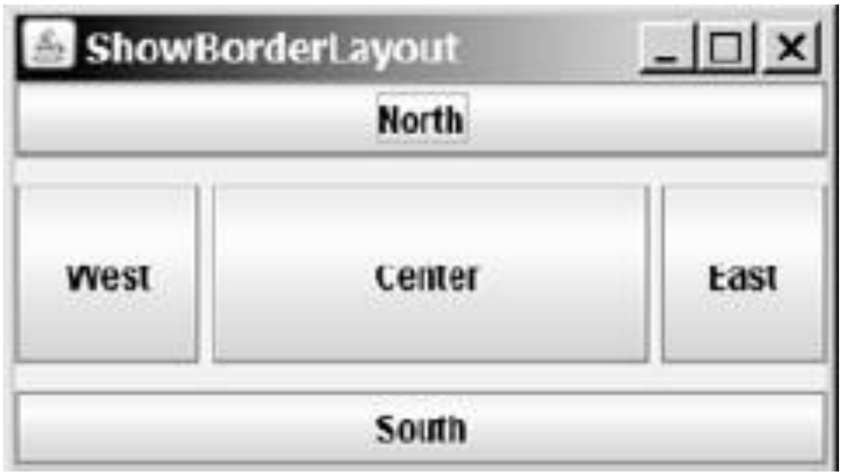


Solution:

```
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JFrame;
import java.awt.GridLayout;

public class ShowGridLayout extends JFrame {

    public ShowGridLayout() {
        //Set GridLayout, 3 rows, 2 columns, and
gaps 5 between
        //components horizontally and vertically
        setLayout(new GridLayout(3,2,5,5));
        //Add labels and text fields to the frame
        add(new JLabel("First Name"));
        add(new JTextField(8));
        add(new JLabel("MI"));
        add(new JTextField(1));
        add(new JLabel("Last Name"));
        add(new JTextField(8));

    }

    /**
     * Main method
     */
    public static void main(String[] args) {
        ShowGridLayout frame = new
ShowGridLayout();
        frame.setTitle("Show GridLayout");
        frame.setSize(200, 125);
        frame.setLocationRelativeTo(null);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLO
SE);
        frame.setVisible(true);
    }
}
```

Activity 3:

Run the below code and ensure that the output matches the UI below the code.

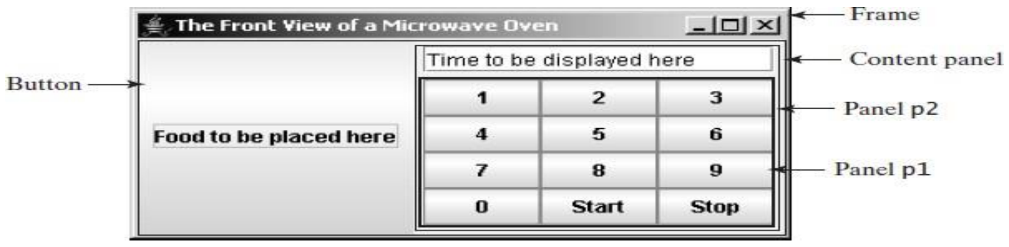


Solution:

```
import javax.swing.JButton;
import javax.swing.JFrame;
import java.awt.BorderLayout;

public class ShowBorderLayout extends JFrame {

    public ShowBorderLayout() {
        //Set BorderLayout with horizontal gaps 5
and vertical gap 10
        setLayout(new BorderLayout(5, 10));
        //Add buttons to the frame
        add(new JButton("EAST"),
BorderLayout.EAST);
        add(new JButton("SOUTH"),
BorderLayout.SOUTH);
        add(new JButton("WEST"),
BorderLayout.WEST);
        add(new JButton("NORTH"),
BorderLayout.NORTH);
        add(new JButton("CENTER"),
BorderLayout.CENTER);

    }

    /**
     * Main method
     */
    public static void main(String[] args) {
        ShowBorderLayout frame = new
ShowBorderLayout();
        frame.setTitle("Show BorderLayout");
        frame.setSize(300, 200);
        frame.setLocationRelativeTo(null);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLO
SE);
        frame.setVisible(true);
    }
}
```

Activity 4:

Run the below code and ensure that the output is similar to below:



Solution:

```
import java.awt.*;
import javax.swing.*;

public class TestPanels extends JFrame {

    public TestPanels() {
        // Create panel p1 for the buttons and
set GridLayout
        JPanel p1 = new JPanel();
        p1.setLayout(new GridLayout(4, 3));

        //Add buttons to the panel
        for (int i = 1; i <= 9; i++) {
            p1.add(new JButton("" + i));
        }
        p1.add(new JButton("" + 0));
        p1.add(new JButton("Start"));
        p1.add(new JButton("Stop"));

        //Create panel p2 to hold a text field
and p1
        JPanel p2 = new JPanel(new
BorderLayout());
```

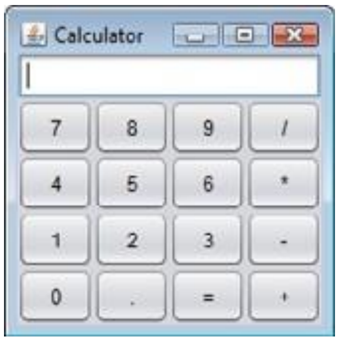
```
        p2.add(new JTextField("Time to be
displayed here"),
                BorderLayout.NORTH);
        p2.add(p1, BorderLayout.CENTER);

        // add contents into the frame
        add(p2, BorderLayout.EAST);
        add(new JButton("Food to be placed
here"),
                BorderLayout.CENTER);
    }

    /**
     * Main method
     */
    public static void main(String[] args) {
        TestPanels frame = new TestPanels();
        frame.setTitle("The Front View of a
Microwave Oven");
        frame.setSize(400, 250);
        frame.setLocationRelativeTo(null);
//Center the frame

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLO
SE);
        frame.setVisible(true);

    }
}
```

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficulty and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

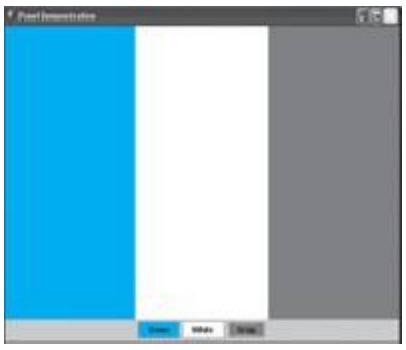
Lab Task 1

Create the following GUI. You do not have to provide any functionality.



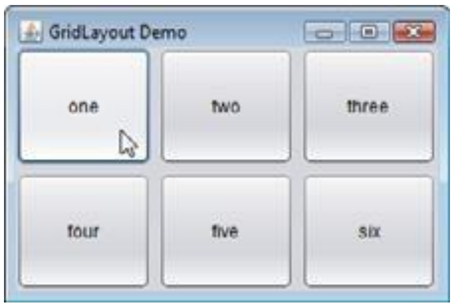
Lab Task 2

Create the following GUI. You do not have to provide any functionality.



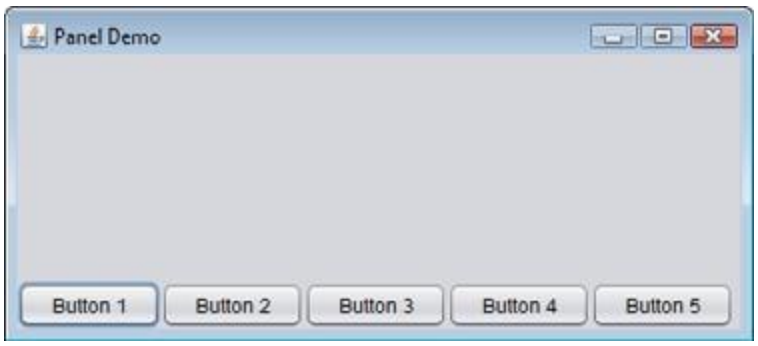
Lab Task 3

Create the following GUI. You do not have to provide any functionality.



Lab Task 4

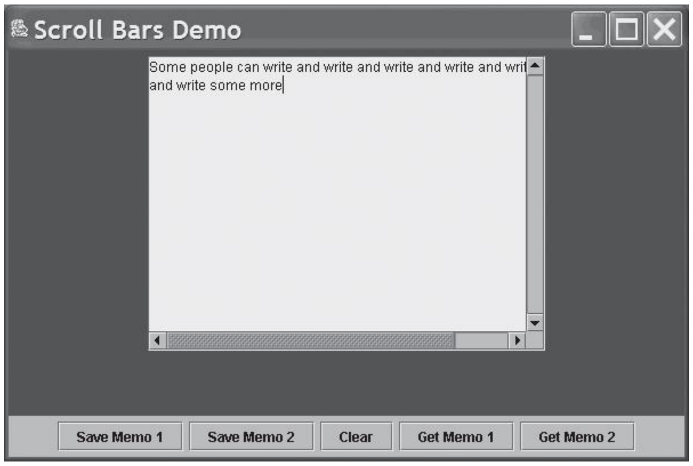
Create the following GUI. You do not have to provide any functionality.



Lab Task 5

Create the following GUI. You do not have to provide any functionality.

Note: Use JScrollPane class for the creating the scroll pane.



Some people can write and write and write and write and writ
and write some more

Lab 13

Graphical User Interface – Event Driven Programming

Objective:

In this lab, student will learn and practice the basic concepts of events-based programming in GUI based interfaces in Java. They will learn event generation and event handling.

Activity Outcomes:

After completing this lesson, you should be able to do the following:

- Understand why events are needed in GUI
- Understand the mechanics of event generation and event handling
- Practice simple event-based programming
- Create a simple but useful GUI based program

Instructor Note:

As pre-lab activity, read Chapter 15 from the text book "Introduction to Java Programming", Y. Daniel Liang, Pearson, 2019.

1) Useful Concepts

Any program that uses GUI (graphical user interface) such as Java application written for windows, is event driven. Event describes the change of state of any object.

Example:

Pressing a button, Entering a character in Textbox Event handling has three main components,

- **Events :** An event is a change of state of an object.
- **Events Source :** Event source is an object that generates an event.
- **Listeners :** A listener is an object that listens to the event. A listener gets notified when an event occurs.

A source generates an Event and sends it to one or more listeners registered with the source. Once event is received by the listener, they processe the event and then return. Events are supported by a number of Java packages, like **java.util**, **java.awt** and **java.awt.event**.

Important Event Classes and Interface

Event Classe	Description	Listener Interface
ActionEvent	generated when button is pressed, menu-item is selected, list-item is double clicked	**ActionListener**
MouseEvent	generated when mouse is dragged, moved,clicked,pressed or released also when the enters or exit a component	**MouseListener**
KeyEvent	generated when input is received from keyboard	**KeyListener**
ItemEvent	generated when check-box or list item is clicked	**ItemListener**

TextEvent	generated when value of textarea or textfield is changed	**TextListener**
MouseWheelEvent	generated when mouse wheel is moved	**MouseWheelListener**
WindowEvent	generated when window is activated, deactivated, deiconified, iconified, opened or closed	**WindowListener**
ComponentEvent	generated when component is hidden, moved, resized or set visible	**ComponentEventListener**
ContainerEvent	generated when component is added or removed from container	**ContainerListener**
AdjustmentEvent	generated when scroll bar is manipulated	**AdjustmentListener**
FocusEvent	**generated when component gains or loses keyboard focus**	**FocusListener**

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	***15 mins***	***High***	***CLO-4***
Activity 2	***15 mins***	***High***	***CLO-4***
Activity 3	***15 mins***	***High***	***CLO-4***

Activity 1:

Run the below code. It should create a label and a button. The label should have text "Hello" but when the button the pressed the text changes to "Bye"

Solution:

```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
```

```
public class Test extends JFrame {

    private JLabel label;

    private JButton b;

    public Test() {

        this.setLayout(new
FlowLayout(FlowLayout.LEFT, 10, 20));
        label = new JLabel("Hello");
        this.add(label);
        b = new JButton("Toggle");
        b.addActionListener(new myHandler());
        add(b);
    }

    class myHandler implements ActionListener {

        public void actionPerformed(ActionEvent
e) {
            label.setText("Bye");
        }

    }

    public static void main(String[] args) {

// TODO Auto-generated method stub
        Test f=new Test();
        f.setTitle("Hi and Bye");
        f.setSize(400, 150);

f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setLocationRelativeTo(null);
        f.setVisible(true);

    }

}
```

Activity 2:

Run and understand the below code. Basically this code sets the text in label on button press event. Any text entered in the textfield is copied into the label. Ensure that it is so and understand how it works.

Solution:

```
import java.awt.FlowLayout;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

public class Test extends JFrame {

    private JTextField tf1;
    private JLabel label;
    private JButton b;

    public Test() {

        this.setLayout(new
FlowLayout(FlowLayout.LEFT, 10, 20));
        tf1 = new JTextField(8);
        this.add(tf1);
        label = new JLabel("New Text");
        this.add(label);
        b = new JButton("Change");
        b.addActionListener(new myHandler());
        add(b);
    }

    class myHandler implements ActionListener {

        public void actionPerformed(ActionEvent
e) {
            String s = tf1.getText();
            label.setText(s);

            tf1.setText("");

        }

    }
    public static void main(String[] args) {

// TODO Auto-generated method stub
        Test f = new Test();
        f.setTitle("Copy Text");
        f.setSize(400, 150);

f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setLocationRelativeTo(null);;
        f.setVisible(true);

    }

}
```

Activity 3:

Run and understand the below code. We now first see which button triggered the event through the getSource event and then either disapper one button or copy text in the TextField into the label.

```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

public class Test extends JFrame {

    private JTextField tf1;
    private JLabel label;
    private JButton b;
    private JButton b1;

    public Test() {

        this.setLayout(new FlowLayout(FlowLayout.LEFT, 10, 20));
        tf1 = new JTextField(8);
        this.add(tf1);
        label = new JLabel("New Text");
        this.add(label);
        b = new JButton("Change");
        b.addActionListener(new myHandler());
        add(b);

        b1 = new JButton("Disappear");
        b1.addActionListener(new myHandler());
        add(b1);
    }

    class myHandler implements ActionListener {

        public void actionPerformed(ActionEvent e) {

            if (e.getSource() == b) {

                String s = tf1.getText();
                label.setText(s);
                tf1.setText("");
            }

            if (e.getSource() == b1) {
                b.setVisible(false);
            }

        }

    }

    public static void main(String[] args) {

// TODO Auto-generated method stub
        Test f = new Test();
        f.setTitle("Copy Text");
        f.setSize(400, 150);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setLocationRelativeTo(null);;
        f.setVisible(true);

    }

}
```

3) Graded Lab Tasks

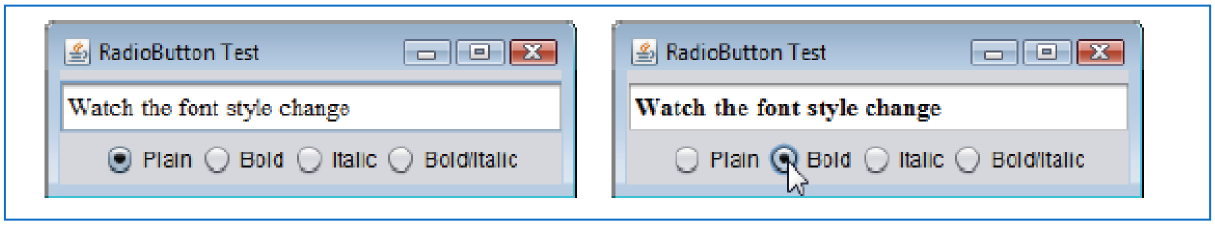
Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Create a frame with one label, one textbox and a button. Display the information entered in textbox on button click.

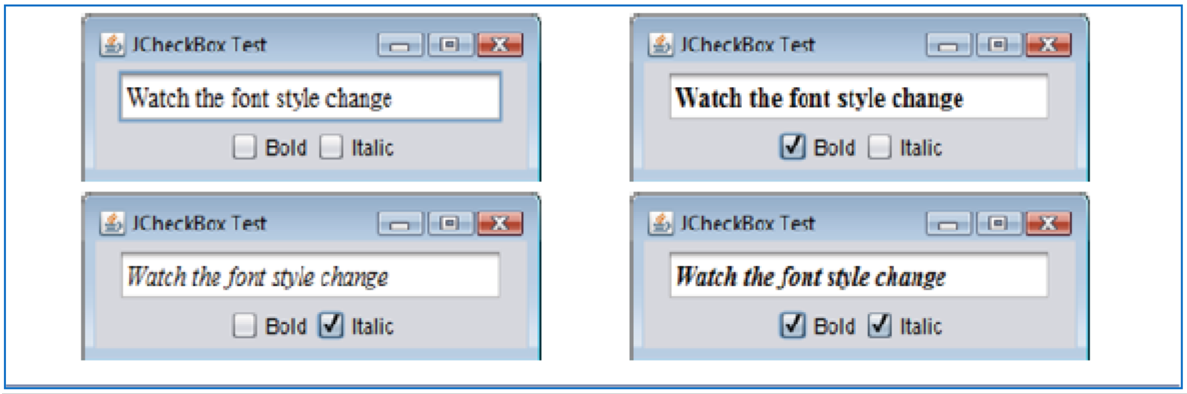
Lab Task 2

Create frames as follows:



Lab Task 3

Create Frames as follows:


JCheckBox Test
Watch the font style change
Bold
Italic
JCheckBox Test
Watch the font style change
Bold
Italic
JCheckBox Test
Watch the font style change
Bold
Italic
JCheckBox Test
Watch the font style change
Bold
Italic

Lab Task 4

Make a functional nonscientific calculator. Recall the last task of the previous lab.