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 (Allocated Time 1 Hr.)

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 (Allocated Time 2 Hrs.)

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.
.