Introducing Classes and Objects - II

Lab Objective

Familiarize students with the implementation of classes and instantiation of objects accordingly.

Lab Outcomes

After completing this lab successfully, students will be able to:

1. **Understand** and **Apply** the concepts of classes and objects.
2. **Write** the definition of a class as per the specification and **create** objects accordingly using constructors.
3. **Write** various instance methods performing different actions on the objects of a class.

Psychomotor Learning Levels

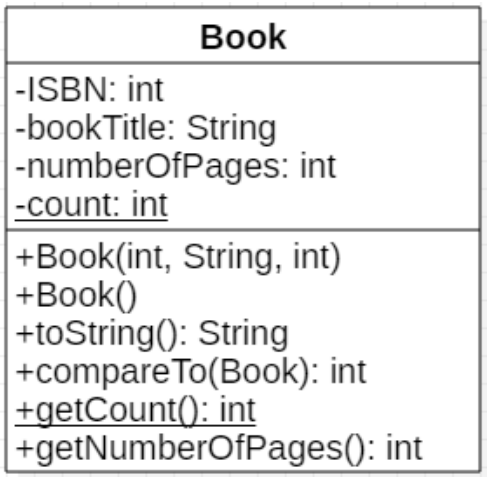
This lab involves activities that encompass the following learning levels in psychomotor domain.

Level	Category	Meaning	Keywords
P1	Imitation	Copy action of another; observe and replicate.	Relate, Repeat, Choose, Copy, Follow, Show, Identify, Isolate.
P2	Manipulation	Reproduce activity from instruction or memory	Copy, response, trace, Show, Start, Perform, Execute, Recreate.

Lab Activities

A. Static Variables in a class

- We have only seen instance variables so far. Consider the following class diagram of Book class.



- Here, ISBN, bookTitle and numberOfPages are instance variables. Each object has their own copy of instance variables.
- On the other hand, **objects share only one copy of a static variable.** Like static methods, static variables belong to a class, not to any objects.

- A static variable such as count in the Book class can be used to count the total number of Book type objects. **Each time, a Book type object is created, we need to increment the value of the count variable within the constructor.**
- A method that returns the value of a static variable is called a static method. In this Book class, getCount() is a static method that returns the value of count.
- **Your job is to implement the Book class as per the above-mentioned class diagram.**

B. Demonstrating Calendar class for manipulating dates

```
public class DateDemo {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Calendar date = Calendar.getInstance();
		System.out.println("The date is " + date.getTime() + " " +
					date.getTimeInMillis());
		//changing the date
		date.set(Calendar.YEAR, 2020);
		date.set(Calendar.MONTH, 0); //January starts from 0
		date.set(Calendar.DAY_OF_MONTH, 1);
		date.set(Calendar.HOUR_OF_DAY, 0);
		date.set(Calendar.MINUTE, 0);
		date.set(Calendar.SECOND, 1);
		System.out.println("The changed date is " + date.getTime() + " " +
					date.getTimeInMillis());
		System.out.println(date.toString());
		System.out.println(System.currentTimeMillis());
	}
}
```

C. Demonstrating Point2D class for manipulating points in a two-dimensional space

javafx.geometry.Point2D	
+Point2D(x: double, y: double)	Constructs a Point2D object with the specified *x*- and *y*-coordinates.
+distance(x: double, y: double): double	Returns the distance between this point and the specified point (*x*, *y*).
+distance(p: Point2D): double	Returns the distance between this point and the specified point p.
+getX(): double	Returns the *x*-coordinate from this point.
+getY(): double	Returns the *y*-coordinate from this point.
+toString(): String	Returns a string representation for the point.

```
import java.util.Scanner;
import javafx.geometry.Point2D;

public class TestPoint2D {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    System.out.print("Enter point1's x-, y-coordinates: ");
    double x1 = input.nextDouble();
    double y1 = input.nextDouble();
    System.out.print("Enter point2's x-, y-coordinates: ");
    double x2 = input.nextDouble();
    double y2 = input.nextDouble();

    Point2D p1 = new Point2D(x1, y1);
    Point2D p2 = new Point2D(x2, y2);
    System.out.println("p1 is " + p1.toString());
    System.out.println("p2 is " + p2.toString());
    System.out.println("The distance between p1 and p2 is " +
      p1.distance(p2));
  }
}
```

D. Immutable Class and Immutable Objects

- **<u>The contents of immutable objects cannot be changed.</u>**
- The corresponding class needs to be immutable too.
- **<u>A class is immutable if it satisfies the following:</u>**
  - All data fields must be private.
  - There can't be any mutator methods (setters) for data fields.
  - No accessor methods can return a reference to a data field that is mutable.
- **<u>Is the following class immutable?</u>**

```
public class Student {
      private int id;
      private String name;
      private java.util.Date dateCreated;
      public Student(int ssn, String newName) {
            id = ssn;
            name = newName;
            dateCreated = new java.util.Date();
      }
      public int getId() {
            return id;
      }
      public String getName() {
            return name;
      }
      public java.util.Date getDateCreated() {
            return dateCreated;
      }
}
```

This lab sheet must be preserved till the end of the semester.

Student ID		**Solved during the LAB**	
Student Name			
Section		**Solved after the LAB**	
Date			

Lab 05: Lab Problems (Mid I Problems)

Lab05_Problem01:

Write a Java program that prompts user to input the length (*l*) and width (*w*), of a rectangle. If *l* and *w* are positive (*l > 0* and *w > 0*), you must display the area and the perimeter of the rectangle. If *l* is equal to 0 and *w* is positive, or vice versa, then you must print "It is a line". If both are 0, you must print "It is a point". Otherwise (*l < 0* and *w < 0*), print "Invalid length and width".
You must take input and print output of the program using dialog boxes (JOptionPane class). The formulas for calculating area and perimeter of the rectangle is given below.

area = $l \times w$, perimeter = $2 \times (l + w)$

Lab05_Problem02:

Consider the following dataset that shows final marks of a set of students in different subjects in EWU.

Subjects	Name				
	Alice	Bob	Charlie	Danny	Elton
CSE103	75	50	65	80	90
CSE106	80	45	60	50	70
CSE110	65	60	70	75	80
CSE207	40	65	85	65	80
CSE209	55	55	75	70	65

A student will pass the course if he/she obtains at least 60.

Write a program in Java that uses the above-mentioned dataset and determines the number of students passed in each course and the student who gets the highest mark. You must use a method named *processCourseData* that takes data of a particular course and prints the number of students passed in that course and the name of the students who obtained the highest mark. A sample output is given below.

```
Final Exam Result
CSE103 - Total Passed: 4, Highest: Elton
CSE106 - Total Passed: 3, Highest: Alice
CSE110 – Total Passed: 5, Highest: Elton
CSE207 – Total Passed: 4, Highest: Charlie
CSE209 - Total Passed: 3, Highest: Charlie
```

Lab05_Problem03:

Define a class *Line* that has four private instance variables: (x1, y1) and (x2, y2), representing the value of the x-coordinate and y-coordinate of the first point and second point on the line, respectively, in a two-dimensional coordinate system. The following figure shows a line with these two points.

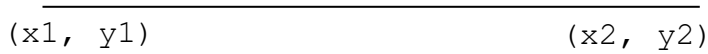


Figure: A line with two points (x1, y1) and (x2, y2)

The class *Line* has the following instance methods.

- *findSlope()*: the method takes no parameter and returns the slope of the line. The formula for calculating the slope of a line using two points is given below.

$$slope = \frac{y2-y1}{x2-x1}$$

- *toString()*: the method returns a String containing the values of *(x1, y1)* and *(x2, y2)* on a line. As an example, for a line that has two points *(3, 4)* and *(9, 5)*, the method may return the following String.

```
Line has points (3, 4) and (9, 5)
```

Use appropriate datatype. Your class definition must also include constructors.

Now, **write** a Java program that creates an array of *Line* type objects, as specified in question no. 5. The size of the array is 4. User may input the values of x and y-coordinates of two points on those lines using Scanner class.

Your program should also have the following static method.

isIntersecting (Line l1, Line l2)

The method takes two parameters – both are *Line* type objects.

The method must return true if two lines are intersecting each other at any point. Otherwise, the method returns false. You can determine whether a line intersects another line by comparing their slope. If slopes of both lines are the same, then lines are parallel, not intersecting each other. Otherwise, they will intersect each other at some point. You must use *findSlope()* instance method as defined in question 5 while defining this static method.

Your main method must call *isIntersecting()* method with appropriate parameters and print the result accordingly.