Lab Manual: 05

Lab Topic: Class Relationships

Course Code: CSE1116 (Object Oriented Programming Laboratory)

Course Instructor: Mir Moynuddin Ahmed Shibly, Lecturer, CSE, UIU.

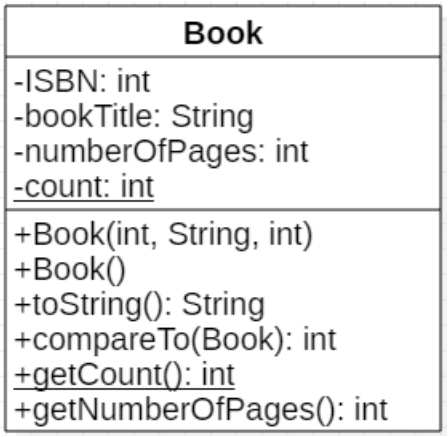
Lab Objective

1. **Familiarize** students with the implementation of classes
2. **Write** various instance methods performing different actions on the objects of a class
3. **Write** the definition of multiple classes and
4. Can **identify** and **hold** their relationships among each other.

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 its 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. Immutable Class and Immutable Objects

- **The contents of immutable objects cannot be changed.**
  The corresponding class needs to be immutable too.
- **A class is immutable if it satisfies the following:**
  - 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.
- **Is the following class immutable?**

```
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.DategetDateCreated() {
            return dateCreated;
      }
}
```

D. 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));
  }
}
```

Problem_1: **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.

```
(x1, y1)                              (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.
Write a Java program that creates an array of *Line* type objects. 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, Linel2)

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

E. Defining Multiple Classes

- We want to develop a minimal, simple object-oriented application for a university.
- A university has three major entities: Students, Faculties and Courses.
- First, we have to identify the relationships among them.
- The following relationship diagram shows the relationships among Student, Course and Faculty class.

Figure1: Relationship among Student, Course and Faculty

Problem_2: Your job is to define the above-mentioned classes as per the specification mentioned below and then write a Main/Driver class that demonstrates the functionalities of these classes.

Student	**Course**	**Faculty**
- studentId: int -studentName: String -studentCGPA: double	- courseId: String - courseTitle: String - credit: double - studentList: Student [] - numberOfStudents: int - faculty: Faculty	- facultyId: int - facultyName: String - facultyPosition: String
+ Student() + Student(studentId, studentName, studentCGPA) + toString(): String	+ Course() + Course(courseId, courseTitle, credit) + toString(): String + addStudent(Student): void + dropStudent(studentId): void + addFaculty(Faculty): void + dropFaculty(): void + printStudentList(): void	+ Faculty() +Faculty(facultyId, facultyName, facultyPosition) + toString(): String

Assignment: Developing a Menu-based Application

- Now, we need to develop a menu-based application.
- The initial menu may have the following options:
  a. Add
  b. Delete
  c. Update
  d. Print
  e. Search
- For each of these options, we may provide further options. Suppose, for 'Add' option, next we may show the following options:

a. Add a Student
b. Add a Course
c. Add a Faculty

For ‘Delete’ and ‘Update’, we may provide the same options.

- For ‘Print’ option, we may further provide the followings:
  a. Print all students
  b. Print all course
  c. Print all faculties
  d. Print information of a student
  e. Print information of a course
  f. Print information of a faculty
  g. Print student list and faculty information of a course
  h. Print courses taken by a student
- Search is very important feature in our application. For ‘Search’ option, we may provide the followings:
  a. Search a Student
  b. Search a Course
  c. Search a Faculty
  d. Search whether a student takes a course
  e. Search whether a faculty teaches a course
  f. Search courses taken by a student
  g. Search courses taught by a faculty