CSE 215L: Programming language II Lab
Faculty: Dr. Mohammad Rashedur Rahman (RRn)
Lab - 08 [Class & Object, Composition]
Fall-2022
Lab Instructor: Md. Mustafizur Rahman

Objective:

- To demonstrate how to define classes and objects
- To design a **Course** class and implement all its methods
- To define a new class with Composition *(aka Aggregation)*
- To reuse existing classes

Tasks:

1. Implement the **Course** class as follows:

Course
- courseName : String - students : String[ ] - numberOfStudents : int
+ Course(courseName : String) + getCourseName() : String + addStudent(student : String) : void + dropStudent(student : String) : void + getStudents() : String[ ] + getNumberOfStudents : int + clear() : void

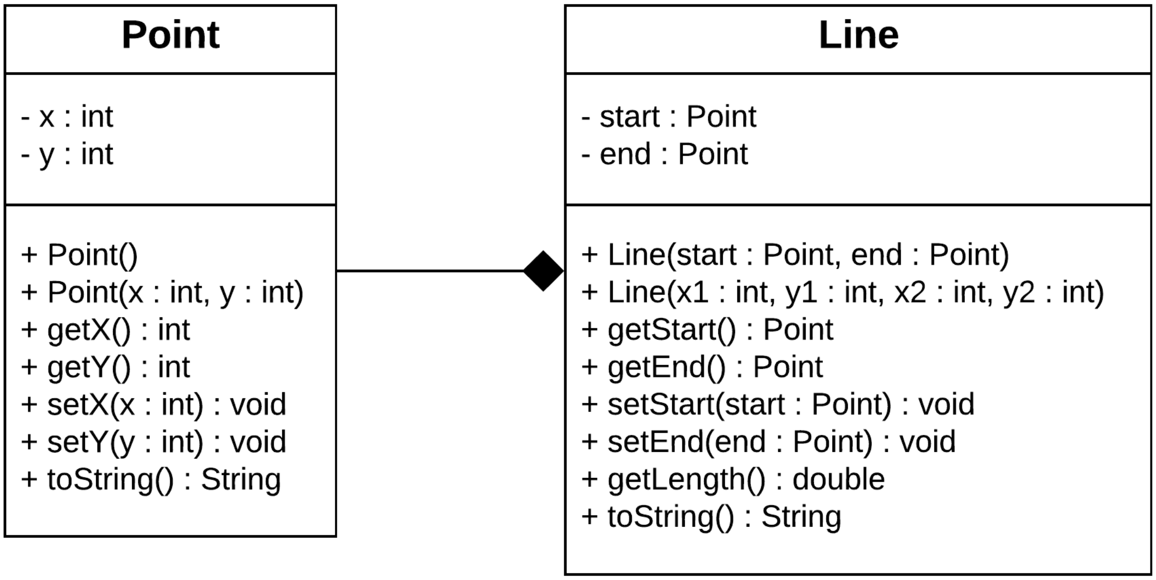
- The program I demonstrated using the **Course** class, was having a fixed size of array. Now, revise the program so that it automatically increases the array size by creating a new larger array and copying the contents of the current array to it.
- Implement the **dropStudent()** method.
- Add a new method named **clear()** that removes all students from the course.

Now write a test program that creates a course, adds three students, removes one, and displays the students in the course.

There are two ways to *reuse* existing classes, namely, *composition* and *inheritance*. With *composition* (aka *aggregation*), you define a new class, which is composed of existing classes. With *inheritance*, you derive a new class based on an existing class, with modifications or extensions.

As an example of reusing a class via composition, suppose that we have an *existing* class called Point, defined as shown in the below class diagram. Suppose that we need a new class called Line. We can design the Line class by re-using the Point class via *composition*. We say that "A line is *composed* of two points", or "A line *has* two points". Composition exhibits a "*has-a*" relationship.

<u>Task-02:</u> Implement the following UML Class Diagrams and test their method


Point
- x : int
- y : int
+ Point()
+ Point(x : int, y : int)
+ getX() : int
+ getY() : int
+ setX(x : int) : void
+ setY(y : int) : void
+ toString() : String
Line
- start : Point
- end : Point
+ Line(start : Point, end : Point)
+ Line(x1 : int, y1 : int, x2 : int, y2 : int)
+ getStart() : Point
+ getEnd() : Point
+ setStart(start : Point) : void
+ setEnd(end : Point) : void
+ getLength() : double
+ toString() : String