CSE 215L: Programming Language II Lab
Faculty: Silvia Ahmed, Sec – 4, 5
Lab instructor: Marufa Ferdausi
Lab 08 – Fall 2019

Objective:
After today's lab, the students should be able:

- To explore the differences between the procedural paradigm and object-oriented paradigm
- To discover the relationships between classes
- To design programs using the object-oriented paradigm
- To simplify programming using automatic conversion between primitive types and wrapper class types

Task – 1

(*The* **Time** *class*) Design a class named **Time**. The class contains:

■ The data fields **hour**, **minute**, and **second** that represent a time.

■ A no-arg constructor that creates a **Time** object for the current time. (The values of the data fields will represent the current time.)

■ A constructor that constructs a **Time** object with a specified elapsed time since midnight, January 1, 1970, in milliseconds. (The values of the data fields will represent this time.)

■ A constructor that constructs a **Time** object with the specified hour, minute, and second.

■ Three getter methods for the data fields **hour**, **minute**, and **second**, respectively.

■ A method named **setTime(long elapseTime)** that sets a new time for the object using the elapsed time. For example, if the elapsed time is **555550000** milliseconds, the hour is **10**, the minute is **19**, and the second is **10**.

Draw the UML diagram for the class and then implement the class. Write a test program that creates two **Time** objects (using **new Time()** and **new Time(555550000)**) and displays their hour, minute, and second in the format hour:minute:second.

(*Hint*: The first two constructors will extract the hour, minute, and second from the elapsed time. For the no-arg constructor, the current time can be obtained using **System.currentTimeMillis**).

Task – 2

(*The* **MyInteger** *class*) Design a class named **MyInteger**. The class contains:

■ An **int** data field named **value** that stores the **int** value represented by this object.

■ A constructor that creates a **MyInteger** object for the specified **int** value.

■ A getter method that returns the **int** value.

■ The methods **isEven()**, **isOdd()**, and **isPrime()** that return **true** if the value in this object is even, odd, or prime, respectively.

■ The static methods **isEven(int)**, **isOdd(int)**, and **isPrime(int)** that return **true** if the specified value is even, odd, or prime, respectively.

■ The static methods **isEven(MyInteger)**, **isOdd(MyInteger)**, and **isPrime(MyInteger)** that return **true** if the specified value is even, odd, or prime, respectively.

■ The methods **equals(int)** and **equals(MyInteger)** that return **true** if the value in this object is equal to the specified value.

■ A static method **parseInt(char[])** that converts an array of numeric characters to an **int** value.

■ A static method **parseInt(String)** that converts a string into an **int** value.

Draw the UML diagram for the class and then implement the class. Write a client program that tests all methods in the class.

Task – 3

(*The* **Queue** *class*) Design a class named **Queue** for storing integers. Like a stack, a queue holds elements. In a stack, the elements are retrieved in a last-in first-out fashion. In a queue, the elements are retrieved in a first-in first-out fashion. The class contains:

■ An **int[]** data field named **elements** that stores the **int** values in the queue.
■ A data field named **size** that stores the number of elements in the queue.
■ A constructor that creates a **Queue** object with default capacity **8**.
■ The method **enqueue(int v)** that adds **v** into the queue.
■ The method **dequeue()** that removes and returns the element from the queue.
■ The method **empty()** that returns true if the queue is empty.
■ The method **getSize()** that returns the size of the queue.

Draw an UML diagram for the class. Implement the class with the initial array size set to 8. The array size will be doubled once the number of the elements exceeds the size. After an element is removed from the beginning of the array, you need to shift all elements in the array one position the left. Write a test program that adds 20 numbers from 1 to 20 into the queue and removes these numbers and displays them

Task – 4

(*Geometry: the* **Circle2D** *class*) Define the **Circle2D** class that contains:

■ Two **double** data fields named **x** and **y** that specify the center of the circle with getter methods.
■ A data field **radius** with a getter method.
■ A no-arg constructor that creates a default circle with (**0**, **0**) for (**x**, **y**) and **1** for **radius**.
■ A constructor that creates a circle with the specified **x**, **y**, and **radius**.
■ A method **getArea()** that returns the area of the circle.
■ A method **getPerimeter()** that returns the perimeter of the circle.
■ A method **contains(double x, double y)** that returns **true** if the specified point (**x**, **y**) is inside this circle (see Figure 1a).
■ A method **contains(Circle2D circle)** that returns **true** if the specified circle is inside this circle (see Figure 1b).
■ A method **overlaps(Circle2D circle)** that returns **true** if the specified circle overlaps with this circle (see Figure 1c).

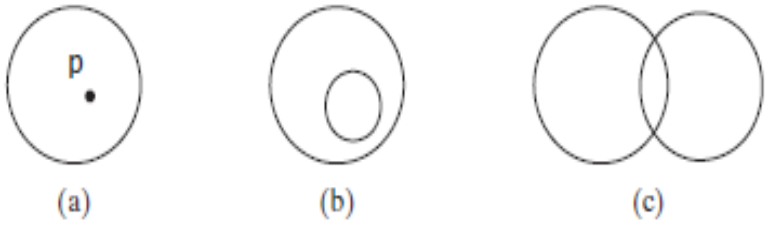

Fig. 1: (a) A point is inside the circle. (b) A circle is inside another circle. (c) A circle overlaps another circle.

Draw the UML diagram for the class and then implement the class. Write a test program that creates a **Circle2D** object **c1** (**new Circle2D(2, 2, 5.5)**), displays its area and perimeter, and displays the result of **c1.contains(3, 3)**, **c1.contains (new Circle2D(4, 5, 10.5))**, and **c1.overlaps(new Circle2D(3, 5, 2.3))**.

Task – 5

(*Geometry: the* **MyRectangle2D** *class*) Define the **MyRectangle2D** class that contains:

■ Two **double** data fields named **x** and **y** that specify the center of the rectangle with getter and setter methods. (Assume that the rectangle sides are parallel to **x-** or **y-** axes.)

■ The data fields **width** and **height** with getter and setter methods.
■ A no-arg constructor that creates a default rectangle with (**0**, **0**) for (**x**, **y**) and **1** for both **width** and **height**.
■ A constructor that creates a rectangle with the specified **x**, **y**, **width**, and **height**.
■ A method **getArea()** that returns the area of the rectangle.
■ A method **getPerimeter()** that returns the perimeter of the rectangle.
■ A method **contains(double x, double y)** that returns **true** if the specified point (**x**, **y**) is inside this rectangle (see Figure 2a).
■ A method **contains(MyRectangle2D r)** that returns **true** if the specified rectangle is inside this rectangle (see Figure 2b).
■ A method **overlaps(MyRectangle2D r)** that returns **true** if the specified rectangle overlaps with this rectangle (see Figure 2c).

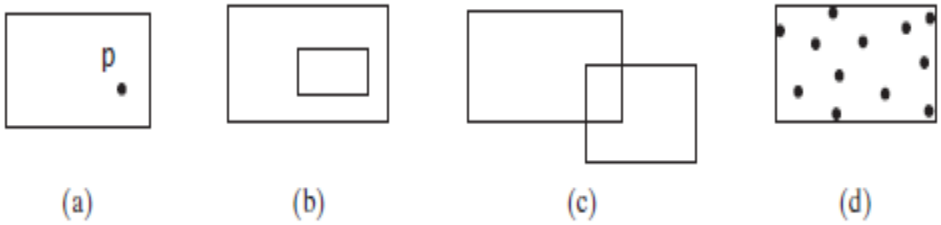

Fig. 2: A point is inside the rectangle. (b) A rectangle is inside another rectangle. (c) A rectangle overlaps another rectangle. (d) Points are enclosed inside a rectangle.
Draw the UML diagram for the class and then implement the class. Write a test program that creates a **MyRectangle2D** object **r1** (**new MyRectangle2D(2, 2, 5.5, 4.9)**), displays its area and perimeter, and displays the result of **r1.contains(3, 3)**, **r1.contains(new MyRectangle2D(4, 5, 10.5, 3.2))**, and **r1.overlaps(new MyRectangle2D(3, 5, 2.3, 5.4))**.

Task – 6
(*The* **MyDate** *class*) Design a class named **MyDate**. The class contains:
■ The data fields **year**, **month**, and **day** that represent a date. **month** is 0-based, i.e., **0** is for January.
■ A no-arg constructor that creates a **MyDate** object for the current date.
■ A constructor that constructs a **MyDate** object with a specified elapsed time since midnight, January 1, 1970, in milliseconds.
■ A constructor that constructs a **MyDate** object with the specified year, month, and day.
■ Three getter methods for the data fields **year**, **month**, and **day**, respectively.
■ A method named **setDate(long elapsedTime)** that sets a new date for the object using the elapsed time.
Draw the UML diagram for the class and then implement the class. Write a test program that creates two **MyDate** objects (using **new MyDate()** and **new MyDate(34355555133101L)**) and displays their year, month, and day.
(*Hint*: The first two constructors will extract the year, month, and day from the elapsed time. For example, if the elapsed time is **561555550000** milliseconds, the year is **1987**, the month is **9**, and the day is **18**. You may use the **GregorianCalendar** class in the **java.util** package**,** to simplify coding.)