

CSCI 2110- Data Structures and Algorithms
Laboratory No. 1
Week of September 11th, 2017

Review of Object-Oriented Programming Concepts

This lab is a quick review to help you to get over the “rustiness” and to get you back on track on the fundamentals of object-oriented programming concepts. Your task is to write, compile and run each program on an Integrated Development Environment (IDE) such as Eclipse.

If the test data is included in the question, use that to test your classes. In addition, test it with two more input sets. Otherwise, create your own test data and run your program for at least 3 input sets such that they cover the range of results expected. In some cases, you may have to test for more than 3 input sets to cover the range of results expected.

Marking Scheme: *Throughout this course, we will be focusing on the complexity of algorithms and consequently, efficiency of programs. While in CS1 and CS2 your main objective was getting the code to work, in CS3 you not only have to get the code to work, but also make it efficient. This will involve reducing unnecessary coding and designing or choosing good algorithms. With this in mind, here’s the marking scheme:*

Each exercise carries 10 points. Your final score will be scaled down to a value out of 10.

Working code, Outputs included, Efficient, Good basic comments included: 10/10

No comments: subtract one point

Unnecessarily inefficient: subtract one point

No outputs: subtract two points

Code not working: subtract up to six points depending upon how many methods are incorrect.

Error checking: *Unless otherwise specified, you may assume that the user enters the correct data types and the correct number of input entries, that is, you need not check for errors on input.*

Submission: *All submissions are through Brightspace. Log on dal.ca/brightspace using your Dal NetId. Submissions are pretty straightforward. Instructions will be also be given in the first lab.*

Deadline for submission: SUNDAY, September 17, 2017 at 11.55 p.m. (five minutes before midnight).

What to submit:

One zip file containing all source codes and a text document containing the sample outputs.

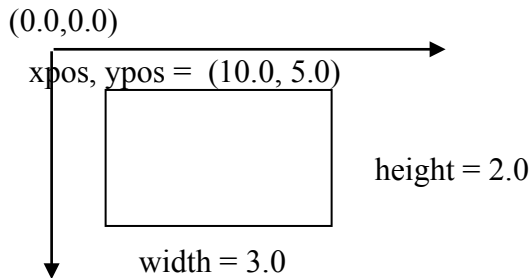
*You **MUST SUBMIT** .java files that are readable by the TA. If you submit files that are unreadable such as .class, you will lose points.*

Exercise 1:

This is similar to the exercise that was discussed in the lectures. However, you need to define two methods, namely, contains and touches, that have slightly different meanings. Please see the diagrams.

Define the Rectangle class that contains:

- Double data fields named xpos, ypos (that specify the top left corner of the rectangle), width and height. (assume that the rectangle sides are parallel to the x and y axes. See example below).



- A constructor that creates a rectangle with the specified xpos, ypos, width, and height.
- Get and set methods for all the instance variables.
- A method contains(Rectangle r) that returns true if the specified rectangle is inside this rectangle.
- A method touches(Rectangle r) that returns true if the specified rectangle touches this rectangle.

You may add other methods if necessary.

Study the figures to understand when the contains method should return true, and when the touches method should return true. If the rectangles overlap, both methods should return false.

Write a test program that creates two Rectangle objects with input specified by the user. Test it for all the cases shown in the diagram.

Exercise 2: Define a Circle class that contains:

- Two double data fields named x and y that specify the center of the circle.
- A double data field radius.
- Getter methods for x, y and radius.
- A constructor that creates a circle with the specified x,y and radius.
- A method contains(Circle c) that returns true if the specified Circle is inside this Circle.
- A method touches(Circle c) that returns true if the specified Circle touches this Circle.

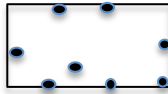
You may add other methods if necessary.

Study the figures to understand when the contains method should return true, and when the touches method should return true. If the Circles overlap, both methods should return false.

Write a test program that creates two Circle objects with input specified by the user. Test it for all the cases shown in the diagram.

Hint: One way to implement the contains and touches methods is to find the distance between the two centers and compare that with the sum of the two radii. The distance between two points $p1=(x1, y1)$ and $p2 = (x2,y2)$ is given by the square root of $(x2-x1)^2 + (y2-y1)^2$

Exercise 3: A bounding rectangle is the minimum rectangle that encloses a set of points in a two-dimensional plane, as shown in the figure below.



Write a method that returns a bounding rectangle for a set of points in a two dimensional plane as follows:

```
public static Rectangle getRectangle(double[][] points)
{
    //This method finds the xpos, ypos, width and height of the
    //bounding rectangle
    //constructs it and returns it
}
```

The Rectangle class is the one that you developed in Exercise 1. Use the following test program that prompts the user to enter five points and displays the **bounding rectangle's center**, width, and height. As you can observe from the code, the x and y coordinates of each point are stored in the 2-d array. For example, if the points are (1.0,2.0), (1.5, 2.0), (3.5, 4.0), (2.5, 1.0), (5.0, 6.0), you would store them in the following 2-d array:

```
[1.0  2.0]
[1.5  2.0]
[3.5  4.0]
[2.5  1.0]
[5.0  6.0]
```

The code below can store up to five points, but you can change the array dimensions appropriately.

```
//Code for Exercise 3
import java.util.Scanner;
public class Exercise3 {
    public static void main(String[] args) {
        double[][] points = new double[5][2];

        Scanner input = new Scanner(System.in);
        System.out.print("Enter " + points.length + " points: ");
        for (int i = 0; i < points.length; i++) {
            points[i][0] = input.nextDouble();
            points[i][1] = input.nextDouble();
        }

        Rectangle boundingRectangle = getRectangle(points);

        //TODO: Display its center, width and height
    }

    public static Rectangle getRectangle(double[][] points)
    {
```

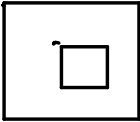
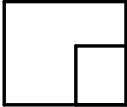
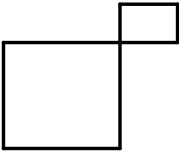
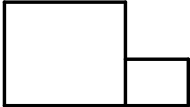
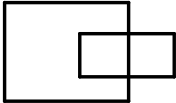
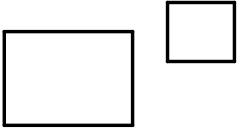
```
    //TODO - Complete this method
    //This method finds the xpos, ypos, width and height of the
    //bounding rectangle
    //constructs it and returns it
}
```

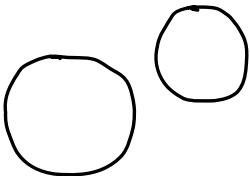
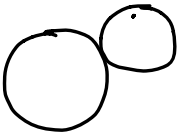
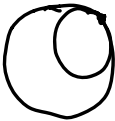
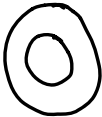
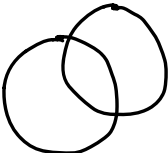
Here's a sample run:

Enter five points: 1.0 2.5 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0

The bounding rectangle's center (5.0, 6.25), width 8.0, height 7.5

Repeat the test for two more cases.

		Contains	Touches
	→	True	False
	→	True	True
	→	False	True
	→	False	True
	→	False	False
	→	False	False

		Contains	Touches
	→	False	False
	→	False	True
	→	True	True
	→	True	False
	→	False	False