CSCI 2110 Data Structures and Algorithms
Lab 1
Week of September 14th

Due: Sunday Sept 20th
23h55 (five minutes to midnight)

Review of Object-Oriented Programming Concepts

This lab is a quick review to help get you back on track and provide a refresher on the fundamentals of object-oriented programming. Your task is to write, compile and run each program using an Integrated Development Environment (IDE) such as *Eclipse* or *IntelliJ*.

Marking Scheme: Throughout this course, we will be focusing on the complexity of algorithms and consequently, the efficiency of programs. In past CS courses your focus was on creating runnable code to specifications. This course builds on previous knowledge, but also focuses on efficiency. This will involve reducing unnecessary coding and designing or choosing good algorithms.
Each exercise will list the test cases that you need to test the code. Ensure that the output that you generate covers all the test cases.

Each exercise carries 10 points.
Working code, Outputs included, Efficient, Good basic comments included: 10/10
No comments or poor commenting: **subtract one point**
Unnecessarily inefficient: **subtract one point**
Not all required methods were present: **subtract one point**
No outputs and/or not all test cases covered: **subtract up to one point**
Code not working: **subtract up to six points** depending upon how many test cases are incorrect.

Your final score will be scaled down to a value out of 10. For example, if there are three exercises and you score 9/10, 10/10 and 8/10 on the three exercises, your total is 27/30 or 9/10.

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.. Instructions will be also be given in the first lab.

What to submit:
One ZIP file containing all source code (files with .java suffixes) **and a text document** containing sample outputs. For each exercise you will minimally have to submit a demo class (eg. Exercise1.java), with a main method and your test code. If your main calls methods from or creates instances of other classes (eg. Rectangle.java) those files must be included in your submission as well.
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. **You will be provided with a test case file for each exercise and the expected outputs. Copy and paste the input into your IDE at runtime. Compare the output to the expected output. Include this test case output in your text file.**

***Hint:** Don't try to tackle the entire exercise in one go. Code a little, test a little, code a little, test a little. You will have better progress if you frequently compile and test your code (every 5-10 lines).

Exercise0A (Review - no submission required)

A basic object-oriented program to define a Rectangle, construct it and display its parameters. Here's the code. Study it and try it out. You can copy and paste it directly into IntelliJ or Eclipse. Make sure the java file names match.

```
//A Basic Object-oriented program
//Illustrates defining and creating a Rectangle object with xpos, ypos, width and height
public class Rectangle{
  //instance variables
  private int xpos, ypos, width, height;

  //constructors
  public Rectangle(){}
  public Rectangle(int xpos, int ypos, int width, int height){
    this.xpos=xpos; this.ypos=ypos; this.width=width; this.height=height;
  }

  //setters and getters
  public void setX(int xpos){this.xpos=xpos;}
  public void setY(int ypos){this.ypos=ypos;}
  public void setWidth(int width){this.width=width;}
  public void setHeight(int height){this.height=height;}
  public int getX(){return xpos;}
  public int getY(){return ypos;}
  public int getWidth(){return width;}
  public int getHeight(){return height;}

  //other methods: moveTo changes xpos and ypos and resize changes
  //width and height
  public void moveTo(int xpos, int ypos){this.xpos=xpos; this.ypos=ypos;}
  public void resize(int width, int height){this.width=width; this.height=height;}

  //toString method
  public String toString(){
    return "[xpos= " +xpos+","+"ypos= " + ypos+"] width: " +
          width+",height: "+height;
    }
}


//Demo class
public class Exercise0A{
  public static void main(String[] args){
    Rectangle rect1 = new Rectangle(10, 20, 300, 400);
    System.out.println("Created one rectangle object:\n" + rect1);
    rect1.moveTo(30, 40);
    System.out.println("Moved to new position: \n" + rect1);
    rect1.resize(350, 450);
    System.out.println("Resized to new dimensions: \n" + rect1);
  }
}
```

Exercise0B (Review– no submission required)

An extension of a basic object-oriented program – has two *contains* methods with test cases. Here's the code. Study it and try it out. You can copy and paste it directly into IntelliJ or Eclipse. Make sure the java file names match.

```
//Rectangle class that defines a Rectangle object with xpos, ypos, width and height
//Has two contains methods
public class Rectangle1{
 //instance variables
 private int xpos, ypos, width, height;

 //constructors
 public Rectangle1(){}
 public Rectangle1(int xpos, int ypos, int width, int height){
  this.xpos=xpos; this.ypos=ypos; this.width=width; this.height=height;}

 //setters and getters
 public void setX(int xpos){this.xpos=xpos;}
 public void setY(int ypos){this.ypos=ypos;}
 public void setWidth(int width){this.width=width;}
 public void setHeight(int height){this.height=height;}
 public int getX(){return xpos;}
 public int getY(){return ypos;}
 public int getWidth(){return width;}
 public int getHeight(){return height;}

 //other methods: moveTo changes xpos and ypos and resize changes
 //width and height
 public void moveTo(int xpos, int ypos){this.xpos=xpos; this.ypos=ypos;}
 public void resize(int width, int height){this.width=width; this.height=height;}

 //toString method
 public String toString(){
  return "[xpos= " +xpos+","+"ypos= " + ypos+"] width: " +
        width+",height: "+height;
 }

 //contains method: returns true if a point (px, py) is contained within this rectangle
 //contains also returns true if the point touches the rectangle
 public boolean contains(int px, int py){
  return (px>=xpos && px<=xpos+width && py>=ypos && py<= ypos+height);
 }
 //contains method: returns true if another rectangle r is contained within this rectangle
 //returns true if the rectangle touches the boundaries
 //it uses the point contains method
 public boolean contains(Rectangle1 r){
  return(this.contains(r.getX(),r.getY())&&
       this.contains(r.getX() + r.getWidth(), r.getY()+r.getHeight()));
 }
}
```

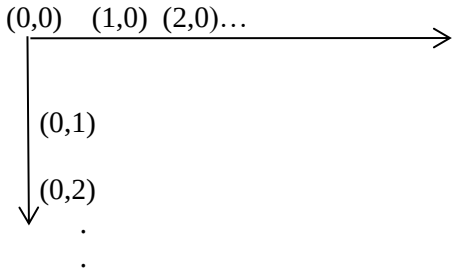
```
//Demo class
public class Exercise0B{
 public static void main(String[] args){
   Rectangle1 rect1 = new Rectangle1(10, 20, 300, 400);
   Rectangle1 rect2 = new Rectangle1(15, 25, 100, 100);

   System.out.println("Point (30,40) is contained in " + rect1 + "?\t" +
                rect1.contains(30,40));
   System.out.println("Point (10,20) is contained in " + rect1 + "?\t" +
               rect1.contains(10,20));
   System.out.println("Point (4,3) is contained in " + rect1 + "?\t" +
               rect1.contains(4,3));
   System.out.println("Rectangle " + rect2 + " is contained in " + rect1 + "?\t" +
               rect1.contains(rect2));
   System.out.println("Rectangle " + rect1 + " is contained in " + rect2 + "?\t" +
               rect2.contains(rect1));
 }
}
```

Exercise1

This exercise is similar to Exercise0B. You will write another Rectangle class, but this time you need to define two methods: *contains* and *touches*. Please see the diagrams below. Note that the data fields are also now of type double and not int. Examine the diagrams closely. *Contains* now only refers to within the rectangle, and *touches* refers to the outside edge.

Define the Rectangle class having:

- Double data fields named *xpos*, *ypos* (that specify the top left corner of the Rectangle), *width* and *height*. (as in Examples 0A and 0B, assume that the Rectangle's sides are parallel to the x and y axes. See example below). Recall that the upper left corner is the coordinate (0,0). X-values increase to the right, and y-values increase moving down.



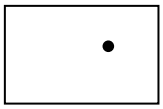
- A constructor that creates a rectangle with the specified *xpos*, *ypos*, *width*, and *height*.
- Get and set methods for all the instance variables.
- A method *contains(double px, double py)* that returns true if the point px, py is inside this Rectangle, false otherwise.
- A method *touches (double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

- A method *touches(Rectangle r)* that returns true if the specified rectangle touches this Rectangle, false otherwise

You may add other methods if necessary. Focus on method reuse. You spent all this time writing contains and touches methods. Use them.

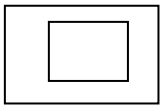
Study the figures below to understand when the contains method should return true, and when the touches method should return true. If the rectangles overlap, both methods should return false.

Point is contained but not touching

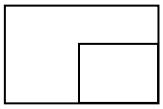


Point is touching but not contained

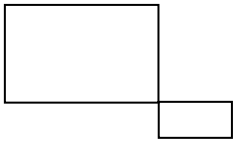
Rectangle is contained but not touching



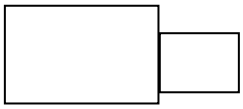
Rectangle is touching but not contained



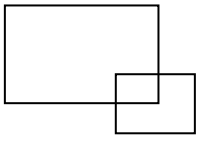
Rectangle is touching but not contained



Rectangle is touching but not contained



Rectangle is neither contained nor touching



Write a test program called **Exercise1.java** that accepts input in the following format:

- Line 1 will be an integer: the number of tests to expect
- All following lines will consist of doubles separated by whitespace. Each line represents a Rectangle.
- **You must use the provided test case text file *Exercise 1 Testcase input* as input and include the output in your submission. Name your output text file *Exercise1StudentOutput.txt*.**

Sample inputs and outputs:

Exercise1.in	*Exercise1.out*
2 1.0 1.0 5.0 5.0 2.0 2.0 3.0 3.0 1.0 1.0 5.0 5.0 2.0 2.0 4.0 4.0	The first Rectangle's top corner is: 1.0, 1.0 It's width is: 5.0 It's height is: 5.0 The second Rectangle's top corner is: 2.0, 2.0 It's width is: 3.0 It's height is: 3.0 The contains method returns: true The touches method returns: false The first Rectangle's top corner is: 1.0, 1.0 It's width is: 5.0 It's height is: 5.0 The second Rectangle's top corner is: 2.0, 2.0 It's width is: 4.0 It's height is: 4.0 The contains method returns: false The touches method returns: true

Exercise 2

This exercise will require you to apply what you've just learned in a slightly different domain. You will write Circle class, and define methods for *contains* and *touches*. Please see the diagrams below. Note that the data fields are of type double and not int.

Define a Circle class having:

- Two double data fields named *xpos* and *ypos* that specify the center of the circle.
- A double data field *radius*.
- A constructor that creates a circle with the specified *xpos*, *ypos* and *radius*.
- Three Getter methods, one each for *xpos*, *ypos* and *radius*.
- Two Setter methods: one for setting the center and one for setting the radius
- A method *getArea()* that returns the area of the circle.
- A method *getCircumference()* that returns the circumference of the circle.

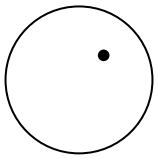
- A method *contains(double x, double y)* that returns true if the specified point (x,y) is inside this circle, false otherwise.
- A method *touches(double x, double y)* that returns true if the specified point (x,y) touches the circle.
- A method *contains(Circle c)* that returns true if the specified circle is completely inside this circle (similar to the case of the rectangle), false otherwise.
- A method *touches(Circle c)* that returns true if the specified circle touches this circle (similar to the case of the rectangle), false otherwise.

You may use an approximation of pi. 3.14 is sufficient. Recall Area = pi*radius*radius, Circumference = 2*pi*radius.

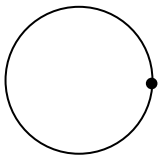
Hint: One way to implement the contains and touches methods is to find the distance between the two centers and compare that with the sum of the two radii. The distance between two points p1=(x1, y1) and p2 = (x2,y2) is given by the square root of $(x2-x1)^2 + (y2-y1)^2$

Study the figures below 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.

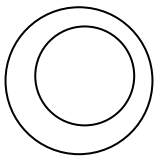
Point is contained but not touching



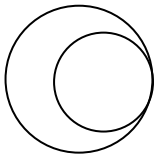
Point is touching but not contained



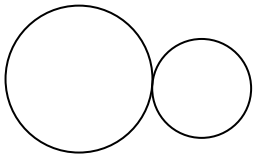
Circle is contained but not touching



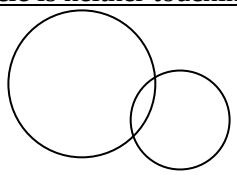
Circle is touching but not contained



Circle is touching but not contained



Circle is neither touching nor contained



Write a test program called Exercise2.java that accepts input in the following format:

- Line 1 will be an integer: the number of tests to expect
- All following lines will consist of doubles separated by whitespace. Each line represents a Circle.
- **You must use the provided test case text file *Exercise 2 Testcase input* as input and include the output in your submission. Name your output text file *Exercise2StudentOutput.txt*.**

Sample inputs and outputs:

Exercise2.in	*Exercise2.out*
2 5.0 5.0 4.0 10.0 2.0 3.0 4.0 2.0 1.0 2.0 2.0 1.0	The first Circle's centre is: 5.0, 5.0 It's radius is: 4.0 It's area is: 50.2 It's circumference is: 25.1 The second Circle's centre is: 10.0, 2.0 It's radius is: 3.0 It's area is: 28.3 It's circumference is: 18.8 The 'contains' method returns: false The 'touches' method returns: false The first Circle's centre is: 4.0, 2.0 It's radius is: 1.0 It's area is: 3.1 It's circumference is: 6.3 The second Circle's centre is: 2.0, 2.0 It's radius is: 1.0 It's area is: 3.1 It's circumference is: 6.3 The 'contains' method returns: false The 'touches' method returns: true