CSCI 2110 Data Structures and Algorithms
Fall 2023
Laboratory No. 1
Week of September 11-15

Due (on Brightspace): Sunday, September 17, 11.59 PM

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*, *IntelliJ,* or *NetBeans*.

Marking Scheme: Throughout this course, we will be focusing on the complexity of algorithms and consequently, the efficiency of programs. In your first-year courses, the 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.
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.

Suggested coding practice: 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).
You will be provided with test cases for each exercise and the expected outputs. Copy and paste the input into your IDE at runtime. Compare the output to the expected output. In addition, you may wish to create your own input tests and check the output.

Exercise0A (Review from lectures – 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.

```java
//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 from lectures – 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.

```
//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()));
  }
}
```

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

Exercise 1
In this exercise you are required to modify the demo class in Exercise0B such that it accepts a number of test cases provided by the user. Write a demo class called Exercise1.java. It must accept input in the following format:

The first line is an integer that indicates the number of test cases.
The next line will contain four integers that indicate the parameters for the first rectangle (xpos, ypos, width and height, respectively) in the first test case.
The next line will contain four integers that indicate the parameters for the second rectangle
Similarly, the next two lines will contain the parameters for the first and second rectangle, respectively, for the second test case.
And so on.
You may assume that the input is error-free, that is, the number of lines and the parameters will match the number of test cases.

As an example, the input is the following five lines:

2	← number of test cases is 2
1 1 5 5	← Values of rect1 for test case 1
2 2 3 3	← Values of rect2 for test case 1
1 1 5 5	← Values of rect1 for test case 2
2 2 10 10	← Values of rect2 for test case 2

The demo class must test whether the second rectangle (rect2) is contained within the first rectangle (rect1) for each test case in the manner shown below. For example, if you copy and paste the above integers into your IDE, the expected output is:

```
Test case: 1
Rectangle 1: [xpos= 1,ypos= 1] width: 5,height: 5
Rectangle 2: [xpos= 2,ypos= 2] width: 3,height: 3
Is Rectangle 2 contained in Rectangle 1? true

Test case: 2
Rectangle 1: [xpos= 1,ypos= 1] width: 5,height: 5
Rectangle 2: [xpos= 2,ypos= 2] width: 10,height: 10
Is Rectangle 2 contained in Rectangle 1? false
```

Here's another input and expected output:

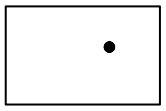
```
4
10 10 100 300
15 15 50 50
10 10 50 50
15 15 100 300
10 10 10 10
20 20 20 20
10 10 50 50
10 10 50 50
```

```
Test case: 1
Rectangle 1: [xpos= 10,ypos= 10] width: 100,height: 300
Rectangle 2: [xpos= 15,ypos= 15] width: 50,height: 50
Is Rectangle 2 contained in Rectangle 1? true

Test case: 2
Rectangle 1: [xpos= 10,ypos= 10] width: 50,height: 50
Rectangle 2: [xpos= 15,ypos= 15] width: 100,height: 300
Is Rectangle 2 contained in Rectangle 1? false

Test case: 3
Rectangle 1: [xpos= 10,ypos= 10] width: 10,height: 10
Rectangle 2: [xpos= 20,ypos= 20] width: 20,height: 20
Is Rectangle 2 contained in Rectangle 1? false

Test case: 4
Rectangle 1: [xpos= 10,ypos= 10] width: 50,height: 50
Rectangle 2: [xpos= 10,ypos= 10] width: 50,height: 50
Is Rectangle 2 contained in Rectangle 1? true
```

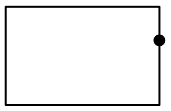
Complete the code for Exercise1.java and run and test it for the above two input cases.

Exercise 2: In this exercise, you will write a class called Rectangle2.java. This class is similar to Rectangle1.java but with two changes:
a) **Modify the contains method** so that it will return true only if the specified rectangle is fully contained in this rectangle (that is, if it touches any side then it is not contained).
b) **Add a method `touches(Rectangle1 r)`** that returns true if the specified rectangle object touches this rectangle object.
Study the figures below to understand when the contains method should return true, and when the touches method should return true. The touches method returns true only when one of the corners or the sides touch. If the rectangles overlap, they are not touching.

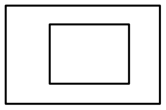
<u>Point is not touching</u>



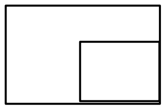
<u>Point is touching</u>



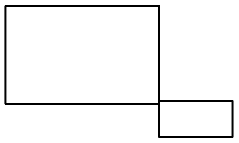
Rectangle is not touching but contained



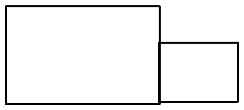
Rectangle is touching and NOT contained



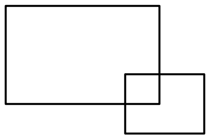
Rectangle is touching but not contained



Rectangle is touching but not contained



Rectangle is not touching and not contained



Write a test program called Exercise2.java that accepts input in a format similar to Exercise 1. For instance, for the following input

```
2
1 1 5 5
2 2 3 3
1 1 5 5
2 2 4 4
```

```
Test case: 1
Rectangle 1: [xpos= 1,ypos= 1] width: 5,height: 5
Rectangle 2: [xpos= 2,ypos= 2] width: 3,height: 3
Is Rectangle 2 contained in Rectangle 1?true
Is Rectangle 2 touching Rectangle 1? false

Test case: 2
Rectangle 1: [xpos= 1,ypos= 1] width: 5,height: 5
Rectangle 2: [xpos= 2,ypos= 2] width: 4,height: 4
Is Rectangle 2 contained in Rectangle 1?false
Is Rectangle 2 touching Rectangle 1? true
```

Here's another input test and the expected output:

```
4
10 10 100 300
15 15 50 50
10 10 50 50
15 15 100 300
10 10 10 10
20 20 20 20
10 10 50 50
10 10 50 50
```

```
Test case: 1
Rectangle 1: [xpos= 10,ypos= 10] width: 100,height: 300
Rectangle 2: [xpos= 15,ypos= 15] width: 50,height: 50
Is Rectangle 2 contained in Rectangle 1?true
Is Rectangle 2 touching Rectangle 1? false

Test case: 2
Rectangle 1: [xpos= 10,ypos= 10] width: 50,height: 50
Rectangle 2: [xpos= 15,ypos= 15] width: 100,height: 300
Is Rectangle 2 contained in Rectangle 1?false
Is Rectangle 2 touching Rectangle 1? false

Test case: 3
Rectangle 1: [xpos= 10,ypos= 10] width: 10,height: 10
Rectangle 2: [xpos= 20,ypos= 20] width: 20,height: 20
Is Rectangle 2 contained in Rectangle 1?false
Is Rectangle 2 touching Rectangle 1? true

Test case: 4
Rectangle 1: [xpos= 10,ypos= 10] width: 50,height: 50
Rectangle 2: [xpos= 10,ypos= 10] width: 50,height: 50
Is Rectangle 2 contained in Rectangle 1?false
Is Rectangle 2 touching Rectangle 1? true
```

Exercise 3
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 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 add other methods if necessary.

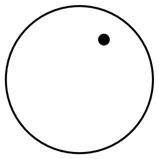
Hint: One way to implement the contains and touches methods is to find the distance between the two centers. The distance between two points p1=(x1, y1) and p2 = (x2,y2) is given by the square root of $(x2-x1)^2 + (y2-y1)^2$

Now compare this distance to the difference between the two radii. If the distance is less than the absolute difference, then the specified circle is contained. Note: You may have to deal separately with the special case when the two circles are identical.

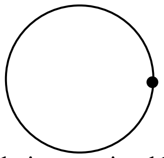
Similarly, if the distance between the centres is equal to the sum of the two radii or if they equal to the absolute difference between the two radii, then they touch each other (externally or internally).

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.

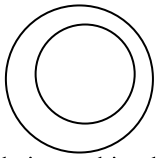
Point is contained but not touching



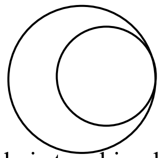
Point is touching but not contained



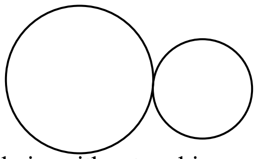
Circle is contained but not touching



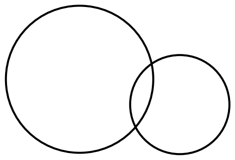
Circle is touching but not contained



Circle is touching but not contained



Circle is neither touching nor contained



Write a test program called Exercise3.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. Test Circles in pairs and provide outputs.

Test whether the first Circle contains or touches the second. Test a variety of cases, including all illustrated cases.

Sample inputs and outputs:

First example

```
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
```

```
Test case: 1
The first Circle's centre is at: 5.0,5.0
The first Circle's radius is :4.0 units
The second Circle's centre is at: 10.0,2.0
The second Circle's radius is :3.0 units
Second circle contained in first circle?: false
Second circle touches first circle? : false

Test case: 2
The first Circle's centre is at: 4.0,2.0
The first Circle's radius is :1.0 units
The second Circle's centre is at: 2.0,2.0
The second Circle's radius is :1.0 units
Second circle contained in first circle?: false
Second circle touches first circle? : true
```

Second Example

```
2
5.0 5.0 4.0
5.0 5.0 3.0
5.0 5.0 4.0
5.0 5.0 4.0
```

```
Test case: 1
The first Circle's centre is at: 5.0,5.0
The first Circle's radius is :4.0 units
The second Circle's centre is at: 5.0,5.0
The second Circle's radius is :3.0 units
Second circle contained in first circle?: true
Second circle touches first circle? : false

Test case: 2
The first Circle's centre is at: 5.0,5.0
The first Circle's radius is :4.0 units
The second Circle's centre is at: 5.0,5.0
The second Circle's radius is :4.0 units
Second circle contained in first circle?: false
Second circle touches first circle? : true
```

Submission: All submissions are through Brightspace. Instructions will also be given in the first lab.

What to submit:
One ZIP file containing the following source codes (files with .java suffixes):
1. Rectangle1.java
2. Exercise1.java
3. Rectangle2.java
4. Exercise2.java
5. Circle.java
6. Exercise3.java
7. A text file containing sample inputs and outputs for Exercise1.java, Exercise2.java and Exercise3.java

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.

Late Submission Penalty: The lab is due on **SUNDAY** at 11.59 PM. Late submissions up to 5 hours (4.59 AM on Sunday) will be accepted without penalty. After that, there will be a 10% late penalty per day on the mark obtained. For example, if you submit the lab on Monday at 12 noon and your score is 8/10, it will be reduced to 7.2/10. Submissions past two days (48 hours) after the grace submission time, that is, submission past 4.59 AM Wednesday will not be accepted.