CSCI 2110 Data Structures and Algorithms
Fall 2024
Laboratory No. 2
Week of September 16-20

Due (on Brightspace): Sunday, September 22, 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). You may use any IDE of your choice – IntelliJ, Eclipse, NetBeans, VSCode, etc.

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.

Commenting your code: Your code should be clear and easy to understand. While extensive comments are not required, you are expected to include brief, meaningful comments to explain complex logic or any non-obvious sections of your code. Use inline comments to clarify key parts of the code. For methods and classes, use Javadoc to briefly describe their purpose, parameters, and return values. This ensures others can understand your code's intent quickly.

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).

Submission: One zip file containing the source codes (.java files) for all classes, and sample outputs. Please see details at the end of this document.

Submission Deadline: Sunday, September 22, 2024, 11.59 PM

Grace Time: Submissions will be accepted until 4.59 AM on Monday, September 23, 2024 without late penalty.

Late Penalty: Submissions received after the grace time will be subject to a 10% per day late penalty. The submission portal will close on Wednesday, September 25, 4.59 AM.

Dropping of Labs: Up to 2 labs can be dropped during the semester.

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.


```
//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. The toString method has also been simplified to display the Rectangle's details in the format (xpos, ypos, width, height). 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 that returns the rectangle's details in the format
  //(xpos, ypos, width, height)

  public String toString() {
    return "(" + xpos + ", " + ypos + ", " + width + ", " + 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("Is Point (30,40) is contained in Rectangle" + rect1 + "?\n" +
                        rect1.contains(30,40) + "\n");
    System.out.println("Is Point (10,20) is contained in Rectangle" + rect1 + "?\n" +
                       rect1.contains(10,20) + "\n");
    System.out.println("Is Point (4,3) is contained in Rectangle" + rect1 + "?\n" +
                       rect1.contains(4,3)+ "\n");
    System.out.println("Is Rectangle " + rect2 + " is contained in Rectangle" + rect1 + "?\n"
+ rect1.contains(rect2) + "\n");
    System.out.println("Is Rectangle " + rect1 + " is contained in Rectangle" + rect2 + "?\n"
+ rect2.contains(rect1) +"\n");
   }
}
```

The output for the above code is:

```
Is Point (30,40) is contained in Rectangle(10, 20, 300, 400)?
true

Is Point (10,20) is contained in Rectangle(10, 20, 300, 400)?
true

Is Point (4,3) is contained in Rectangle(10, 20, 300, 400)?
false

Is Rectangle (15, 25, 100, 100) is contained in Rectangle(10, 20, 300, 400)?
true

Is Rectangle (10, 20, 300, 400) is contained in Rectangle(15, 25, 100, 100)?
false
```

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, namely, xpos, ypos, width and height, 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: (1, 1, 5, 5)
Rectangle 2: (2, 2, 3, 3)
Is Rectangle 2 contained in Rectangle 1? true

Test case: 2
Rectangle 1: (1, 1, 5, 5)
Rectangle 2: (2, 2, 10, 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: (10, 10, 100, 300)
Rectangle 2: (15, 15, 50, 50)
Is Rectangle 2 contained in Rectangle 1? true

Test case: 2
Rectangle 1: (10, 10, 50, 50)
Rectangle 2: (15, 15, 100, 300)
Is Rectangle 2 contained in Rectangle 1? false

Test case: 3
Rectangle 1: (10, 10, 10, 10)
Rectangle 2: (20, 20, 20, 20)
Is Rectangle 2 contained in Rectangle 1? false

Test case: 4
Rectangle 1: (10, 10, 50, 50)
Rectangle 2: (10, 10, 50, 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. Save the outputs in a text file.
At the top of Rectangle1.java and Exercise1 .java file, put the following information in comments.
// Full Name: [Your Full Name]
// ID Number: [Your Banner ID]

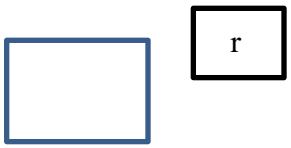
Exercise 2: In this exercise, you will write a class called Rectangle2.java. This class is similar to Rectangle1.java but with the following changes. It must have three methods, namely, contains, touches, and overlaps with the following behaviour:

contains(Rectangle2 r): This method will return true if all four corners of r are inside or touching this rectangle.

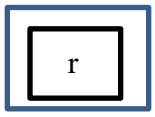
touches(Rectangle2 r): This method will return true if any corner or side of rectangle r touches the sides of this rectangle, but does not result in a full containment or overlap.

overlaps(Rectangle2 r): This method will return true if one or more corners of r are inside this rectangle, but not all of the corners (partial overlap).

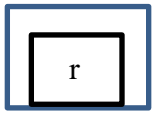
The following diagrams illustrate the contains, touches, and overlaps methods.



contains(r) : false
touches(r) : false
overlaps(r): false

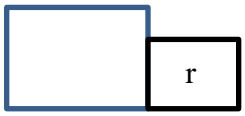


contains(r) : true
touches(r) : false
overlaps(r): false

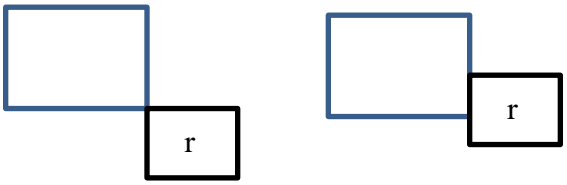


contains(r) : true
touches(r) : true
overlaps(r): false

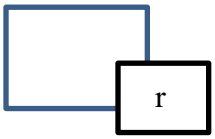
Note: there could be other similar cases where contains and touches are true: left edge touching, right edge touching, top edge touching, or all four sides touching (two identical rectangles)



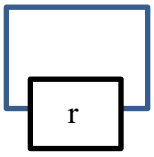
contains(r) : false
touches(r) : true
overlaps(r): false



Same as above



contains(r) : false
touches(r) : false
overlaps(r): true



contains(r) : false
touches(r) : false
overlaps(r): true

Note: With the above definition, contains and overlaps cannot be both true simultaneously.
Similarly, touches and overlaps cannot be both true simultaneously.
Similarly, contains, touches, and overlaps cannot all be true simultaneously.

Write a test program called Exercise2.java that accepts input in a format similar to Exercise 1. For instance, for the following input (the arrows and explanations are not part of the input).

```
5
0 0 10 10    <-- Rectangle 1 for test case 1
20 20 5 5    <-- Rectangle 2 for test case 1
0 0 20 20    <-- Rectangle 1 for test case 2
5 5 5 5      <-- Rectangle 2 for test case 2
0 0 20 20    <-- Rectangle 1 for test case 3
0 0 20 20    <-- Rectangle 2 for test case 3
0 0 10 10    <-- Rectangle 1 for test case 4
10 0 5 5     <-- Rectangle 2 for test case 4
0 0 20 20    <-- Rectangle 1 for test case 5
15 15 10 10  <-- Rectangle 2 for test case 5
```

The expected output is:

```
Test case: 1
Rectangle 1: (0, 0, 10, 10)
Rectangle 2: (20, 20, 5, 5)
Is Rectangle 2 contained in Rectangle 1? false
Does Rectangle 2 touch Rectangle 1? false
Does Rectangle 2 overlap with Rectangle 1? false

Test case: 2
Rectangle 1: (0, 0, 20, 20)
Rectangle 2: (5, 5, 5, 5)
Is Rectangle 2 contained in Rectangle 1? true
Does Rectangle 2 touch Rectangle 1? false
Does Rectangle 2 overlap with Rectangle 1? false

Test case: 3
Rectangle 1: (0, 0, 20, 20)
Rectangle 2: (0, 0, 20, 20)
Is Rectangle 2 contained in Rectangle 1? true
Does Rectangle 2 touch Rectangle 1? true
Does Rectangle 2 overlap with Rectangle 1? false

Test case: 4
Rectangle 1: (0, 0, 10, 10)
Rectangle 2: (10, 0, 5, 5)
Is Rectangle 2 contained in Rectangle 1? false
Does Rectangle 2 touch Rectangle 1? true
Does Rectangle 2 overlap with Rectangle 1? false

Test case: 5
Rectangle 1: (0, 0, 20, 20)
Rectangle 2: (15, 15, 10, 10)
Is Rectangle 2 contained in Rectangle 1? false
Does Rectangle 2 touch Rectangle 1? false
Does Rectangle 2 overlap with Rectangle 1? true
```

Here's another input 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: (10, 10, 100, 300)
Rectangle 2: (15, 15, 50, 50)
Is Rectangle 2 contained in Rectangle 1? true
Does Rectangle 2 touch Rectangle 1? false
Does Rectangle 2 overlap with Rectangle 1? false

Test case: 2
Rectangle 1: (10, 10, 50, 50)
Rectangle 2: (15, 15, 100, 300)
Is Rectangle 2 contained in Rectangle 1? false
Does Rectangle 2 touch Rectangle 1? false
Does Rectangle 2 overlap with Rectangle 1? true

Test case: 3
Rectangle 1: (10, 10, 10, 10)
Rectangle 2: (20, 20, 20, 20)
Is Rectangle 2 contained in Rectangle 1? false
Does Rectangle 2 touch Rectangle 1? true
Does Rectangle 2 overlap with Rectangle 1? false

Test case: 4
Rectangle 1: (10, 10, 50, 50)
Rectangle 2: (10, 10, 50, 50)
Is Rectangle 2 contained in Rectangle 1? true
Does Rectangle 2 touch Rectangle 1? true
Does Rectangle 2 overlap with Rectangle 1? false
```

Run Exercise2.java for the above test cases and save the outputs in a text file.
At the top of Rectangle2.java and Exercise2 .java file, put the following information in comments.
// Full Name: [Your Full Name]
// ID Number: [Your Banner ID]

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. A text file containing sample inputs and outputs for Exercise1.java and Exercise2.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.