CSCI 2110 Data Structures and Algorithms
Fall 2025
Laboratory No. 1
Week of September 29 – October 3

Due (on Brightspace): Sunday, October 5, 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.*

Since Tuesday, September 30th is a holiday and several lab sessions are held on Tuesday, this is primarily a "do-at-home" lab. If you need assistance, please feel free to walk into the labs B03 (Monday 1-2.30) or B01 (Friday 1-2.30) in Mona Campbell Room 1201. TAs will be there to assist you.

Submission Deadline: Sunday, October 5, 2025, 11:59 PM

Grace Time: Submissions will be accepted until 4:59 AM on Monday, October 6, without late penalty.

Late Penalty: Submissions received after the grace time will be subject to a 10% per day late penalty for up to two days.

Dropping of Labs: Up to 2 labs can be dropped during the semester. No SDA submission needed.

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 the code, copy it and test it.

```
//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 the code, copy it and test it.

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

Note: Exercises 0A and 0B are simple starter programs demonstrated in class. In the demo, the `main` method uses hard-coded parameters for convenience. In your exercises, however, you should not hard-code values—always use console input instead. This is a best practice and will make your programs more flexible.

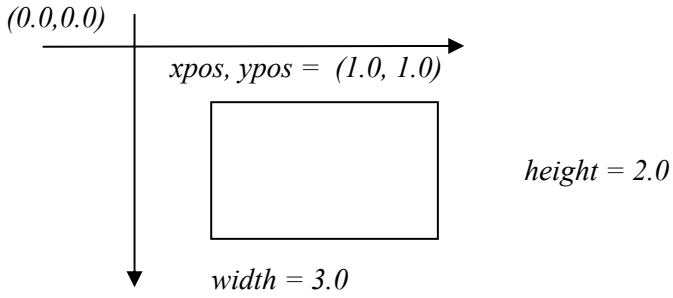
Exercise 1: This exercise is an extension of Exercise 0. You will write three classes, a Point class, a Rectangle class and a Circle class, and in the Circle class, you will write two contains methods, one to check if a point is contained in the Circle and another to check if the Rectangle is contained in a Circle.

First define a Point class as follows:

- Double data fields px and py that specify the x and y coordinates of the point.
- Constructor that sets an instance of the Point with the given px and py values.
- Getters and Setters
- toString method to display the data fields.

Next define a Rectangle class as follows:

- 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).

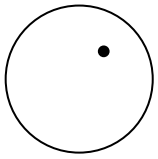


- A constructor that creates a rectangle with the specified xpos, ypos, width, and height.
- Get and set methods for all the instance variables.
- toString method to display the data fields.

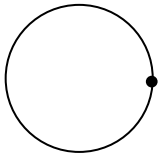
Next define a Circle class as follows:

- 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
- Method contains(Point p) that returns true if a given Point is inside this Circle object. See examples below. It should return true even if the Point touches the Circle.

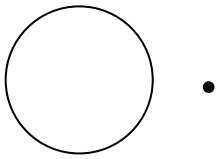
Point is contained



Point is contained

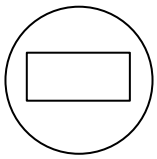


Point is not contained

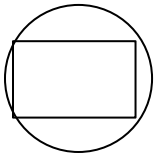


- Method contains(Rectangle r) that returns true if a given Rectangle object is fully contained in this Circle object. The rectangle may touch the circle (edges/corners), but must not cross outside it. See examples below.

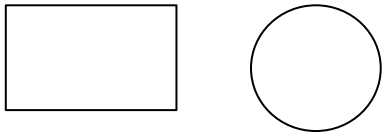
Rectangle is contained



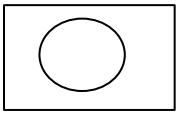
Rectangle is contained



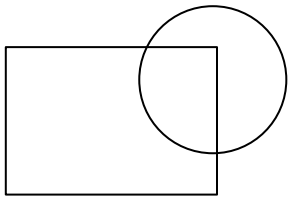
Rectangle is not contained



Rectangle is not contained



Rectangle is not contained



Algorithm Hint:
Using the contains(Point p) method in the Circle class, check if the four corners of the Rectangle are contained in the Circle. If yes, then return true; else return false.

Important Note about comparing Doubles: Note that you will be comparing doubles. If you use the == to compare two doubles, it may produce unexpected outcome. For example, (1.1+2.2)= =3.3 returns false. It is common to define a small threshold value EPSILON (usually set to 10^-8). So you would write your equals method to return true if the result is + or – EPSILON.

Here's how you would do it: Define a small threshold, e.g., `EPSILON = 1e-8`. You can define it as `private static final double EPSILON = 1e-8;` in your class file.
When comparing doubles `a` and `b`, treat them as equal if `Math.abs(a - b) <= EPSILON`

Write a client program called TestGeometry.java with a main method that prompts the user to enter the center and radius of a Circle object, then the x and y for a Point object, and the xpos, ypos, width and height for a Rectangle object, creates these objects and displays if the Point is contained in the Circle and if the Rectangle is contained in the Circle.

Use the following template in your code:

```
import java.util.Scanner;

public class TestGeometry {
    public static void main(String[] args) {
     Scanner sc = new Scanner(System.in);

      // Read a circle
      // TO-DO

      // Read a point
      // TO-DO

      // Read a rectangle (top-left x y, width, height)
      // TO-DO

      // Display results
      // TO-DO

      sc.close();
  }
}
```

Here's a sample screen dialog:

```
Enter circle center (x y) and radius: 5.0 5.0 10.0
Enter a point (x y): 2.0 3.0
Enter rectangle (xpos ypos width height): 10.0 10.0 50.0 40.0

Circle[center=(5.0, 5.0), radius=10.0]
Point (2.0, 3.0) contained in circle? true
Rectangle[xpos=10.0, ypos=10.0, width=50.0, height=40.0]
Rectangle contained in circle? false
```

Run your code for three sample runs. Cut and paste the outputs into a text file.

Exercise 2

A complex number has two parts, a real part a and an imaginary part b, and is written as a + bi. For example, 1.0 + 3.0i is a complex number with the real part being 1.0 and the imaginary part being 3.0. In this exercise, you will write a class to model complex numbers and manipulate them.

As in Exercise 1, **Important Note about comparing Doubles:** Define a small threshold, e.g., `EPSILON = 1e-8`. When comparing doubles `a` and `b`, treat them as equal if `Math.abs(a - b) <= EPSILON`

Design a class called Complex with the following properties:

- Two instance variables `real` and `imag` – both `double`
- A constructor that creates an instance of a Complex object with the given real and imag values.
- A default constructor that creates an instance of a Complex object 0.0 + 0.0i.
- Get and set methods
- A toString method that returns the String (a + bi) where a and b are the real and imag parts, respectively.
- Methods isReal() that returns true if the complex number is real (that is, the imag value is 0.0).
- Method isImaginary() that returns true if the complex number is imaginary (that is, the real value is 0.0).
- Method equals(Complex other) that returns true if this instance of the Complex number is equal to the given Complex number (other). Note: Two complex numbers are equal if their real parts are equal and their imaginary parts are equal.
- Method magnitude() that returns the magnitude of this complex number. Note: magnitude(a+bi) = Math.sqrt(a*a + b*b)
- Method Complex add(Complex other) that adds this instance of the Complex number to the given instance (other) and returns the result as a Complex number. Note: (a+bi)+(c+di) = (a+c) + (b+d)i
- Method Complex sub(Complex other) that subtracts the given instance of the Complex number (other) from this instance and returns the result as a Complex number. Note: (a+bi)-(c+di)=(a-c)+(b-d)i
- Method Complex multiply(Complex other) that multiplies this instance of the Complex number with the given instance (other) and returns the result as a Complex number.Note: (a+bi)*(c+di)= (ac-bd)+(ad+bc)i

Test all the methods in the Complex class by using the client program given below.

```
import java.util.Scanner;

public class ComplexTest {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter complex number 1 (real imag): ");
        double a1 = sc.nextDouble(), b1 = sc.nextDouble();
        Complex z1 = new Complex(a1, b1);

        System.out.print("Enter complex number 2 (real imag): ");
        double a2 = sc.nextDouble(), b2 = sc.nextDouble();
        Complex z2 = new Complex(a2, b2);

        System.out.println();
        System.out.println("Number 1 is " + z1);
        System.out.println(z1 + (z1.isReal() ? " is a pure real number" : "
is NOT a pure real number"));
        System.out.println(z1 + (z1.isImaginary() ? " is a pure imaginary
number" : " is NOT a pure imaginary number"));

        System.out.println();
        System.out.println("Number 2 is " + z2);
        System.out.println(z2 + (z2.isReal() ? " is a pure real number" : "
is NOT a pure real number"));
        System.out.println(z2 + (z2.isImaginary() ? " is a pure imaginary
number" : " is NOT a pure imaginary number"));

        System.out.println();
        System.out.println("The magnitude of " + z1 + " is " +
z1.magnitude());
        System.out.println("The magnitude of " + z2 + " is " +
z2.magnitude());
        System.out.println(z1 + " + " + z2 + " is " + z1.add(z2));
        System.out.println(z1 + " - " + z2 + " is " + z1.sub(z2));
        System.out.println(z1 + " * " + z2 + " is " + z1.multiply(z2));

        sc.close();
    }
}
```

A sample dialog is given below:

```
Enter complex number 1 (real imag): 3.0 4.0
Enter complex number 2 (real imag): 1.0 -2.0

Number 1 is (3.0 + 4.0i)
(3.0 + 4.0i) is NOT a pure real number
(3.0 + 4.0i) is NOT a pure imaginary number

Number 2 is (1.0 + -2.0i)
(1.0 + -2.0i) is NOT a pure real number
(1.0 + -2.0i) is NOT a pure imaginary number
```

```
The magnitude of (3.0 + 4.0i) is 5.0
The magnitude of (1.0 + -2.0i) is 2.23606797749979
(3.0 + 4.0i) + (1.0 + -2.0i) is (4.0 + 2.0i)
(3.0 + 4.0i) - (1.0 + -2.0i) is (2.0 + 6.0i)
(3.0 + 4.0i) * (1.0 + -2.0i) is (11.0 + -2.0i)
```

Run your code for three sample runs. Cut and paste the outputs into a text file.

WHAT TO SUBMIT:

One zip file containing all of the listed files:

YourBannerID_Lab1.zip
- Point.java
- Rectangle.java
- Circle.java
- TestGeometry.java
- Complex.java
- ComplexTest.java (copy the code given to you)
- SampleRuns.txt (Text file containing sample runs for both Exercise 1 and 2)

You must submit .java files that can be compiled and run by the TAs. Do not submit .class files.

Do not include `package` statements in your code. These cause compilation issues when TAs test your programs and have to be removed manually.

Make sure your Name and Banner ID appears in Comments in all the .java files.