Recommended Group Brainstorm (NO computers during this time)

Good programmers think before they begin coding. Part I of this assignment involves brainstorming with a group of peers with no computers to talk about a strategy for solving this week's lab. Breakup into groups based on how you're seated (3-4 people per group) and brainstorm about how to solve the problems below. Make sure everyone understands the problem and sketch out potential ways to move toward a solution. You may find it helpful to look over the required readings for this week.

Before you begin coding, you should be able to answer the following questions:

- Why is it good practice to make our instance variables private?
- What would happen if you used == to compare two objects?
- When writing an equals method, you should be comparing two objects – the one that called the method and the one that was passed in as the actual parameter. Inside the equals method definition, how do you access the attributes of each of these objects? You may want to use the example in the textbook for guidance.
- How do we compare doubles? (don't use ==)

Note: Brainstorms are to help you get started with the lab and get you in the habit of designing before you code. You won't submit them to eLC.

Introduction

This lab introduces you to fundamental concepts of Object Oriented Programming (OOP), arguably the dominant programming paradigm in use today. In the paradigm, a program consists of component parts (objects) that are independent of each other and that interact in order to achieve a desired result. At a very basic level, an object-oriented program consists of the following types of things.

Classes: The template or blueprint from which objects are created.
Objects: The instantiation of a class. Once a class (the template) has been defined, many instances of it can be created.
Methods: The named procedures or functions defined for an object. A method accepts input arguments and contains statements that are executed when the method is invoked. Some methods return values.
Instance Variables: Variables that are initialized when an object is created. Each instance of a class can have distinct values for these variables.

In this lab, you will practice working with these fundamentals. A skeleton of code has been written for you defining the basic class and method structure that you are to use. You will then add code to the existing skeleton methods so that the classes function as intended. You are then to use a testing program illustrate how the component parts of what you have created interact.

Lab Objectives

By the end of the lab, you should be able to create classes utilizing:

- access modifiers;
- instance variables (also called fields);

- methods which return values and void methods (which do not return any value);
- methods calling other methods;
- accessor and mutator methods;
- The **`equals()`** method.

What to Submit

The files **`Circle.java`** and **`CircleTester.java`** should be submitted to eLC for grading.

Instructions

The file **`Circle.java`** contains a partial definition for a class representing a circle. Save it to your computer and study it to see what methods it contains. Then complete the ***Circle*** class as described below. Note that you won't be able to test your methods until you complete the test client class called **`CircleTester.java`**.

1. You must include a comment stating your name, the date, program purpose, and containing a statement of academic honesty. When writing, you must also abide by the Java formatting and naming conventions.
2. Complete the **`Circle`** class following the instructions detailed below:
   - **a.** Declare four *private* instance variables: **name**, **`x`**, **`y`** and **`radius`**. The instance variables **`x`** and **`y`** represent the coordinates of the center of the circle. Note that you should not declare any other instance variables other than these.
   - **b.** Fill in the code for method **`toString`**, which should **return** a "string" containing values for the name of the circle, the center of the circle, **`x`** and **`y`**, and the radius of the circle. **Note**: this method does not print the string – it returns the string. The returned string value should be formatted as:

     ```
     name: name
     center: (x,y)
     radius: r
     ```

   - **c.** Fill in the code for the accessor (getter) methods: **getName**, **`getX`**, **`getY`** and **`getRadius`** properly. Note that accessors are also called "getter" methods.
   - **d.** Fill in the code for the mutator (setter) methods: **setName**, **`setX`** and **`setY`** properly **.** Mutators are also called "setter" methods.
   - **e.** Fill in the code for the mutator(setter) method: **`setRadius`** properly. Note that:
     i. If the new radius value passed as a parameter to the method is **not valid,** than the method should leave the original radius *unchanged*.
     ii. A radius is valid only when it is greater than or equal to 0.
   - **f.** Fill in the code for the method **`area`** that returns the area of the circle, computed by

     `PI * radius`^{2} //Use the value of PI provided by the constant **`Math.PI`**.

g. Fill in the code of the method **`perimeter`** that returns the perimeter of the circle computed by

```
2 * PI * radius
```

h. Fill in the code of the method **`diameter`** that returns the diameter of the circle.

i. Fill in the code of the method **`isUnitCircle`** that returns **`true`** if the radius of the circle is **`1`** and is centered at the origin, i.e., the coordinates of the circle's center are (0, 0). Otherwise, this method returns **`false`**. Note: since you're comparing doubles, you should not use ==. Remember how to properly compare doubles from lecture.

3. The **`CircleTester.java`** file contains a driver program to try and test out your **`Circle`** class. Using the comments in the **`CircleTester.java`** file, finish the file so that it does the following:

   a. Display the name, center and radius of **`circle1`**.

   b. Set the radius of **`circle2`** to **`5.3`**.

   c. Display the name, center and radius of **`circle2`**.

   d. Display the diameter, area and perimeter of **`circle1`**.

   e. Display the diameter, area and perimeter of **`circle2`**.

   f. Display a message that indicates whether or not **`circle1`** is a unit circle.

   g. Display a message that indicates whether or not **`circle2`** is a unit circle.

4. Compile and test your **`Circle`** and **`CircleTester`** files, and verify your code is working properly. Then, set breakpoints in your methods, and the first line of the **`CircleTester`** class, and observe the flow of control within an Object-Oriented program.

5. Add to your **`Circle`** class the following methods:

   a. **`public boolean equals(Circle anotherCircle)`**

   This method returns **`true`** when the radius and centers of both circles are the same; otherwise, it returns **`false`**. This method can be implemented in one line. Remember not to compare doubles using ==.

   b. **`public double distance(Circle anotherCircle)`**

   This method returns the distance between the centers of the circle executing the method and **`anotherCircle`**. Let (x, y) and (x_a, y_a) be the centers of the circle executing the method and **`anotherCircle`** respectively, the distance between their centers is computed by

   $$\sqrt{(x - x_a)^2 + (y - y_a)^2}$$

   The **`Math`** class contains methods for computing both square roots and powers. Below are the definitions of the methods.

   ```
   //Returns a double value containing the square root of a double number.
   double sqrt(double number)
   ```

```
//Returns a double value of the first argument raised to the power of
the second argument.
double pow(double base, double exponent)
```

c. `public boolean isSmaller(Circle anotherCircle)`

The method **isSmaller** returns **true** when the circle executing the method (calling object) is smaller than the parameter (**anotherCircle**). Otherwise, it returns **false**. To see which is larger, use the diameter of each circle. You can use the diameter method written earlier to get this value from each object. Do this instead of using radius to practice calling methods within methods.

d. `public int compareTo(Circle anotherCircle)`

Works similar to the **compareTo** method in the String class. If the calling object is larger, the method returns a positive **1**. If the calling object is smaller than anotherCircle, it returns **-1**. Otherwise, it returns **0**. You may want to use the isSmaller method in your implementation.

e. `public boolean intersects(Circle anotherCircle)`

The method **`intersects`** returns **`true`** when the circle executing the method and **`anotherCircle`** have an intersecting area (one or more points enclosed by both circles); otherwise, it returns **`false`**.

Two circles intersect (by this lab's definition) if the distance between the centers of the two circles is less than the sum of their radius. Your method must call the method **`distance`** to obtain the distance between the two circles.

6. After you've added these methods, add at least three different tests for each method to your **`CircleTester`** class, and verify you have successfully written the above methods. Remember, your methods must work properly with **any input we can provide**, so test rigorously. For each test, you should output pass/fail instead of dumping values to the screen. An example can be found in the test file.
7. Check your method signatures using CircleMethodCheck.java (found on the course website). If you pass the tests, it means you are using the proper method signatures. **Passing these tests does not guarantee that you have implemented the methods correctly but it helps avoid you missing points due to invalid headers (signatures).** To use this class, simply put it into the src folder of your project and then run CircleMethodCheck.java.

Additional Requirements

These are things that make the graders lives easier, and ultimately, you will see in the real world as well. Remember the teaching staff does not want to touch your code after they gave you requirements; they want to see the perfect results they asked for! Here is a checklist of things you can **lose points** for:

- (10 points) If the source file(s)/class(es) are named incorrectly (case matters!)
- (10 points) If your source file(s) have a package declaration at the top

- (10 points) If any source file you submit is missing your Statement of Academic Honesty at the top of the source file. All submitted source code files must contain your Statement of Academic Honesty at the top of each file.
- (10 points) If you have more than one instance of Scanner in your program. Your program should only have one instance of Scanner.
- (15 points) Inconsistent I/O (input/output) that does not match our instructions or examples exactly (unless otherwise stated in an assignment's instructions). Your program's I/O (order, wording, formatting, etc.) must match our examples and instructions.
- (100 points) If the source file(s) are not submitted before the specified deadline's late period ends (48 hours after the deadline) or if they do not compile.
- (25 points) Late penalties will be deducted as per the course syllabus.
- If your (10 points) comments or (10 points) variables are "lacking"
    - Here, "lacking" means that you or a TA can find **any** lines of code or variables that take more than 10 seconds to understand, and there is no comment, or the variable name does not make sense (variable names like **b**, **bb**, **bbb**, etc. **will almost never be acceptable)**
- (10 points) Indentation is not consistent throughout your source code
    - Refresh your memory of indentation patterns in chapter 2 in the course textbook
    - Be careful of a combination of tabs and spaces in your files (use one or the other)!
- (100 points) If you use a non-standard Java library or class (StringBuilder class, Arrays class, any class in java.util.stream) or other code specifically disallowed by the lab/project.

If any of the above do not make sense to you, talk to a TA, or ask on Piazza!

eLC Submission and Grading

After you have completed and thoroughly tested your programs, upload **`Circle.java`** and **`CircleTester.java`** to ***eLC*** to receive credit. Always double check that your submission was successful on ***eLC***!

The lab will be graded according to the following guidelines.

- A score between 0 and 100 will be assigned.
- If the source file(s) are not submitted before the specified deadline's late period ends (48 hours after the deadline), or if they do not compile, a 0 is assigned.
- The program will be evaluated using both the **`CircleTester.java`** file and a separate testing file. Multiple instances of the Circle class will be created and their methods invoked.
- All instructions and requirements must be followed; otherwise, points maybe deducted.