CECS 277 LAB ABSTRACT FACTORY PATTERN

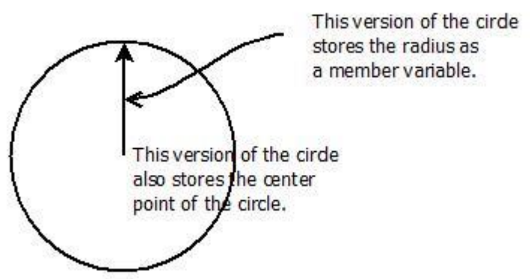
OBJECTIVE: Learn how to implement the Abstract Factory Pattern by augmenting an existing Factory Pattern implementation.

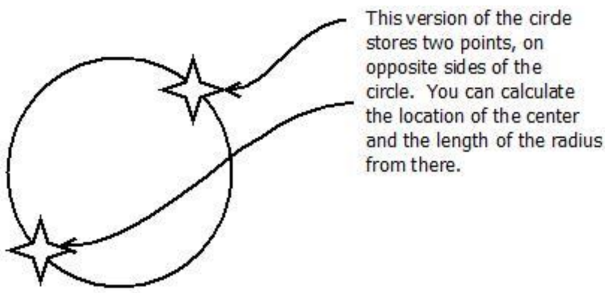
INTRODUCTION: Please remember the coding standards here.

You have the code that we went through in class today here. Do not panic at the number of classes. Most of them are quite small, and you do not really need to know the details of the bigger ones to get good value from this lab. The code that we went over in class today is here.

This working factory pattern presently has two factories: ConcreteFactoryPointDimensions, and ConcreteFactoryPoints. The GeometricShape objects constructed in the first one have a single point that locates the object, and then some sort of "dimension" that finishes off the definition. For the rectangle, that means the upper left hand corner of the rectangle, the width the length, and an angle to give you the orientation of the rectangle. In the case of the line segment, that is just the starting point and a vector. The other factory that we have cranks out GeometricObjects that rely strictly on a collection of one or more points to define themselves. That's pretty easy for the rectangle and the line segment.

Now we want to add another GeometricObject: the circle. In keeping with the theme that we have already established, The ConcreteFactoryPointDimensions version of the circle will be just the center point and the radius. Nothing exotic there. The ConcreteFactoryPoints will store its definition as two points: on opposite sides of the circle. In both cases, the circle's constructor will take in two points, but how that circle is stored in the object will differ. Perhaps a picture will help:


This version of the circle stores the radius as a member variable.
This version of the circle also stores the center point of the circle.
ConcreteFactoryPointDimensions


This version of the circle stores two points, on opposite sides of the circle. You can calculate the location of the center and the length of the radius from there.
ConcreteFactoryPoints

Just like the other GeometricObjects in this lab, the Circle will have the ability to calculate its perimeter, its area, provide its location (the center point) and will have a toString method for it.

Note that the `Point` class has a `between` method that you can use on the two points in the ConcreteFactoryPoints version of the circle to find the center of the circle. No, no, don't thank me so profusely, it's all part of the job, honest.

PROCEDURE:

CECS 277 LAB ABSTRACT FACTORY PATTERN

1. Create a new project with all of the files that I gave you, and make sure that you can create GeometricObject instances through the factory with the code that I give you.
2. Update the ShapeType enumeration to handle the new shape type.
3. Create your CircleAbstractClass class as an extension of the GeometricShape abstract class.
    a. You are going to want to provide a method for getArea, getPerimeter, getLocation at this level. Remember that you have getPoints that will come from one of your two concrete implementations of CircleAbstractClass, so use that.
    b. Do not forget your `toString()` method.
4. Create Circle2Points class
    a. This concrete class will just store a clone of the two points that the constructor takes in.
5. Create CirclePointRadius class
    a. This concrete class just stores the center point of the circle (calculated by finding the midpoint between the two given points) and the raidus.
    b. Note that both Circle2Points and CirlcePointRadius both take the same argument for their constructor: an array of just 2 points.
6. Update the ConcreteFacotyrPointDimensions and ConcreteFactoryPoints factories so that they can handle your new Circle objects.
7. Write a runner class (call it FactoryRunner) with a Main that will:
    a. Create a ConcreateFactoryPoints and ConcreteFactoryPointDimensions factory using the FactoryProvider class.
    b. Create GeometricObjects
        i. Rectangle
        ii. Line Segment
        iii. Circle
    c. For each object, display the points that you supply to the factory.
    d. For each object, print out the created object.

SAMPLE CONSOLE:

```
Testing the factory!
Rectangle- Points: Point: x value: 0.0 y value: 0.0 Point: x value: 5.0 y value: 0.0
Point: x value: 5.0 y value: -10.0 Point: x value: 0.0 y value: -10.0
Area: 50.0 Perimeter: 30.0
Line Segment - from: Point: x value: 0.0 y value: 0.0 to Point: x value: 5.0 y value:
0.0 Area: 0.0 perimeter: 5.0
Points and Dimensions
45 Degree rectangle
Rectangle- Points: Point: x value: -5.0 y value: 0.0 Point: x value:
8.881784197001252E-16 y value: 5.0 Point: x value: 10.000000000000004 y value: -5.0
Point: x value: 5.0000000000000036 y value: -10.0
Area: 100.00000000000001 Perimeter: 42.42640687119285
```

```
Line segment Point and Direction:
Line Segment - from: Point: x value: -5.0 y value: 0.0 to Point: x value:
8.881784197001252E-16 y value: 5.0 Area: 0.0 perimeter: 7.071067811865476
Points used to create circles: Points: Point: x value: 5.0 y value: 0.0 Point: x
value: 5.0 y value: -10.0
2 point circle: Circle - Located at: Point: x value: 5.0 y value: -5.0 of radius: 5.0
Radius and center circle: Circle - Located at: Point: x value: 5.0 y value: -5.0 of
radius: 5.0
Completed satisfactorily
```

WHAT TO TURN IN:

- Circle2Points.java
- CircleAbstractClass.java
- CirclePointRadius.java
- ConcreteFactoryPointDimensions.java
- ConcreteFactoryPoints.java
- FactoryRunner.java
- Your console output
- Your UML class diagram
    - This is going to get messy. Do not include the classes for the Line Segment and the Rectangle in the UML diagram. That should tidy it up a bit.