Object-Oriented Programming: Basics (Page 1 of 2)

The Circle class (page one) is the really important problem. The problems on page two are for those with more extensive previous OOP or Java experience.

1. Make a new Eclipse project called “shapes1”. or “oop-basics-shapes” or something similar. Your eventual goal will be to make a Circle class with various capabilities (see below), and a test routine that makes some circles and tests them out. Put your Circle class and your test routine in two separate classes, like this:

- **Circle.java**

```
public class Circle {
  public double radius;
  ...
}
```

- **CircleTest.java**

```
public class CircleTest {
  public static void main(String[] args) {
    Circle c = new Circle(...);
    ...
  }
}
```

Capabilities

- Give the Circle class a radius field of type double
- Give Circle a method that computes the area (πr^2)
- Give Circle a constructor to which you can pass the radius
- Have the constructor use the “this” variable

Notes

- Do not try to put in all four capabilities at once! Unless you have previous Java experience, I *strongly* recommend you build up to the solution in a piecemeal fashion, closely following the model of the Person class at the end of the notes. For example, first make a Circle class with a radius field only, and test it out from the main in CircleTest. Then add the getArea method, and test it. Then add in a constructor. Then test it out again. Then change the constructor to use the “this” variable, and test yet again. To reiterate, use the four Person examples from the end of the lecture as a model for both your code and the iterative development/testing process.
- The Circle class does not have a main, so you cannot execute it directly. You only directly run the CircleTest class (R-click, Run As, Java Application).

Object-Oriented Programming: Basics (Continued)

2. Make a program that creates an array of 100 circles, each with a random radius. Print out the sum of the areas of the circles. Also print the biggest and smallest areas. Hint: remember that in the two-step array allocation process, the following line only makes *space* for 100 circles (or, more technically, it allocates an array of 100 null Circle pointers), it does not *create* any circles:

```
Circle[] circles = new Circle[100];
```

To actually create the circles, you have to do a loop:

```
for(int i=0; i<circles.length; i++) {
  circles[i] = new Circle(...);
}
```

3. Create a Rectangle class that contains width and height fields. Also give it a getArea method. Again, make a few test cases.

4. Create a Square class with width and getArea. Then, give both Square and Circle *set*Area methods that let you specify a desired area. Make a few test cases.

5. Questions to ponder (as segues to next lecture):

- Suppose you create a method that takes a Rectangle as an argument. Now suppose you want to pass a Square to it (after all, squares are rectangles, aren't they?). Why won't it work? From what we know so far, how could you fix this problem?
- Since there is no particular relationship among Circle, Square, and Rectangle, what would you do if you wanted to make an array of mixed shapes, then loop down the array and sum up the areas?
- Suppose that, for efficiency reasons in the Circle class, you wanted to make an area instance variable, instead of recomputing it each time in the getArea method. So, instead of
  public double getArea() { return(Math.PI * radius * radius); }
  you instead made the Circle constructor compute and store the area like this:
  public Circle(double radius) { this.radius = radius; area = Math.PI * radius * radius; },
  then you had getArea simply do this:
  public double getArea() { return(area); }
  Why will this strategy *fail* with what we know so far about OOP in Java?