Lab 9, CSC 203 - Exceptions

This lab explores *Exceptions* through the implementation of a program to handle exceptions when creating an object that represents a circle.

Objectives

- To be able to develop your test own Exception class.
- To be able to add an Exception mechanism to an existing class.
- To explore the differences between checked and unchecked Exceptions.
- To be able to write code that uses a try-catch-finally block.

Overview

You will be given code implementing a *Circle* class and will add functionality to create and throw your own exceptions given "bad" data in the Circle constructor. You will then write a CircleTest class to practice using a try-catch block.

Exploration

First we'll write some code to explore creating and using Exception classes. Follow the step below. You do not need to formally answer the questions, but make sure you know what the answers are!

1. Use the `Circle.java` code (given on PolyLearn) as a basis for the lab. Do not make any changes to this file other than the ones described below in Steps 6 and 8.

2. Complete the rest of these steps in order.

3. Write a class called `CircleException` that extends `Exception`.
   -Create one constructor that accepts a String message and passes that on to the parent constructor.

4. Write a class called `ZeroRadiusException` that extends `CircleException`.
   -Create a default constructor that calls the parent constructor with the message “zero radius”.

5. Write a class called `NegativeRadiusException` that extends `CircleException`.
   -Create a private double instance variable to store the radius.
   -Create a single constructor that accepts a radius as a double. Call the parent constructor with

the message “negative radius”. Update the instance variable with the value passed to the constructor.
-Create a public method radius() to query for the radius.

6. Modify the Circle class such that it examines the radius given in the non-default constructor. Throw a ZeroRadiusException if the given radius is 0. Compile the class… you should get an error.

7. Fix your error by having CircleException extend RuntimeException instead of Exception. Why did this fix the error? What else could you have done to fix the error?

8. Modify the Circle class such that it throws a NegativeRadiusException if a negative radius is given to the constructor.

9. Create a CircleTest class using the code shown below. Compile and run the code. The Circle should only get printed if the Circle constructor is successful. Why? Change the value given to your Circle constructor such that you try it with a valid radius, a zero radius, and a negative radius. When does the Circle get printed? When does the "In finally" get printed? When does the "Done" get printed?

```
10.
11.  public class CircleTest
12.  {
13.     public static void main(String[] args)
14.     {
15.        try
16.        {
17.           Circle c1 = new Circle(0);
18.           System.out.println(c1);
19.        }
20.        catch (CircleException e) {
21.           System.out.println(e.getMessage());
22.        }
23.        finally {
24.           System.out.println("In finally.");
25.        }
26.        System.out.println("Done.");
27.     }
28.  }
```

29. Then change your catch block so you specifically catch NegativeRadiusExceptions and ZeroRadiusExceptions. In the NegativeRadiusException code, additionally print out the radius. System.out.println(e.getMessage() + ": " + e.radius());

30. Experiment again with constructing various valid and invalid Circles.

31. Remove (or comment out) the NegativeRadiusException from your catch block and then construct the Circle with a negative radius. Where does the error message appear?

32. What happens if you construct a bad Circle *before* the try block?

33. Change the code back to exactly how it is shown in Step 9. Put a return statement in the catch block after the message is printed. Predict what will be printed when the code runs. Do the "In finally" or "Done" get printed?

Testing

Test your code with the basic JUnit tester, Lab09Tests.java. Your code must pass these tests to get *any* credit for the lab. Your code will be tested with tests that you have not seen, so you shoule add some more tests. Testing exceptions with JUnit can be a little tricky. You are given sample tests to show how to write a test when you expect an exception to be thrown and when you don't expect an exception to be thrown. You could also catch an exception in a JUnit test to assert things about the exception's message and other methods. Pay special attention to having exception messages exactly match those specified. Strings that don’t match will fail our additional tests.

Demonstration

Once you have completed the above steps, demonstrate completion of this lab to your instructor. Be prepared to answer any of the questions in the Exploration part of the lab. Make sure to hand in your code however your specific instructor expects (push to GitHub, hand in on PolyLearn, etc).