Institute of Technology, University of Washington Tacoma TCSS 305
Programming Practicum, Autumn 2017
Assignment 1 – Unit Testing
Value: 4% of the course grade
Due: Friday, 6 October 2017, 23:59:59

Program Description:

This assignment will give you practice writing unit tests. You are to write a set of unit tests for the provided Circle class.

- Your unit tests must be written as JUnit **4** tests. The unit testing lecture slides and the Point class tests developed during class will provide explanations and examples of how to write unit tests. The lecture slides include links to additional information online. An internet search for tutorials or examples of unit testing might help you find additional useful sources of information.
- Your unit tests must pass and provide complete code coverage of the Circle class. Use the EclEmma code coverage tool to check that your tests provide complete code coverage of the Circle class.
- Your unit tests should not produce any console output when run. All results of the tests should be reported by using JUnit assertions as demonstrated in class.
- A correct set of unit tests should expose bug(s) in the Circle class. This means that your unit tests must include at least one test that would fail on the provided code, but will pass after you identify and correct the 'bug(s)'. So, when you discover the bug(s), correct it but leave the test in place.
- Review the API documentation provided for the Circle class here:

http://repos.insttech.washington.edu/~cfb3/

Implementation Guidelines:

You must not change any code in the Circle class except to correct the bug mentioned previously.

Hints and Suggestions:

Each unit test should test a single method of the Circle class. Each unit test ideally should have only a single JUnit assertion or test a single exception. This does not mean that the Circle class and your test class will necessarily have the exact same number of methods. It may be possible to achieve complete code coverage of the Circle class without writing tests for every method of the Circle class (this will be true because you will use some methods of the Circle class to help test other methods). In some cases you may need to write several test methods in your test class to thoroughly test a single method of the Circle class. Name each of your test methods appropriately so that the name clearly indicates which Circle class method is being tested.

NOTE: The toString() method of the Circle class internally calls toString() in the Point and Color classes. A call toString() on the default black unit circle centered at the origin would return the following String:

```
Circle [radius=1.00, center=java.awt.Point[x=0,y=0], color=java.awt.Color[r=0,g=0,b=0]]
```

(This is correct output, not a bug!)

Stylistic Guidelines:

You will be graded on program style, which includes:

- use of descriptive method and variable names
- Javadoc comments on your test class and on every constant, field, and method in your test class

Notice that the provided Circle class produces no warnings from the plugin tools. Be careful to write your unit test class in a way which minimizes warnings from the tools. Ideally your unit tests should produce no warnings. If you have questions about what a particular warning means, and a quick online search doesn't help, ask about it as soon as possible.

Remember to include a non-javadoc header comment at the beginning of your test class with some basic information, in addition to full Javadoc comments. Examples of acceptable file header comments and of Javadoc comments appear in the Circle class and in the Point class code example from lecture.

Submission and Grading:

Create your Eclipse project by downloading the hw1-project.zip file from Canvas, importing it into your workspace (as described for hw0-project.zip in Assignment 0), and using “Refactor” to change “username” in the project name to your UWNetID. Remember to make this change before you first share the project to Subversion. Incorrectly named projects will be penalized when graded.

Also, as with Assignment 0, you can use whichever bracket style you like. The provided class uses the same line style which you may change to next line style if you wish.

You must check your Eclipse project into Subversion (following the instructions from Lecture 1 and Assignment 0), including all configuration files that were supplied with it (even if you have not changed them from the ones that were distributed). When you have checked in the revision of your code you wish to submit, make a note of its Subversion revision number. To get the revision number, perform an update on the top level of your project; the revision number will then be displayed next to the project name. Your revision number will pick up where you left off on Assignment 0; if you submitted revision 5 of Assignment 0, the first commit of Assignment 1 will have a number greater than 5. This is because you have a single Subversion repository for all your projects, and the revision number counts revisions in the entire repository.

After checking your project into Subversion, you must submit (on Canvas) an executive summary, containing the Subversion revision number of your submission, an “assignment overview” explaining what you understand to be the purpose and scope of the assignment, and a “technical impression” section describing your experiences while carrying out the assignment.

The filename for your executive summary must be “username-testing.txt”, where username is your UWNetID. As with the naming convention for your Eclipse project, your assignment will be penalized if it does not follow this naming convention. An executive summary template, which you must use, is available on Canvas. In particular, your executive summary must have a line containing exactly the text “Subversion Revision Number: #”, with no leading spaces, where “#” is the Subversion revision number you made a note of above (with no parentheses or other symbols). Executive summaries without a line following this exact format will be penalized. Executive summaries will only be accepted in plain text format – other file formats (RTF, Microsoft Word, Acrobat PDF, Apple Pages) are not acceptable. Using an unacceptable file format for the executive summary will be penalized.

Part of your program's score will come from its "external correctness." For this assignment, you will receive full credit for external correctness if your unit tests pass, achieve full code coverage, and produce no console output.

Another part of your program's score will come from its "internal correctness." Internal correctness for this assignment includes following coding conventions and providing documentation of your code (Javadoc comments). It also includes the quality of the tests you write. It is possible to achieve full code coverage with poorly designed tests. I will look at your tests to see if they do a reasonable job of testing the Circle class methods. I will look to see that you have identified and corrected the bug which exists in the Circle class.

Internal correctness also includes whether your source code follows the stylistic guidelines discussed in class. This includes criteria such as the presence of Javadoc comments on every method and field (even private ones!), the use of variable names, spacing, indentation, and bracket placement specified in the class coding standard, and the absence of certain common coding errors that can be detected by the tools. It is therefore to your advantage to be sure the plugin tools like your code before you submit it.

For this assignment, the percentage breakdown is 10% executive summary, 45% external correctness, 45% internal correctness.