CS4354 – Fall 2014 – Assignment 6

Due date: Friday, Dec. 5, 2014 at 5:00 pm.

Goal:

The goal of this assignment is to help students understand the use of JUnit to test Java code.

Description:

In this assignment you will create a set of unit tests, to test the behavior of parts of the code written for Assignment 4 & 5.

A unit test is an automated piece of code that invokes a unit of work in the system and then checks a single assumption about the behavior of that unit of work. A unit of work is a single logical functional use case in the system that can be invoked by some public interface (in most cases). A unit of work can span a single method, a whole class or multiple classes working together to achieve one single logical purpose that can be verified.

Think of unit testing as a way to test the behavior of the code (or parts of the code) written, without actually having to run the program. For example, in the case of assignments 4 & 5, assume that the front-end (user interface) part of the program, and the back-end part of the program, are written by two different developers. How would the developer of the back-end be able to ensure that the code he/she has written does what it is supposed to do without having access to the front-end?

A good unit test is:

- Able to be fully automated
- Has full control over all the pieces running (Use mocks or stubs to achieve this isolation when needed)
- Can be run in any order, if part of many other tests
- Runs in memory (no DB or File access, for example)
- Consistently returns the same result (You always run the same test, so no random numbers, for example.)
- Runs fast
- Tests a single logical concept in the system
- Readable
- Maintainable
- Trustworthy (when you see its result, you don't need to debug the code just to be sure)

In this assignment, you are asked to create JUnit tests to test the classes **CommunicationMngr** and **VehicleInventory** and all their **methods**. First you should consider testing the behavior of theses classes/methods under normal operation scenarios. For example, to test the method `newMessage` of the class

`CommunicationMngr`, you may need to create a test method, which creates a mock `Customer` object and some text and then calls the method `newMessage` to add the message to the `newMessages` queue. To ensure that everything worked as planned, you can then extract that message from the queue and check that the correct values have been assigned to the new message.

Subsequently, you can consider creating test cases for unusual scenarios, e.g. when a certain input or behavior is expected to cause an exception to be thrown.

Note that, in order to use the operations of the class CommunicationMngr, you may first have to create an object of that class. That can be done in a method annotated with `@BeforeClass` in the test class (CommunicationMngrTest). This method will be executed before any other test methods are executed.

At the end create a TestRunner class, which has a main method that runs the unit tests for the test classes that you have created, and prints out the test results.

Tasks:

1. Implement the JUnit tests as stated in the description section above. Try to be creative by coming up with test cases that can test as many different situations as possible.

2. Use a standard Java coding style to improve your program's visual appearance and make it more readable. I suggest the Google Java coding style: https://google-styleguide.googlecode.com/svn/trunk/javaguide.html

3. Use Javadoc to document your code.

Logistics:

This assignment will be done and submitted **individually** by each student.
Submit your answer in a single file (assign6_xxxxxx.zip). The xxxxxx is your TX State NetID.

Submit an electronic copy only, using the Assignments tool on the TRACS website for this class. Do NOT include executable or .class files in your submission.