Testing Methodology
Assignment 2
Unit testing using JUnit

COSC 565/475: Software Engineering I

Fall Semester 2018

Deadline: 27th November

What is JUnit?

JUnit is a framework for testing parts, or units of your code. In general, these units are considered to be the methods of each class. JUnit can help you to make sure that each of your classes work as expected. In unit testing you will usually write one test class for each of the classes that you want to test. Your test class will often include a test method for each method implemented by the class being tested. But keep in mind that this is not always feasible, or necessary. A test case will, and should often touch on more than a single method. These tests should be written to make it likely that when all the tests pass, the code is functioning as required.

JUnit with Eclipse

- Run Eclipse IDE. We will create a new workplace project
    - so click File -> New -> Project,
    - then choose Java and click Next.
    - Type in a project name -- for example, ProjectWithJUnit. Click Finish.
    - The new project will be generated in your IDE. Let's configure our Eclipse IDE, so it will add the JUnit library to the build path.
    - Click on Project -> Properties, select Java Build Path, Libraries, click Add External JARs and browse to directory where your JUnit is stored. Pick *junit.jar* and click Open.
    - You will see that JUnit will appear on your screen in the list of libraries. By clicking Okay you will force Eclipse to rebuild all build paths.
    - To create such a test, right-click on the ProjectWithJUnit title,
    - select **New -> Other**, expand the "Java" selection, and choose **JUnit**.
    - On the right column of the dialog, choose **Test Case**, then click **Next**.

Example

```
public class HelloWorld {
        public String say() {       return("Hello World!");    }
}
```

```
import junit.framework.TestCase;
public class TestThatWeGetHelloWorldPrompt extends TestCase {
   public TestThatWeGetHelloWorldPrompt(
       String name) {
       super(name);
   }
   public void testSay() {
       HelloWorld hi = new HelloWorld();
       assertEquals("Hello World!", hi.say());
   }
   public static void main(String[] args) {
       junit.textui.TestRunner.run(
           TestThatWeGetHelloWorldPrompt.class);
   }
}
```

Basic Information

Before you begin, we would like to call your attention to the following conventions:

- A Test Case Class is named [classname]Test.java, where classname is the name of the class that is tested.
- A Test Case Method is a method of the Test Case Class which is used to test one or more of the methods of the target class. Test Case Methods are annotated with @Test to indicate them to JUnit. Cases without @Test will not be noticed by JUnit.

JUnit assertions are used to assert that a condition must be true at some point in the test method. JUnit has many types of assertions. The following is a selection of the most commonly used assertions:

_ **assertEquals**(expected, actual): Assert that expected value is equal to the actual value. The expected and actual value can be of any type, for example integer, double, byte, string, char or any Java object. If the expected and actual values are of type double or oat, you should add a third parameter indicating the delta. It represents the maximum difference between expected and actual value for which both numbers are still considered equal.
_ **assertTrue**(condition): Asserts that the Boolean condition is True.
_ **assertFalse**(condition): Asserts that the Boolean condition is False.
_ **assertNull**(object): Asserts that an object is null.
_ **assertNotNull**(object): Asserts that an object is not null.
_ **assertSame**(expected object, actual object): Asserts that two variables refer to the same object.
_ **assertNotSame**(expected object, actual object): Asserts that two variables do not refer to the same object.

Whenever an assertion fails, an AssertionError is thrown, which is caught by the JUnit framework and presented as a red bar, indicating test failure. Assert statements accept an extra message parameter before the other parameters.

Loading the Project

Download the file Assignment2.zip from blackboard

In Eclipse, choose File -> New -> Java Project. Give it a name ("Lab1", for instance) and click Finish.

Running the Test Cases

When you run a test class, JUnit will run each method annotated with @Test separately and show a green bar if all of them pass, and a red bar if any of them fail. It is important that anything happening in a test method is independent from the other test methods, otherwise you risk getting weird results.

Assignment Tasks

- You have been given a template for the *BattleshipGame class and BattleshipException class*. You have to write minimum **25 test cases** for the 2 classes.
- Create BattleshipGameTest.java
- If you run your test cases for BattleshipGameTest.java at this point, they should be all pass.
- Write the following test cases
    - Check that the board is of the right size if a positive board is requested.
    - brief Check that a BattleshipException is thrown if the number of columns is < 0.
    - brief Check that a BattleshipException is thrown if the number of rows is < 0
    - brief Check that a BattleshipException is thrown if a 0x0 board is requested.
    - brief Check that ship placement works, including 0.
    - brief Check that placeShip throws IndexOutOfBoundsException when column or row < 0.
    - brief Check that placeShip throws IndexOutOfBoundsException when column > numCols-1 or row > numRows-1.
    - brief Check that placeShip throws BattleshipException if ship is placed diagonally
    - Check that placeShip throws BattleshipException if ships overlap.
    - Check that a BattleshipException is thrown if the coordinate pairs are specified in the wrong order (i.e., startCol > endCol or startRow > endRow).
    - brief fireShot() should return true if a battleship is hit.
    - brief fireShot() should return false if a battleship is not hit.
    - brief fireShot() should return true both times if a battleship is hit twice in the same spot.
    - brief fireShot() should return false if coordinates are negative.
    - brief fireShot() should return false if coordinates are generally out of bounds.
- Archive your project folder in zip format and name it FirstNameLastName.zip where the first name and last name refers to your name.

- Comment your code and comment all the test cases
- Uses **5 different assert statements**. The most common assert statement is assertEquals() which takes at least two arguments. Each JUnit test case (method) should have at least one of the assert statements listed below, otherwise the test passes, which can be misleading if you never actually check the value of any data.
- Examine the API for JUnit and add a few additional methods and test cases that use other Assert class methods, such as assertArrayEquals, assertTrue, assertFalse, assertNull and fail.

FOR BONUS ONLY: Create public boolean isGameOver() function that returns true if game is over, false otherwise

i.e @return true if game is over, false otherwise

IMPORTANT
Working independently. Your answers to the questions on this assignment will be individually marked, and must be your own work. You will be assigned 0 marks for this entire assignment, if any of your answers to individual questions bears a close resemblance to another student's submission, or to something previously published on the internet or elsewhere.

Reference
https://www.ibm.com/developerworks/library/j-introducing-junit5-part1-jupiter-api/index.html