Battleship Testing Assignment

Advanced Object-Oriented Design

Gerb

# Objective:

The student will create a test plan and a test script for software that was not written by the student and use them to test software.

# Overview

For this project you will be testing several versions of the Ocean class (from the Battleship 4 assignment). At some point in the future, you will be given a library containing multiple versions of the Ocean class. Some of these work, others don’t. Your job will be to figure out which ones work and which do not.

You will do this in three steps:

1. (*In groups of 3-4*) Write a test plan, similar to the one you created for the Boat class.
2. (*Also in your groups*) Write a test script that tests a version of the Ocean class and provides output that makes it easy to tell whether it functioned properly.
3. (*Alone, in a single class period*) Run your test script on several versions of the Ocean class and report which work and which do not.

# Step 1: Test Plan

For this part of the project, you will form groups of 3-4 students. Your group will generate a thorough test plan. Because this test plan will serve as the basis for your test scripts in the next step, your test plan will need to be thorough. The teacher will not be grading each group’s test plan.

# Step 2: Test Script

A *test script* is a set of steps used to execute a test plan on a piece of software. Your test script will be *automated*, meaning that it will be run by executing a Java program that takes no input from the user. This is probably best done by creating a method that executes your test script:

public static void RunOceanTestScript()

Your method will test a subclass of Ocean using all the test cases from your test plan. Because you will have a limited time actually to perform the testing, it will help you to make the output very easy to read. A two column format is often used:

Test Expected Actual

---- -------- ------

Call hit on E-3 True True

Call sunk on E-3 True False

With a two column format it is easy to see whether the test worked the way it should have.

You will be given a .jar file containing a version of the Ocean class that should work properly. The class will be called GoodOcean. You can test your test script (called *validating* the script) by including the .jar file in your project and running your script to test it. All tests should work properly.

To include the .jar file in your project, you can follow the steps listed in the **How-to-Load-a-Jar-File-Into-Eclipse** file.

When you are validating your script, you may find it helpful to create subclasses of ocean where one or more of the methods are specialized not to work, so you can make sure the test script correctly identifies them as faulty.

**Tip**: Test boundaries!

# Step 3: Test!

For this part of the project you will be working alone. You will be given a .jar file containing several subclasses of the Ocean class. Some of them will work, others will not. You will use the test script your group created, but you will not be working with the other members of the group. Using your test script, you will determine which of the versions of the Ocean class work and which do not. You will be assessed on how accurately you are able to determine which work and which do not.

# Evaluation

You will be evaluated based on how many of the classes to be tested you correctly evaluated, and whether you were on task during the group project work.