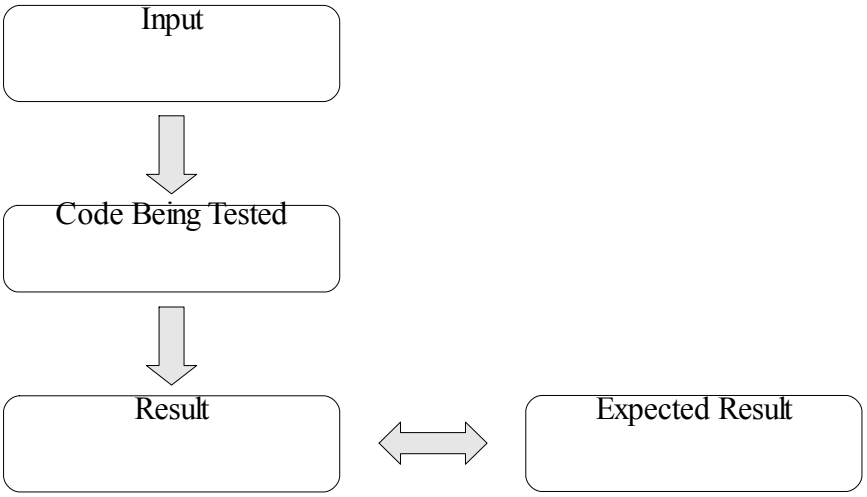
Lab: Unit Tests

CS 162
Summer 2012

Testing Overview
The testing process:



For unit tests, the code being tested is a method. The input is one or more parameters to that method. The result is either a return value from the method, or a side effect to the object that can be observed by calling another method on that same object. (For example, calling a "setter" can be tested by calling the associated "getter" method.)

In unit tests, the expected result is often a literal value: a hard-coded number or string. Basically, you work out a result by hand and hard-code the correct answer as the expected result. To compare actual results and expected results, use the "assert" methods provided by JUnit: assertEquals, assertTrue, assertFalse, etc. Note: the expect result should be provided as the first parameter and the actual result should be the second. If you switch the order, when tests fail, the JUnit messages will be confusing.

Often, a unit test needs to perform some preparatory work before being able to send the input to the code being test. For example, the unit test may need to make an instance of the class being tested before calling the method on that instance. Or, the unit test may need to call other methods to change the state of the object before calling the method in question. However, the fewer steps in a unit test, the easier it is to avoid a mistake in the test itself.

Getting Ready
You will need `Pirate.java` and `TreasureChest.java`, these should be available from the assignment website or the course management system (blackboard). You lab work should be individual.

Exercise 1: The Pirate Class
You will be working individually for this exercise. Add the `Pirate.java` file to a new project. Set up the package directory path accordingly. Write the following unit tests:

1. A test for the constructor. This test will make sure that the constructor sets the name and the initial gold amount correctly. This will require testing the `getName()` and `getNumGold()` methods at the same time.
2. A test for the constructor in which passing a negative amount for the initialGold value throws an Exception.

For the next step, you will write a new method called `removeGold()` which will cause the pirate to remove a certain amount of gold from their personal stash of gold. However, you need to write the test *first,* before you create a working method. Your goal is to design the test is such a way that it will pass when the method is working. A good test should indicate when the actual functionality agrees with expected functionality.

Include this file when you submit your files to TEACH after finishing the rest of the lab.

Exercise 2: The TreasureChest Class
Add the `TreasureChest.java` file to your project at this time. In this part, you will write (not in code but in English), all the tests you would need to write for the following functions from TreasureChest.java. You will need to read the Javadoc comments for each of the listed functions.

Describe each test in one English sentence (no code is required). Remember that when testing with JUnit, you can only use public functions in your test code, so your English description can only check values that can be obtained through public functions. One description is provided for you (although the constructor needs more tests).

1. The TreasureChest constructor

   Provide an initial amount of gold > 0 (positive input) and verify the amount of gold in tc

2. The addGold method

3. The removeGold method

4. The checkGold method

Now implement one of the four test groups above. When your methods are fully implemented, your unit tests should pass. Include a description of your work at the top of the JUnit test file and submit it to TEACH.