January 25th: In class exercise
(nothing to hand in)
You are encouraged to work together in groups in class. Part of this exercise involves discussion and exploration with others.

Background:
In 1979, Glenford Meyers published a book called the "Art of Software Testing". In that book he presented a short self-test for practitioners, and this has become a classical software testing problem.

"*Create a set of test data for the program—data the program must handle correctly to be considered a successful program. The program reads three integer values from an input dialog. The three values represent the lengths of the sides of a triangle. The program displays a message that states whether the triangle is scalene, isosceles, equilateral or invalid. Remember that a scalene triangle is one where no two sides are equal, whereas an isosceles triangle has two equal sides, and an equilateral triangle has three sides of equal length.*"

It turns out that many professional programmers were unable to come up with a complete set of test cases for this relatively simple program. Today we will use the triangle problem to "warm up" for this class and learn a bit about some potential challenges for software testers.

Group discussion:
Have a short discussion in your group about what types of tests you should run for this program? What makes a triangle valid or invalid? Think about geometry, such as the sum of any two sides of a triangle must be greater than the third. Although this is a simple problem there are a lot of things to think about. Of course, these tests are not aware of the code of the program (which can vary greatly depending on the implementation). In this exercise we will use a code-coverage tool to help us visualize how much of the code has been "tested". This tool is called JacCoco (https:// https://www.jacoco.org/jacoco/trunk/index.html/). We will use a version I have packaged for you with a triangle program. This program works from the command line using maven. You are welcome to get the tool from git and set it up yourself. The version I have provided works on most Unix type machines but needs maven and Junit 4.2.

Exercise Environment:
We will be using the COM S server: pyrite.cs.iastate.edu for this exercise where this has been tested. Today's exercise has also been tested on a Mac laptop. You are welcome to try other environments if you can get it to run there. You will need Java, a text editor. You may also want to have an html viewer.

If you have not logged into pyrite before you need to have an "ssh" and "sftp" client. You can download "putty" (https://www.putty.org/) and WinSCP (https://winscp.net/eng/index) for this purpose

Download:
In Canvas under exercises, you can download "jacoco-0.8.11.for417.tar.gz". Using sftp (you can use putty or a native shell if you have a Linux or Mac machine) transfer the file to your pyrite account.

Open your sftp client and go to the directory where you have the downloaded file

- sftp yournetid@pyrite.cs.iastate.edu

change to the desired directory on pyrite (I suggest you make a COMS_417 directory). The command below assumes you have this directory

- cd COMS_417

Transfer the file (assumes you start in the directory with jacoco-0.8.11.for417.tar.gz. If not you need the path)

- put jacoco-0.8.11.for417.tar.gz

Now exit and ssh into pyrite

- ssh yournetid@pyrite.cs.iastate.edu
- cd COMS_417

Unpack the file:

- gunzip jacoco-0.8.11.for417.tar.gz
- tar -xvf jacoco-0.8.11.for417.tar

This will create a directory structure for you with the files

Jacoco-0.8.11

Inside of that directory move to the "examples" directory and to the "triangle" example under that:

- cd jacoco-0.8.11
- cd examples/triangle

In the " src" directory there are two directories (main and tests). The program we care about testing is the TriangleType.java program. There is also an enumerated type class Triangle.java.

For this exercise you don't need to be well versed in Junit. We will cover that later.

Go to the source directory and look at the TriangleType.java program. You will see there is a fault in the program (which I put in there but is commented out). You can also look at the Junit tests inside of the 'test' directory to see if you think they are complete.

Go back to the triangle directory (jacoco-0.8.8/examples/triangle) and use the mvn commands below to generate the classes and run unit tests on the triangle program:

- mvn clean

- mvn test
- mvn jacoco:restore-instrumented-classes
- mvn jacoco:report

If successful, this will result in several new directories. The 'target' directory has a 'sites' directory with the coverage reports and a 'surefire' directory with the Junit test results. Find the relevant reports (see lecture notes from Tuesday).

Now tar (compress) the directory:

- tar -cf results.tar target/

This creates a file called results.tar

Use sftp and the get command to bring it to your local machine.

Navigate to reports directories and find the Junit report and the coverage fields.
.

Questions to answer in your group:

1. What is the line/branch coverage?
2. Did you find the fault?
3. Can you think of a test case that should be added?
4. Can you get to 100 percent coverage?

Fix the fault by commenting out the incorrect code (line 36) and adding in the correct version (line 32)

Now add a new fault as shown below on Line 18:

```
// Reject non-positive sides
   //correct code - comment out
    //     if (s1 <= 0 || s2 <= 0 || s3 <= 0)

// add new fault    - last condition wrong
    if (s1 <= 0 || s2 <= 0 || s3 < 0)
```

Now try to add a test case to find this fault. Can you find it? Discuss with respect to RIPR.