JUnit Testing

- Objectives - when we have completed this set of notes, you should be familiar with:
  - How to test your program in interactions
  - Concepts of Unit testing
  - How to write JUnit tests in jGRASP
  - The assertEquals and assertArrayEquals methods

Testing: The Basics

- Remember the following terminology:
  - **Failure**: An undesired (incorrect) result produced by the software.
  - **Fault (or Defect)**: the underlying cause of the failure (a “bug” or “error” in your code).
- The purpose of testing is to identify failures so that the underlying faults (or defects) can be removed.
- Debugging is the process of removing a fault. (Note that debugging occurs after a failure has revealed the existence of a fault.)

Testing: The Basics

- **Unit Testing**: testing one unit or component at a time. (e.g., testing a class and its methods)
- **Integration Testing**: testing the interfaces among components (classes/methods) in a software system with multiple components.
- **System Testing**: testing the entire software system to make sure it meets the customer's requirements and expectations. (i.e. checking the driver program's output).
- Our focus will be on **Unit Testing**.

Testing: The Basics

- Consider Triangle2. (see Triangle3 for solution)
- To perform unit tests on the getClassification() method, you could execute something like the following code in interactions (or similar code in a driver program):

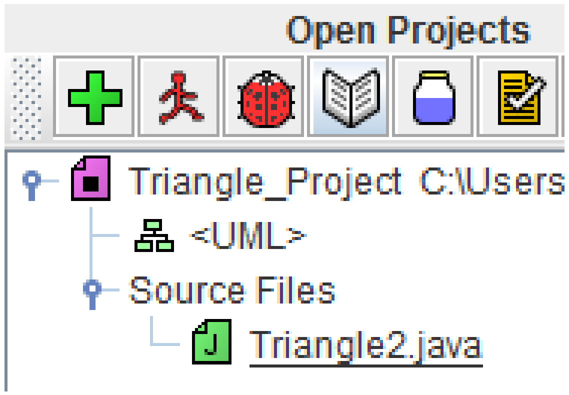
```
Triangle2 t1 = new Triangle2(5, 5, 5);
t1.getClassification()
equilateral

Triangle2 t2 = new Triangle2(5, 7, 5);
t2.getClassification()
scalene

 . . .
```

Testing: The Basics

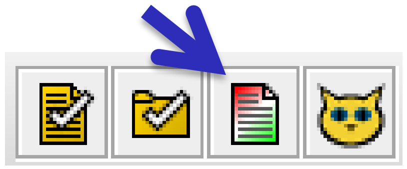
- If you've been testing your classes in interactions, you may have noticed some drawbacks:
  - It can become tedious. Change code -> end interactions, recompile -> re-do the interactions.
  - Changes to one method necessitate re-testing other methods as well -> re-doing even more interactions.
- What if there was a way to write a few simple statements, save them as a test, and then be able to rerun all the saved tests with one click?
- There is! **The JUnit framework**.

JUnit

- Make sure that all of your program's files are in a jGRASP project.

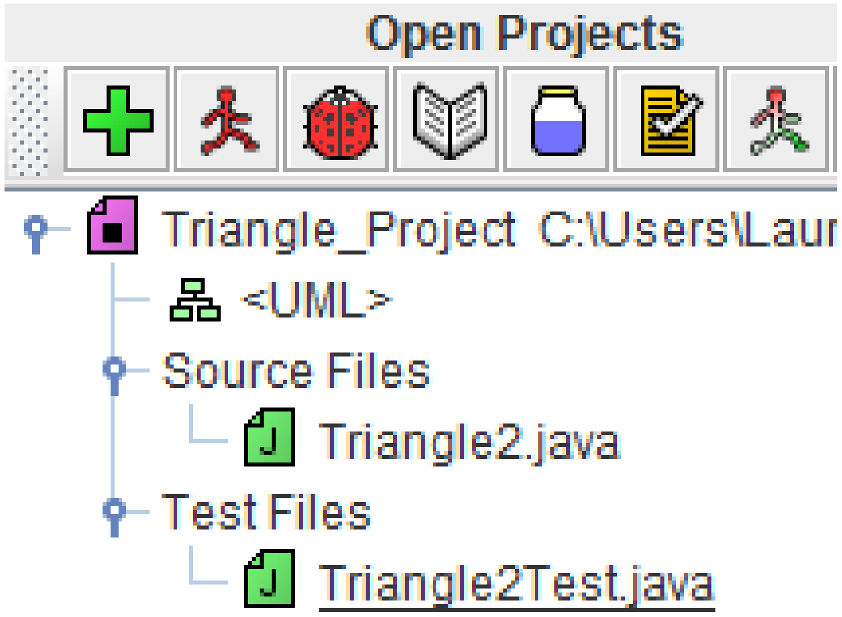


- To set up a test file, open the class that you want to test, then click the Create Test File button:



JUnit

- You’ll now see a Triangle2Test file in the project:



JUnit

- In the test file, delete the `@Before` method and the `org.junit.Before` import (we will not cover `@Before`, but you can use it if you wish).
- Also delete the contents of the defaultTest method for now.

```
public class Triangle2Test {

      /** A test that always fails. **/
       @Test public void defaultTest() {

      }
}
```

JUnit

- Suppose that we want to make sure that an equilateral triangle is correctly classified. First, change the Javadoc and method header to describe the test:

```
/** Tests an equilateral classification. **/
      @Test public void equilateralTest() {
```

- Note that the **@Test** tag makes the method a test case; `public void` is required; you get to choose the method name

JUnit

- Now add code in the method to set up an equilateral triangle (just like you would in interactions:

```
/** Tests an equilateral classification. **/
 @Test public void equilateralTest() {
    Triangle2 t = new Triangle2(5, 5, 5);

  }
```

AssertEquals

- To test the method, you can in invoke the AssertEquals method. This method will report a <u>failure</u> if the expected value (i.e., the correct value) does not match the actual value (e.g., your method’s return value).

- When comparing integer values or objects, you can use one of following forms of assertEquals:

```
Assert.assertEquals(expected, actual);

Assert.assertEquals(error msg, expected, actual);
```

AssertEquals

- In our example, we are testing the getClassification method to make sure that its return value is equilateral for our 5, 5, 5 triangle.
  - Expected value: `"equilateral"`
  - Actual value: t.getClassification()
- Add the following code to your method:

```
Assert.assertEquals("equilateral",
                     t.getClassification());
```

AssertEquals

- Compile and run your test. If the output is OK, then your test passed.

- The method was correct for a triangle with sides: 5, 5, 5

```
JUnit version 4.9b2
 .
 Time: 0.004

 OK (1 test)
```

JUnit

- Add a method to test the isosceles output:

```
/** Tests isosceles classification. **/
    @Test public void isoscelesTest() {
      Triangle2 t = new Triangle2(5, 7, 5);
      Assert.assertEquals("isosceles",
         t.getClassification());
   }
```

JUnit

- Also add a method to test the scalene classification AFTER setSides is invoked (to check for errors in setSides):

```
@Test public void scaleneAfterSetTest() {
   Triangle2 t = new Triangle2(5, 7, 5);
   t.setSides(3, 4, 5);
   Assert.assertEquals("scalene",t.getClassification());
}
```

JUnit

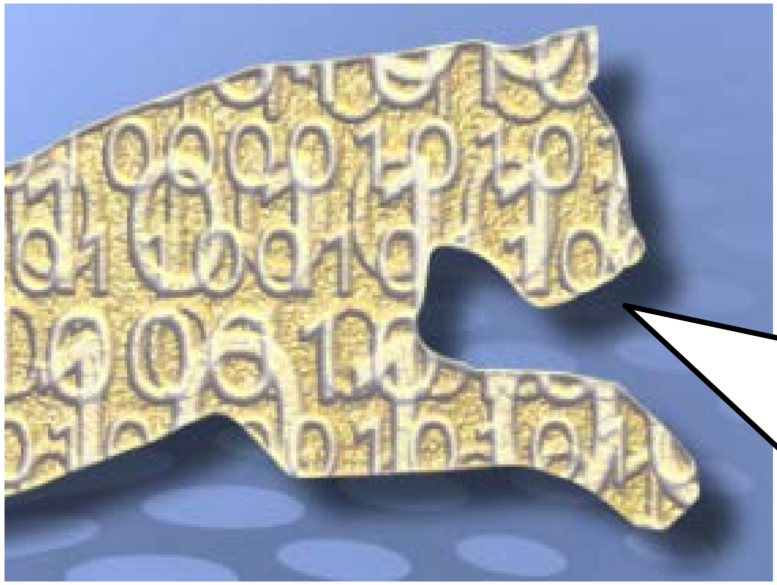
- When you run the method, it fails!

```
org.junit.ComparisonFailure:
  expected:<[isosceles]> but was:<[scalene]>
```

- Looking closely, you discover that there is a logic error in the source code on line 32.
- After you make the change, the `scaleneAfterSetTest` method fails due to a logic error in the setSides method.

JUnit

- Take a look at the scaleneTest method; it includes an error message in the output if the scalene method is incorrect.
- This type of output should be familiar…



Someone writes JUnit tests so that I can grade your projects. Otherwise, you wouldn't have the opportunity to raise your grade with multiple submissions!

Other Assert Methods

- As previously stated, if you wish to compare Strings or integers, then use the following:

```
Assert.assertEquals(expected, actual);
```

- To test floats or doubles:

```
Assert.assertEquals(expected, actual,
  delta);
```

  - Delta is the number of decimal points that you want to compare; for example, 0.0001 compares two doubles to 4 decimal places

Other Assert Methods

- To test arrays:

```
Assert.assertArrayEquals(expected, actual);
```

- You may also have to get creative when testing methods like toString. Suppose we only wanted to make sure that toString contains the word scalene:

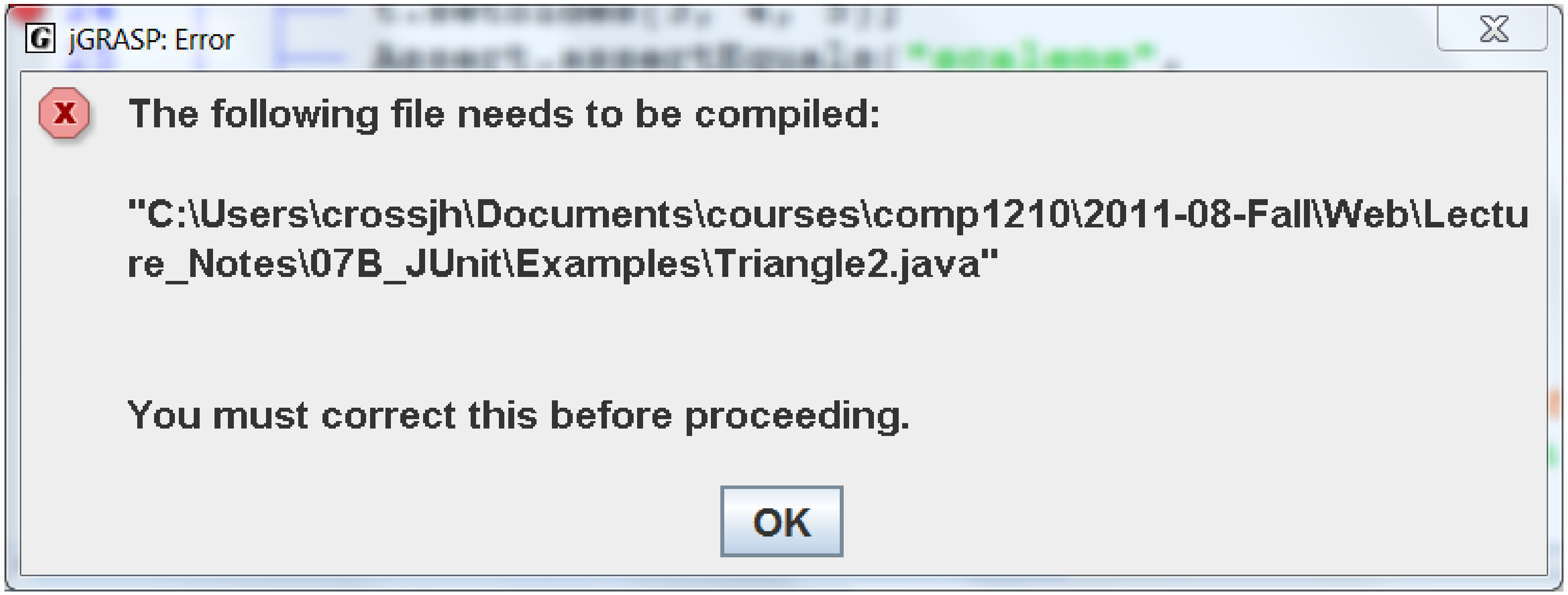
```
boolean hasExp = t.toString().contains("scalene");

Assert.assertTrue(hasExp);
```

- For details on all assert methods see:

  http://www.junit.org/apidocs/org/junit/Assert.html

Errors

- If you get this error message then you need to recompile the project before running the test:



The following file needs to be compiled:

"C:\Users\crossjh\Documents\courses\comp1210\2011-08-Fall\Web\Lecture_Notes\07B_JUnit\Examples\Triangle2.java"

You must correct this before proceeding.

Errors

- If you get compiler errors like the one below,

```
Triangle2Test.java:1: package org.junit does not exist
```

then you may need to:

- Make sure the project is open.
- Make sure the test file is in the project.
- If the test file is in the Source Files category of the Project, Right-click the test file and choose “Mark as Test” to move it into the Test Files category