

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 assertEquals methods

The jGRASP tutorial [Using JUnit with jGRASP](#) provides additional details and examples



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 you could have 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
```



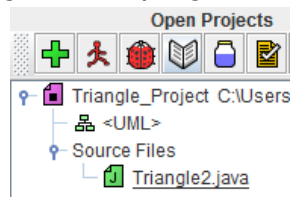
Testing: The Basics

- If you've been testing your classes in interactions, you may have noticed some drawbacks:
 - It can become tedious.
Change code -> recompile -> re-do the interactions.
 - Changes to one method necessitate re-testing other methods as well, thus 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.**

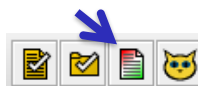


JUnit

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

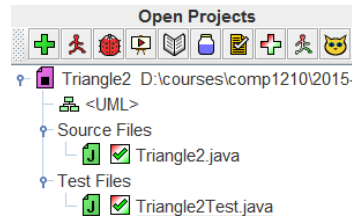
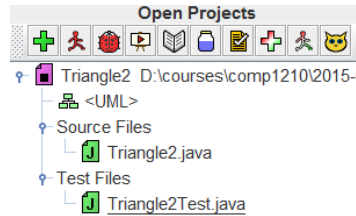


- To set up a test file, open the class that you want to test, then click the Create Test File button:



JUnit

- You'll now see a Triangle2Test file in the project:
- Before running JUnit After running JUnit

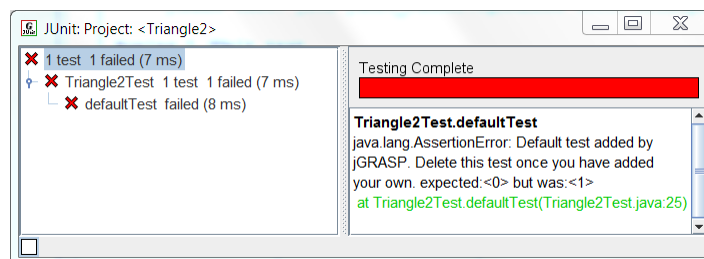


- Green check marks indicate all test methods passed. Red **X** indicates at least one test failed



JUnit

- jGRASP Test Results Window showing a failed test method: defaultTest



- Clicking on the green link in right pane takes you to the point of failure in the test method



JUnit

- In the test file, you may ignore the `@Before` method and the `org.junit.Before` import (we will not cover `@Before`, but you can use it if you wish); or comment/delete these statements; also comment out: `import static org.junit.Assert.*`;
- Comment out the default `Test` method, and use it as a reference for making your own test methods; or simply delete it

```
/** A test that always fails. */  
@Test public void defaultTest() {  
    Assert.assertEquals("Default test added by jGRASP. Delete "  
        + "this test once you have added your own.", 0, 1);  
}
```



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 invoke the assertEquals method. This method will report a failure 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.

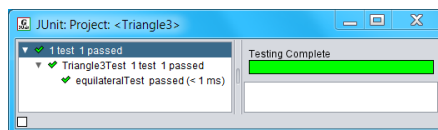


- The method was correct for a triangle with sides: 5, 5, 5

Running 1 JUnit test.

Completed 1 tests 1 passed

- JUnit Results Window



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):

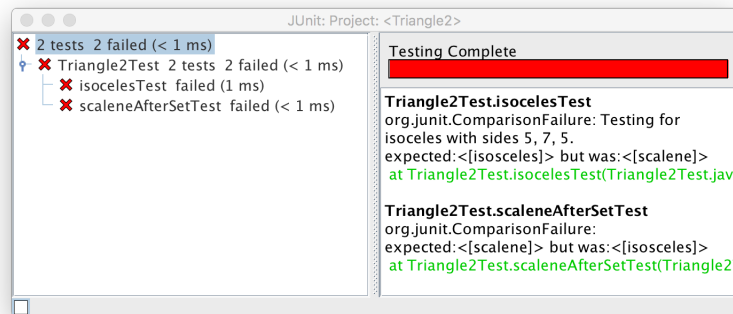
```
@Test public void scaleneAfterSetTest() {  
    Triangle2 t = new Triangle2(5, 7, 5);  
    t.setSides(3, 4, 5);  
    Assert.assertEquals("scalene", t.getClassification());  
}
```



JUnit

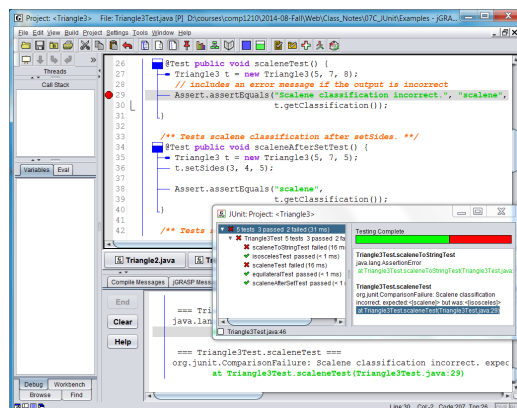
- When you run the method, it fails!

org.junit.ComparisonFailure: Scalene classification incorrect.
expected:<[scalene]> but was:<[isosceles]>



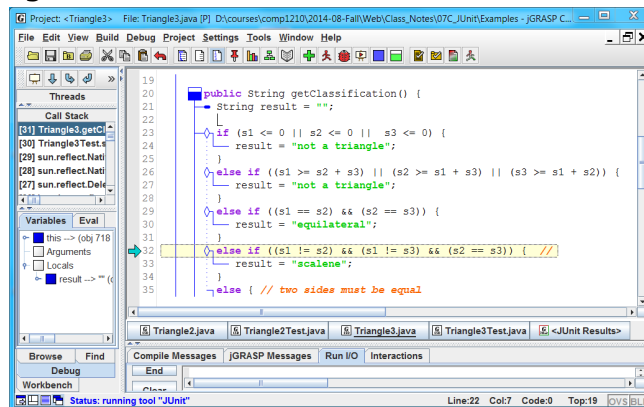
JUnit

- Set a breakpoint (Brkpt) on the statement that calls the method that failed
- Run Debug on the test file
- When program stops at Brkpt, "step-in" to method
- Look for the error as you step



JUnit

- Looking closely, you discover that there is a logic error in the source code on line 32.

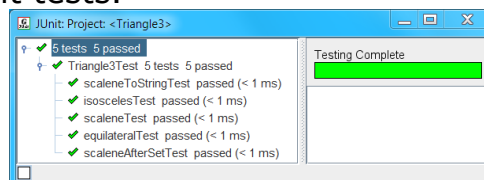


```
19 public String getClassification() {
20     String result = "";
21     if (s1 <= 0 || s2 <= 0 || s3 <= 0) {
22         result = "not a triangle";
23     }
24     else if ((s1 >= s2 + s3) || (s2 >= s1 + s3) || (s3 >= s1 + s2)) {
25         result = "not a triangle";
26     }
27     else if ((s1 == s2) && (s2 == s3)) {
28         result = "equilateral";
29     }
30     else if ((s1 != s2) && (s1 != s3) && (s2 == s3)) { //
31         result = "scalene";
32     }
33     else { // two sides must be equal
34     }
35 }
```



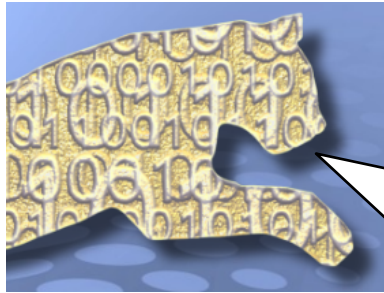
JUnit

- After you make the change, the `scaleneAfterSetTest` method fails due to a logic error in the `setSides` method.
- See if you can find and correct the error using the technique above.
- After correcting the errors uncovered by the test methods, the program should pass all of the JUnit tests.



JUnit

- Take a look at the `scaleneTest` method; the `Assert stmt` has an error message to be included in the output if the `scaleneTest` method fails.
- 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: (double arrays, 3rd parameter is delta)

```
Assert.assertEquals(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:

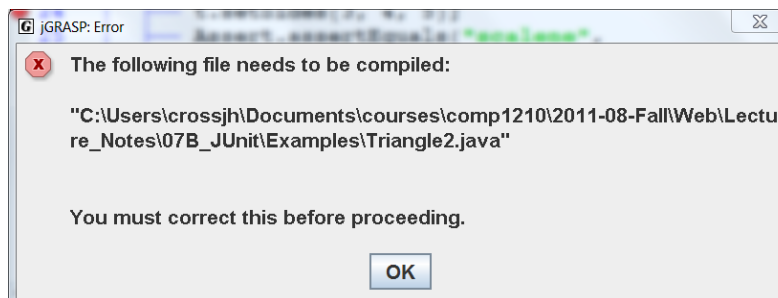
```
boolean hasExp = t.toString().contains("scalene");  
Assert.assertTrue(hasExp);  
Assert.assertFalse(!hasExp);
```

- For details on all assert methods see:
<http://junit.sourceforge.net/javadoc/org/junit/Assert.html>



Errors

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



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; alternatively, a file can be dragged from Source Files to Test Files or from Test Files to Source Files

