JUnit

- Objectives - when we have completed this set of notes, you should be familiar with:
  - Basic terminology of testing
  - Categories of testing
  - Concepts of Unit testing
  - JUnit testing with jGRASP
  - The assertEquals and assertArrayEquals methods
  - The assertTrue and assertFalse methods

The jGRASP tutorial Using JUnit with jGRASP provides additional details and examples
(http://www.jgrasp.org)

Testing – The Basics

- Testing 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 finding and removing a fault (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
- Our focus will be on **Unit Testing**

Testing Using Interactions

- Consider Triangle2 (see Examples folder)
- To perform unit tests on the getClassification() method, you might execute the following code in jGRASP 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()
   isosceles
```

Testing Using Interactions

- When using interactions to test your classes, you may have noticed some drawbacks:
  - It can become tedious . . .
    Change code -> recompile -> re-do the interactions
  - Changes to one method may 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 run/rerun all the saved tests with one click?
- There is! **The JUnit framework**

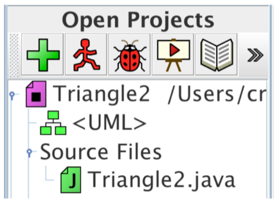
Testing Using JUnit

Basic Steps:

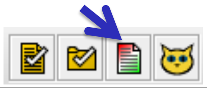
- Create a jGRASP project and add your source files (and JUnit test files if they exist)
- Open source file to be tested; create JUnit test file for class to be tested (run test file; comment out or replace default test method that fails, if any)
- Add test methods to test class; run tests as test methods are added (or run all JUnit test files from project menu)
- For a failed test, run debug on test file with breakpoint on call to method that failed; step-in to method that failed; then step through method looking for error; fix and run tests again.

jGRASP Project and Test File

- Make sure the source files for your program are in a jGRASP project (See Triangle2 project)

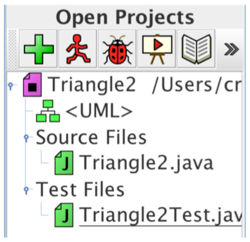


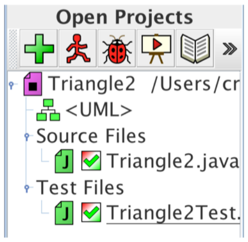
- To create a new test file, open the source file that you want to test, then click the Create Test File button on the top toolbar:



Project Test Results

- See test file (Triangle2Test.java) in the project
- Before running JUnit After running JUnit

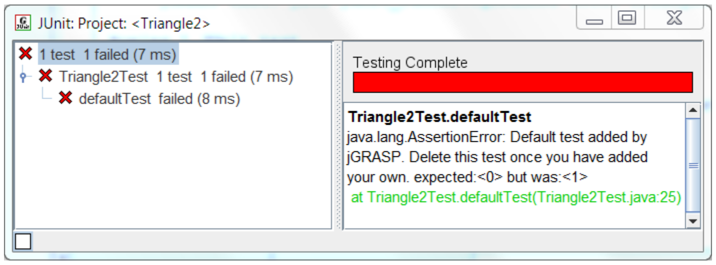




- Green check marks indicate all test methods passed. Red **X** indicates at least one test failed

JUnit Results Window

- JUnit Window showing results of a failed test method: defaultTest


JUnit: Project: <Triangle2>
1 test 1 failed (7 ms)
Triangle2Test 1 test 1 failed (7 ms)
defaultTest failed (8 ms)
Testing Complete

```
Triangle2Test.defaultTest
java.lang.AssertionError: Default test added by
jGRASP. Delete this test once you have added
your own. expected:<0> but was:<1>
 at Triangle2Test.defaultTest(Triangle2Test.java:25)
```

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

The Test Class File

- In the test file, comment out or delete:
  `import static org.junit.Assert.*;`
  You may ignore the `import org.junit.Before;` and the `@Before` method (we will not cover `@Before`); or comment/delete these statements

- Comment out the defaultTest 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);
}
```

Adding First Test Method

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

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

}
```

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

Adding Object to Test

- Now add code in the test method to set up an object for the test; in this case, we create an equilateral triangle (just like you would in interactions):

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

}
```

Adding assertEquals

- To test the method, you can in invoke the assertEquals method; this method will report pass if the expected value (i.e., the correct value) is equal to the actual value (e.g., your source method's return value); otherwise, the assertEquals method will report a failure
- When comparing primitive values (except float and double) and most objects for equality, you can use one of following forms of assertEquals:

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

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

Adding assertEquals

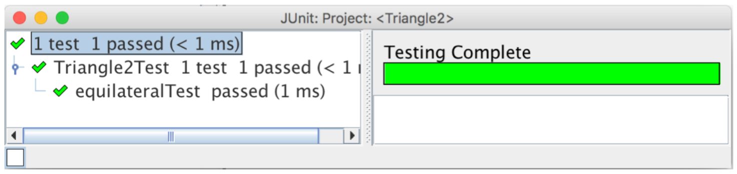
- In our example, we are testing the getClassification method to make sure its return value is equilateral for our (5, 5, 5) triangle.
  - Expected value: `"equilateral"`
  - Actual value: t.getClassification()
- Add the assert to complete your test method:

```
/** Tests equilateral classification. **/
@Test public void equilateralTest() {
   Triangle2 t = new Triangle2(5, 5, 5);
   Assert.assertEquals("For sides 5, 5, 5: ",
         "equilateral", t.getClassification());
}
```

Compile/Run Test File

- Compile and run your test using buttons on edit window toolbar
- Results are shown in Run I/O tab and JUnit Window that opens; the method was correct for a triangle with sides: 5, 5, 5

```
Runing 1 JUnit test.
Completed 1 tests  1 passed
```


JUnit: Project: <Triangle2>

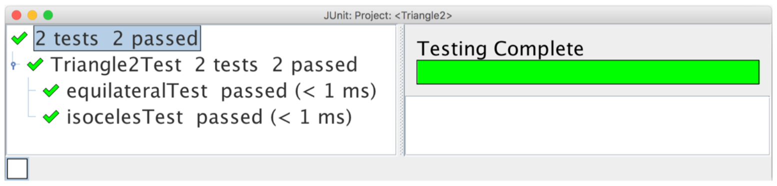
```
1 test  1 passed (< 1 ms)
  Triangle2Test  1 test  1 passed (< 1
    equilateralTest  passed (1 ms)
```

Testing Complete

Test Method for Isosceles

- Add a method to test the isosceles classification and run the test file:

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


JUnit: Project: <Triangle2>

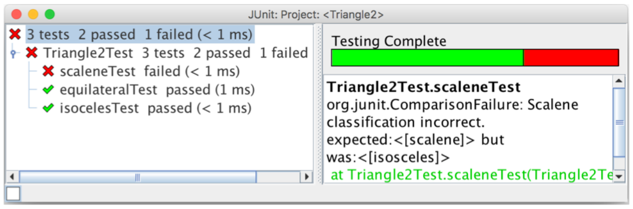
```
2 tests  2 passed
  Triangle2Test  2 tests  2 passed
    equilateralTest  passed (< 1 ms)
    isocelesTest  passed (< 1 ms)
```

Testing Complete

Test Method for Scalene

- Add a method to test the scalene classification and run the test file:

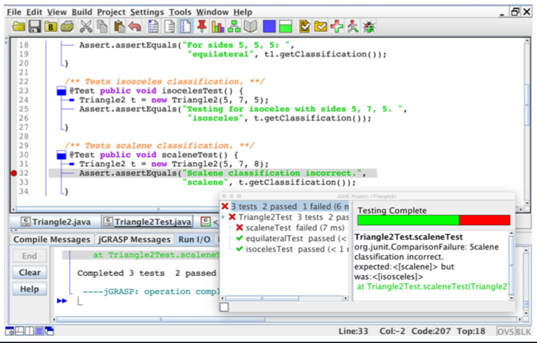
```
/** Tests scalene classification. **/
@Test public void scaleneTest() {
   Triangle2 t = new Triangle2(5, 7, 8);
   Assert.assertEquals("Scalene classification incorrect.",
                       "scalene", t.getClassification());
}
```



```
JUnit: Project: <Triangle2>
3 tests  2 passed  1 failed (< 1 ms)
  Triangle2Test  3 tests  2 passed  1 failed
    scaleneTest  failed (< 1 ms)
    equilateralTest  passed (1 ms)
    isoscelesTest  passed (< 1 ms)

Testing Complete
Triangle2Test.scaleneTest
org.junit.ComparisonFailure: Scalene
classification incorrect.
expected:<[scalene]> but
was:<[isosceles]>
 at Triangle2Test.scaleneTest(Triangle2Te
```

Debugging Failed Test

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



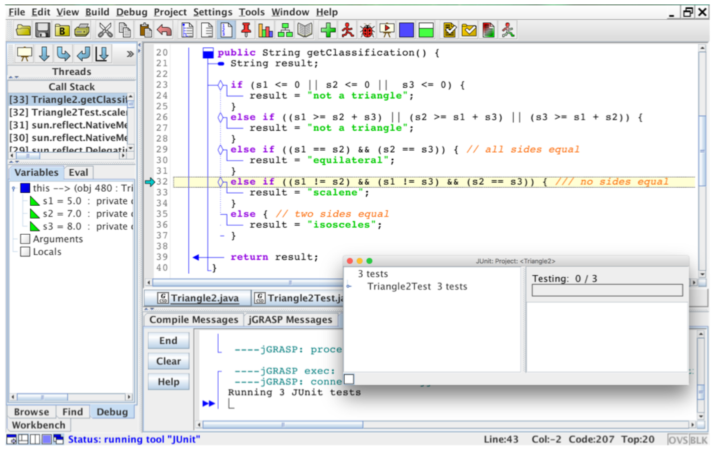
```
         Assert.assertEquals("For sides 5, 5, 5: ",
                          "equilateral", t1.getClassification());
      }

   /** Tests isosceles classification. **/
   @Test public void isocelesTest() {
      Triangle2 t = new Triangle2(5, 7, 5);
      Assert.assertEquals("Testing for isoceles with sides 5, 7, 5. ",
                          "isosceles", t.getClassification());
   }

   /** Tests scalene classification. **/
   @Test public void scaleneTest() {
      Triangle2 t = new Triangle2(5, 7, 8);
      Assert.assertEquals("Scalene classification incorrect.",
                          "scalene", t.getClassification());
   }
```

```
   at Triangle2Test.scaleneT

 Completed 3 tests  2 passed

  ----jGRASP: operation compl
```

```
3 tests  2 passed  1 failed (6 m
  Triangle2Test  3 tests  2 pas
    scaleneTest  failed (7 ms)
    equilateralTest  passed (<
    isocelesTest  passed (< 1

Testing Complete
Triangle2Test.scaleneTest
org.junit.ComparisonFailure: Scalene
classification incorrect.
expected:<[scalene]> but
was:<[isosceles]>
 at Triangle2Test.scaleneTest(Triangle2
```

Finding and Correcting Error

- Find the error, correct it, and run test file



```
public String getClassification() {
   String result;

   if (s1 <= 0 || s2 <= 0 ||  s3 <= 0) {
      result = "not a triangle";
   }
   else if ((s1 >= s2 + s3) || (s2 >= s1 + s3) || (s3 >= s1 + s2)) {
      result = "not a triangle";
   }
   else if ((s1 == s2) && (s2 == s3)) { // all sides equal
      result = "equilateral";
   }
   else if ((s1 != s2) && (s1 != s3) && (s2 == s3)) { /// no sides equal
      result = "scalene";
   }
   else { // two sides equal
      result = "isosceles";
   }

   return result;
}
```

```
[33] Triangle2.getClassit
[32] Triangle2Test.scale
[31] sun.reflect.NativeMe
[30] sun.reflect.NativeMe
[29] sun.reflect.Delegati

this --> (obj 480 : Tri
  s1 = 5.0  :  private
  s2 = 7.0  :  private
  s3 = 8.0  :  private
Arguments
Locals
```

```
 ----jGRASP: proce

 ----jGRASP exec:
 ----jGRASP: conne
Running 3 JUnit tests
```

JUnit: Project: <Triangle2>

```
3 tests
  Triangle2Test  3 tests
```

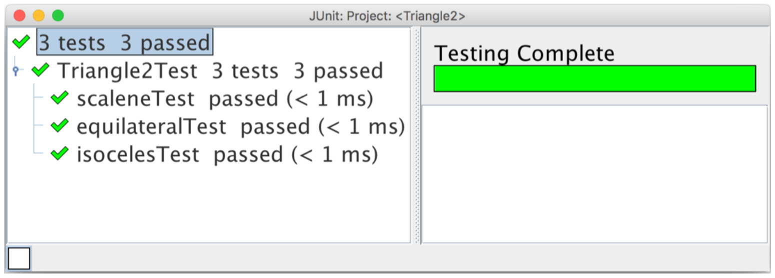
Testing: 0 / 3

Status: running tool "JUnit"

Line:43 Col:-2 Code:207 Top:20

So Far, So Good

- After correcting the error and running the test file again, all tests should pass



JUnit: Project: <Triangle2>

```
✓ 3 tests  3 passed
  ✓ Triangle2Test  3 tests  3 passed
    ✓ scaleneTest  passed (< 1 ms)
    ✓ equilateralTest  passed (< 1 ms)
    ✓ isocelesTest  passed (< 1 ms)
```

Testing Complete

Levels of Test Coverage

How well have we tested?

- **Method coverage** – each method in the source code has been called by the test code
- **Statement coverage** – each statement in the source code has been executed by the test code
- **Condition coverage** – each condition in the in the source code has been exercised both true and false by the test code

  A method with n conditions requires a maximum of n+1 tests to reach condition coverage (assuming carefully chosen tests) but more than n+1 is common

Condition Coverage

- Consider the first part of the if statement in the getClassification method:

```
if (s1 <= 0 || s2 <= 0 ||  s3 <= 0) {
    result = "not a triangle";
}
```

- Three conditions indicate four possible paths for condition coverage (remember, || is a short circuit binary operator: if the first condition is true, the second condition is not executed)
- The four paths are: T X X, F T X, F F T, F F F (where T is true, F is false, X is don't care)
- These test values for the sides cover the conditions:
  (-5, 7, 5)   (5, -7, 5)   (5, 7, -5)   (5, 7, 5)
  - Note that the last one also tests for isosceles

Condition Coverage

- We can create multiple test methods, each with an assert statement, or a single test method with the three assert statements to cover the first part of the if statement in the getClassification method:

```
if (s1 <= 0 || s2 <= 0 ||  s3 <= 0) {
    result = "not a triangle";
}
```

```
/** Tests for negative sides. **/
@Test public void negativeSidesTest() {
   Triangle2 t = new Triangle2(-5, 7, 5);
   Assert.assertEquals("not a triangle",t.getClassification());
   t = new Triangle2(5, -7, 5);
   Assert.assertEquals("not a triangle",t.getClassification());
   t = new Triangle2(5, 7, -5);
   Assert.assertEquals("not a triangle",t.getClassification());
}
```

Other Assert Methods

- Recall, to compare primitives (except float and double) and most objects, we can use the following:

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

- To test <u>float</u> or <u>double</u> values:

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

`delta` – the maximum difference (`delta`) between `expected` and `actual` for which both numbers are still considered equal (i.e., `Math.abs(expected - actual) <= delta`); e.g., 0.000001 compares two doubles to 6 decimal places (10^{-6})

Other Assert Methods

- To test arrays: (double arrays, 3rd/4th parameter is delta)

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

- To test boolean results:

```
Assert.assertTrue(booleanExpression);
Assert.assertTrue(errorMsg, booleanExpression);
Assert.assertFalse(booleanExpression);
Assert.assertFalse(errorMsg, booleanExpression);
```

Example for Triangle2 t:

```
Assert.assertTrue(t.toString().contains("scalene"));
```

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

Errors

- If you get compiler errors like the one below,

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

  then jGRASP likely considers the file to be a source file rather than a test file; you consider the following:

  - Make sure the jGRASP project is open
  - Make sure the test file is in the project and that it's in the Test Files category
  - 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 as needed

Grading System – the Hints

- Recall, the equilateralTest method; the assert statement has an error message to be included in the output if the equilateralTest method fails

```
Assert.assertEquals("For sides 5, 5, 5: ",
   "equilateral", t.getClassification());
```

- This type of output should be familiar to you since our grading system uses JUnit tests to grade your programs. The “hints” provided by the grading system are the error messages in the assert statements.