Using JUnit with jGRASP

Preliminaries

1. **Java JDK –** If you have already installed Java JDK, go to the next step. Otherwise, install Java JDK on your computer – http://java.sun.com/javase/downloads/index.jsp

2. **JUnit** – If you have already installed JUnit, go to the next step. Otherwise, download the the ZIP file version (e.g., junit4.8.2.zip) of the current release of JUnit (http://www.junit.org/). Unzip the installation file in the *program files* folder on Windows or as appropriate on other systems. If you have already installed Checkstyle on your computer, you should use essentially the same procedure to install JUnit.

3. **jGRASP** – If you have already installed **jGRASP 1.8.8_01 or later** (which includes a plug-in for JUint), go to the next step. Otherwise, download and install the latest version of jGRASP (http://www.jgrasp.org/).

4. **JUnit Configuration in jGRASP** – On the toolbar, select **Tools** > **JUnit** > **Configure**. Make sure that **JUnit Home** is set to the folder containing the JUnit executables (e.g., C:\Program Files\JUnit\junit4.8.2). If **JUnit Home** is blank or incorrectly set, click the **Browse** button and set it to the folder containing the JUnit JAR file.

Procedure for Creating, Compiling, and Running a JUnit Test File

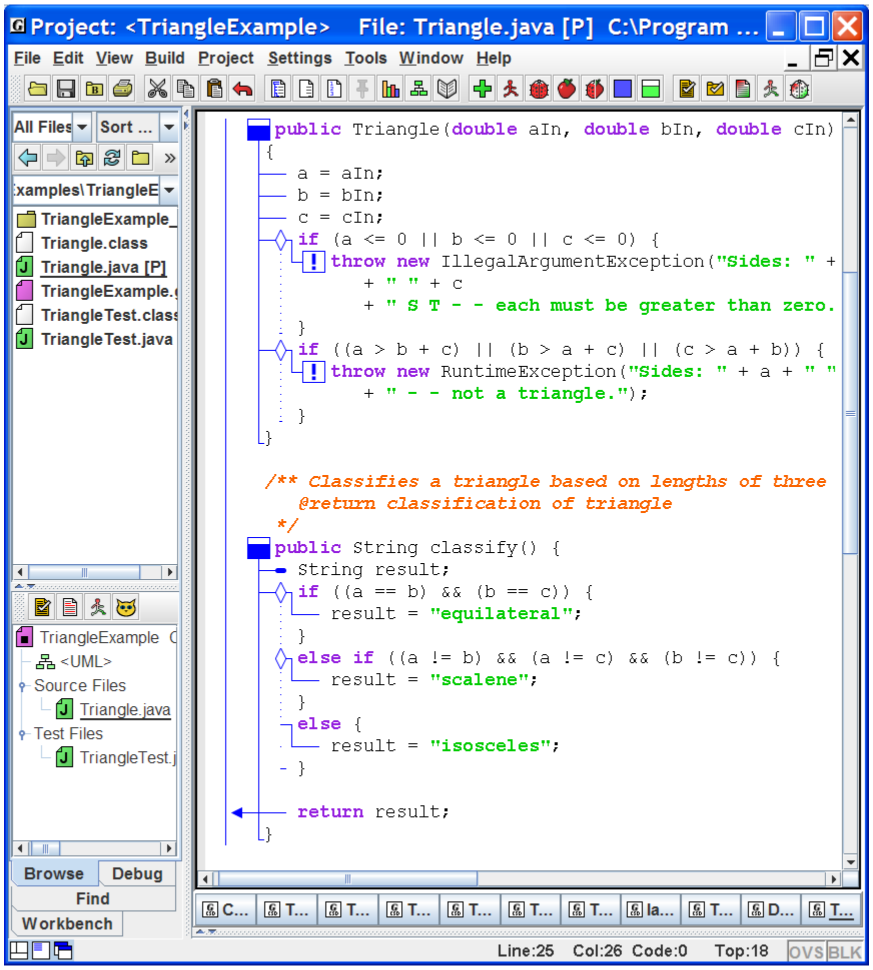
To create a test file for a java source file:

1. Open the source file in jGRASP.
2. On the menu, click **Tools > JUnit > Create Test File**. Alternatively, you can click the Create Test File button on the toolbar. After the test file has been created, this menu option becomes **Edit Test File**. Note that the naming convention expected by JUnit for the test file is "Test" appended to the name of the source file (e.g., for source file Triangle.java, the test file will be TriangleTest.java).
3. Test cases are added to the test file in form of test methods. Note that the defaultTest() method provided by jGRASP asserts that 0 equals 1 (i.e., *expected* value is 0, *actual* value is 1) so that the test will fail. This test method should be replaced by your own relevant test methods. The **Triangle Example** below describes the details of creating, compiling and running test methods.
4. The test file is compiled and run using the options **Tools > JUnit > Compile Tests** and **Tools > JUnit > Run Tests**. Alternatively click the corresponding buttons on the toolbar: to compile the test file and to run the test file.

"Marking" a Test File – If a test file is imported into jGRASP (i.e., not created as indicated above), it must be marked as a test file: **Settings > Compiler Settings > File** then select "Java (Test)" as the Environment, click **Use** then **OK**. When a file is recognized by jGRASP as a test file, the buttons for Create/Edit, Compile, and Run Tests will appear on the toolbar. If you place a test file in a project, it should appear in the Test Files category rather than the Source Files category. If your test file is in the Source Files category, right-clicking on the file and clicking "Mark as Test" will move it to the Test Files category. When both categories are present, you can also simply drag files from one category to another.

Triangle Example

Let's consider the object-oriented version of the classic triangle program. Given three sides (a, b, and c), we must determine: (1) if the sides can form a triangle; i.e., all sides are greater than zero, and no single side is greater than the sum of the other two; and (2) if we have a triangle, classify it as equilateral, scalene, or isosceles. The source code for this example is included with the jGRASP distribution in the jGRASP\examples\Tutorials\JUnitExamples\TriangleExample folder. Figure 1 shows Triangle.java opened in jGRASP. The constructor and classify method are visible in the figure.



```
public Triangle(double aIn, double bIn, double cIn)
{
   a = aIn;
   b = bIn;
   c = cIn;
   if (a <= 0 || b <= 0 || c <= 0) {
      throw new IllegalArgumentException("Sides: " +
         + " " + c
         + " S T - - each must be greater than zero.
   }
   if ((a > b + c) || (b > a + c) || (c > a + b)) {
      throw new RuntimeException("Sides: " + a + " "
         + " - - not a triangle.");
   }
}

/** Classifies a triangle based on lengths of three
   @return classification of triangle
 */
public String classify() {
   String result;
   if ((a == b) && (b == c)) {
      result = "equilateral";
   }
   else if ((a != b) && (a != c) && (b != c)) {
      result = "scalene";
   }
   else {
      result = "isosceles";
   }

   return result;
}
```

Figure 1. Triangle.java

Figure 2 shows TriangleTest.java after several test cases in the form of methods have been added. Note the following:

1. Each test method is annotated with **@Test** in the method header; for example:
   `@Test public void equilateralTest()`

2. The most common way to create test cases is by invoking the Assert.assertEquals method which has the following two forms.

```
public static void assertEquals(String message,
                                Object expected,
                                Object actual)

public static void assertEquals(Object expected,
                                Object actual)
```

3. In the test methods for Test 5 - Test 9, the assert.Equals method with arguments for *message*, *expected*, and *actual* is used. The test input is included in String *message*, and for *actual*, the method to be tested is invoked.

4. For example, in Test 5, which is attempting to determine if the classify method can correctly classify an equilateral triangle, a triangle object is created as follows:

```
  Triangle t = new Triangle(12, 12, 12);
```

This is followed by the assert statement with the following arguments.

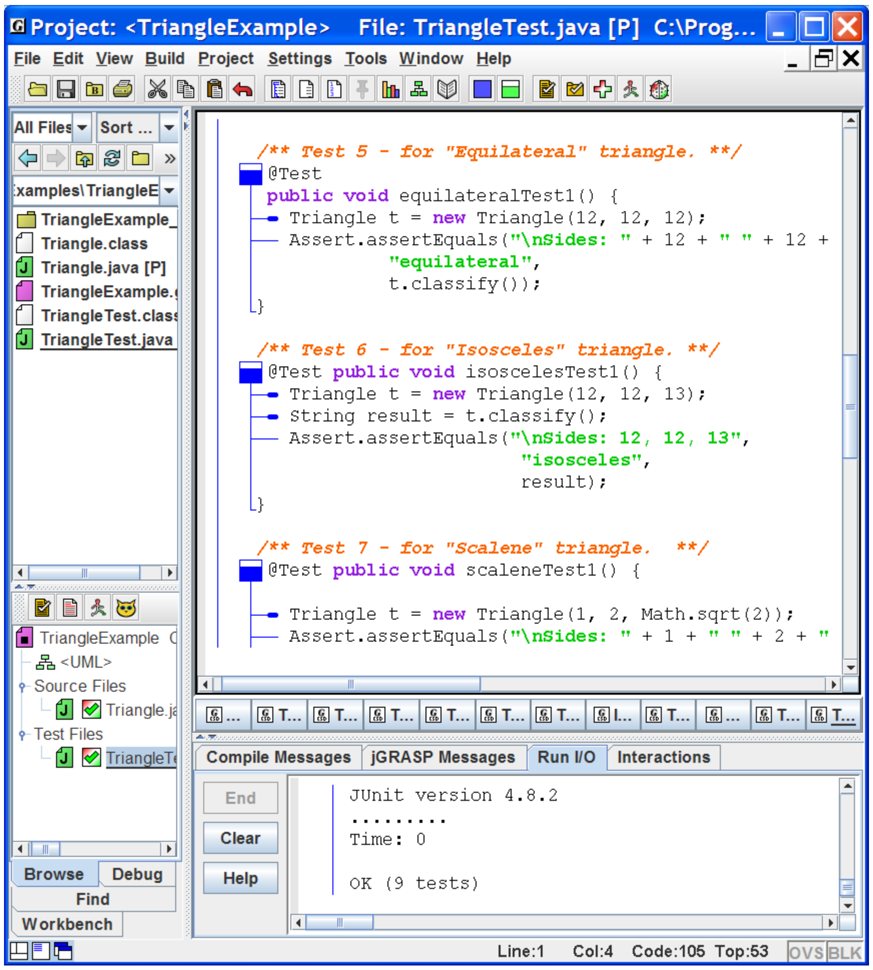
message is set to the String `"\nSides: " + 12 + " " + 12 + " " + 12`

expected is set to `"equilateral"`

actual is set to `t.classify()`

If the value returned by t.classify() does not equal the expected value “equilateral” then the test fails and *message*, *expected*, and *actual* are printed out by JUnit with the failure notification.

```
/** Test 5 - for "Equilateral" triangle. **/
 @Test
 public void equilateralTest1() {
   Triangle t = new Triangle(12, 12, 12);
   Assert.assertEquals("\nSides: " + 12 + " " + 12 + " " + 12,
            "equilateral",
            t.classify());
}
```



```
/** Test 5 - for "Equilateral" triangle. **/
@Test
public void equilateralTest1() {
   Triangle t = new Triangle(12, 12, 12);
   Assert.assertEquals("\nSides: " + 12 + " " + 12 + " " + 12 +
                "equilateral",
                t.classify());
}

/** Test 6 - for "Isosceles" triangle. **/
@Test public void isoscelesTest1() {
   Triangle t = new Triangle(12, 12, 13);
   String result = t.classify();
   Assert.assertEquals("\nSides: 12, 12, 13",
                      "isosceles",
                      result);
}

/** Test 7 - for "Scalene" triangle.  **/
@Test public void scaleneTest1() {

   Triangle t = new Triangle(1, 2, Math.sqrt(2));
   Assert.assertEquals("\nSides: " + 1 + " " + 2 + "
```

```
JUnit version 4.8.2
.........
Time: 0

OK (9 tests)
```

Figure 2. TriangleTest.java

5. When you run JUnit Tests, the results are shown in the Run I/O tab. For example:

```
JUnit version 4.8.2
.........
Time: 0.015

OK (9 tests)
```

Results are also indicated in the project tab. A green check mark ✓ is shown in the JUnit status symbol in front of each test file and associated source file in the project tab to indicate that all tests are passed. A red X indicates at least one test failed. When neither the check mark ✓ or the red X is displayed, it means either the test file or source file as been edited and the tests need to be run again. In Figure 2 above, the green check marks ✓ are displayed in the project tab to indicate that all tests for the Triangle.java source file passed.

6. You can experiment with *failure* by changing Test 5 to make it fail. For example, change one of the sides in the constructor from 12 to 15 and run the tests. Alternatively, misspell “equilateral” as “equilatera”. You can also make the test fail by “making a mistake” in the classify method. A failed test indicates there is either an error in the method you are testing (e.g., the classify method) or an error in the test case itself, or possibly errors in both. When a test passes, it increases the level of confidence in the correctness of the program.

7. If you are new to JUnit and Java, you should probably not be concerned with Test1 - Test 4 in the Triangle example. Test 5 - Test 9 are much more typical. However, if you are familiar with throwing Java exceptions, let’s consider Test 1 and Test 4. Test 1 indicates that it is expecting an exception (IllegalArgumentException.class) to be thrown when we attempt to create a triangle with the first side set to -12 (i.e., a = -12). Since this exception will occur in the constructor, the object will never be created.

```
/** Test 1 - for bad arguments. **/
 @Test(expected = IllegalArgumentException.class)
 public void argumentTest1() {
   Triangle t = new Triangle(-12, 5, 10);
}
```

In the constructor, we have an *if* statement that throws the exception if a side of the triangle is not greater than zero.

```
if (a <= 0 || b <= 0 || c <= 0) {
   throw new IllegalArgumentException("Sides: " + a + " " + b
      + " " + c
      + " S T - - each must be greater than zero.");
}
```

In the case of one side being greater than the sum of the other two, a RuntimeException is thrown in the source file. Test 4 is a check for this.

```
/** Test 4 - for "Not a triangle". **/
 @Test (expected = RuntimeException.class)
 public void notTriangleTest() {
   Triangle t = new Triangle(2, 5, 10);
}
```