James Finn
CPE 470

Handout #8
1/2008

JUnit and Eclipse

JUnit

JUnit is an open-source Java framework for writing and running unit tests.

JUnit documentation can be found at http://junit.sourceforge.net/. You should read:

- *JUnit Cookbook*: http://junit.sourceforge.net/doc/cookbook/cookbook.htm.
- The section in the FAQ on writing tests is also good: http://junit.sourceforge.net/doc/faq/faq.htm#tests.

To use JUnit, you need the JUnit jar file. This is included with Eclipse. If you are using a different Java development environment, you can download it from http://www.junit.org.

Eclipse

To use JUnit in an Eclipse project, you must add the JUnit jar file to the project. The easiest way to do this is to cheat and get Eclipse to do it for you:

1. Start to create a Test class in your project. Put

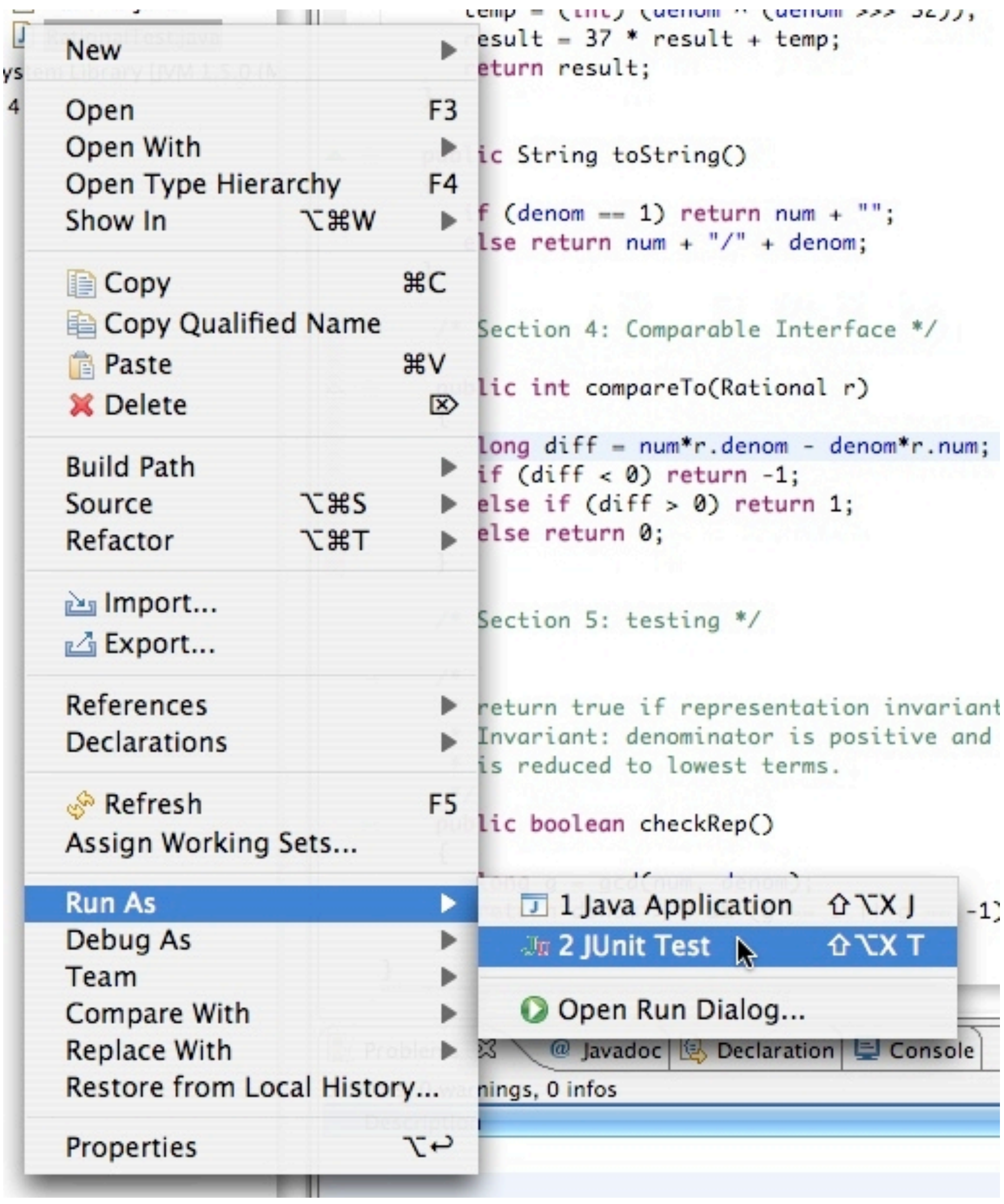
   ```
   import org.junit.*;
   ```

   at the top of the file. This will produce the error message "The import org.junit cannot be resolved." This is because the JUnit jar file is not in the project.

2. Right-click on the red *X* error icon, and select the item "Quick Fix" from the popup menu. *Note*: Quick Fix is a very useful feature. For many basic syntax errors, Eclipse can guess correctly what the problem is and fix it for you!

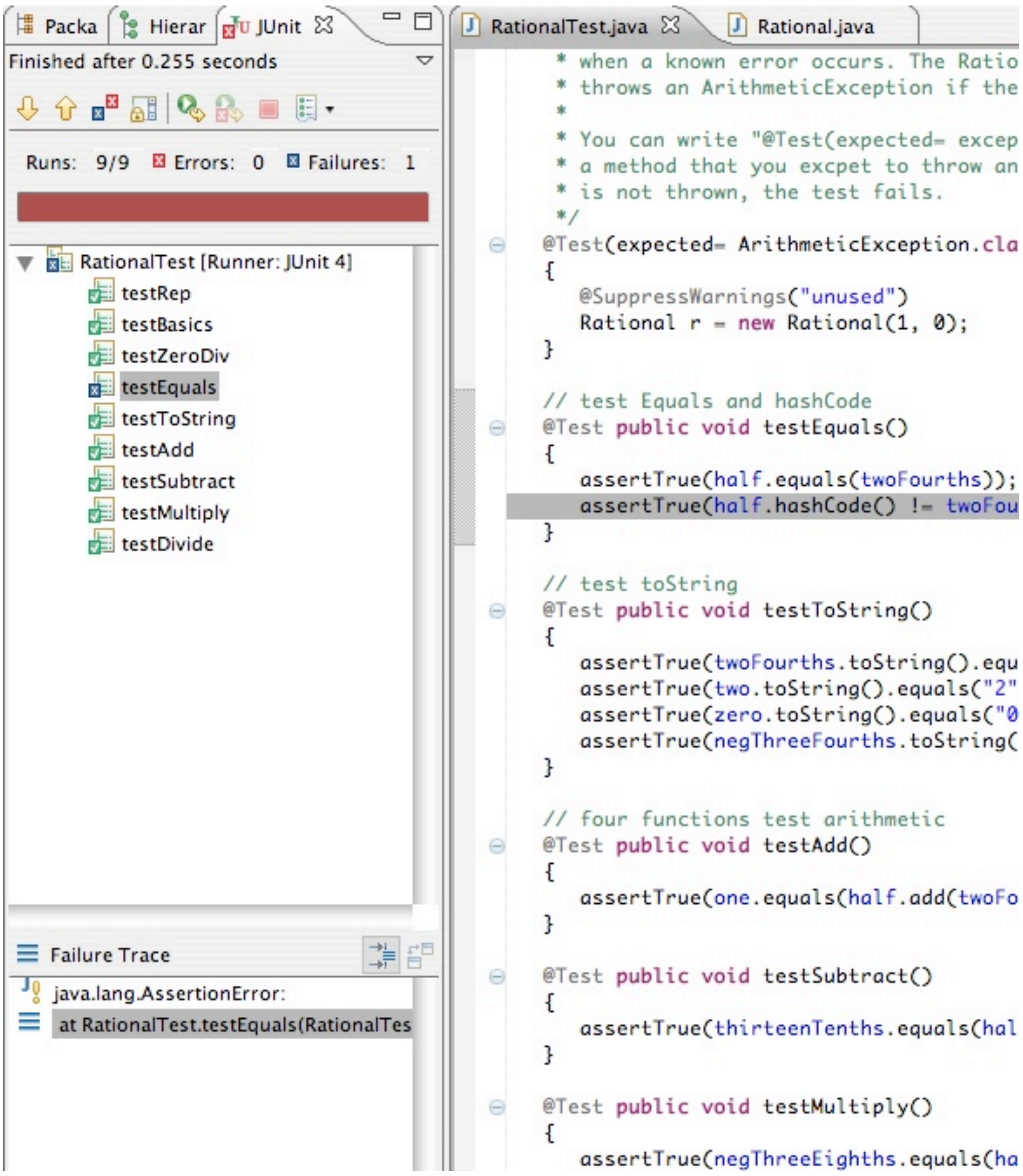
3. The resulting dialog box contains Eclipse's guesses for the cause of the error and how to fix it. One item says (depending on what version of Eclipse you are running) "Fix project setup..." or "Add JUnit 4 to the build path". Double-click on this item and Eclipse will add the JUnit jar file to your project.

The JUnit documentation explains how to write a main program that runs JUnit tests. Eclipse makes life a little easier. The Eclipse Run As... item lets you run the unit tests:



```
temp = (int) (denom ^ (denom >>> 32));
esult = 37 * result + temp;
eturn result;

ic String toString()

f (denom == 1) return num + "";
lse return num + "/" + denom;

Section 4: Comparable Interface */

lic int compareTo(Rational r)

long diff = num*r.denom - denom*r.num;
if (diff < 0) return -1;
else if (diff > 0) return 1;
else return 0;

Section 5: testing */

return true if representation invariant
Invariant: denominator is positive and
is reduced to lowest terms.

lic boolean checkRep()
```

Click on JUnit Test. The tests will be run. Test results appear in a JUnit view (in the same frame as the Navigator and Package views). A green bar will appear if all tests succeeded:, and a red bar appears if there are any failures. A Failure Trace pane will show which test failed; double click on the line and the editor will open with that line selected.



```
Finished after 0.255 seconds

Runs:  9/9    Errors:  0    Failures:  1

RationalTest [Runner: JUnit 4]
    testRep
    testBasics
    testZeroDiv
    testEquals
    testToString
    testAdd
    testSubtract
    testMultiply
    testDivide

java.lang.AssertionError:
at RationalTest.testEquals(RationalTes
```

```
 * when a known error occurs. The Ratio
 * throws an ArithmeticException if the
 *
 * You can write "@Test(expected= excep
 * a method that you excpet to throw an
 * is not thrown, the test fails.
 */
@Test(expected= ArithmeticException.cla
{
   @SuppressWarnings("unused")
   Rational r = new Rational(1, 0);
}

// test Equals and hashCode
@Test public void testEquals()
{
   assertTrue(half.equals(twoFourths));
   assertTrue(half.hashCode() != twoFou
}

// test toString
@Test public void testToString()
{
   assertTrue(twoFourths.toString().equ
   assertTrue(two.toString().equals("2"
   assertTrue(zero.toString().equals("0
   assertTrue(negThreeFourths.toString(
}

// four functions test arithmetic
@Test public void testAdd()
{
   assertTrue(one.equals(half.add(twoFo
}

@Test public void testSubtract()
{
   assertTrue(thirteenTenths.equals(hal
}

@Test public void testMultiply()
{
   assertTrue(negThreeEighths.equals(ha
```