



CS 315

Software Design

Homework 6

Unit testing with JUnit

Due: Oct. 29, 11:30 PM

Objectives

- To perform thorough white box testing
- To gain experience writing equals

JUnit

JUnit is a testing tool for Java. In particular, it helps you create classes that perform testing on other classes. It generates a fair amount of the testing framework for you and also provides a nice user interface to evaluate the results of running your tests.

There are two classes that you will use extensively when working with JUnit: `TestCase` and `TestSuite`. The tests that you write will go in a class that is a subclass of `TestCase`. Within your `TestCase` class, you will create one or more methods to test each method in the class being tested. Typically, a test method will first create some objects needed to perform the test, create an object to hold the expected result, perform the operation, and then compare the actual result to the expected result. For example, to test the `getDegree` method, you might do something like:

```
public void testGetDegree() {
    // Creates a polynomial representing 3x**2
    DensePolynomial p = new DensePolynomial (3, 2);
    int expected = 2;
    assertTrue (p.getDegree() == expected);
}
```

The methods that you will use most frequently to verify results are:

```
assertTrue (boolean condition)
```

The test passes if the condition is true.

```
assertFalse (boolean condition)
```

The test passes if the condition is false.

```
assertEquals (Object o1, Object o2)
```

The test passes if the two objects are equal (using the `equals` method). There are actually many overloads of this method for all the primitive types as well.

`fail()`

Fail unconditionally. You would put this somewhere in your test case that you thought should be unreachable. There is an example later using this to facilitate testing that exceptions are thrown properly.

It is often the case that many of your test methods will use the same objects in their test. To avoid needing to repeat the code, you can place the code in a method called `setUp`. JUnit calls this method before calling each of your test methods. If you use `setUp`, you should not be calling it explicitly in your own code.

JUnit also will build a test suite for you automatically. A test suite contains a main program and will call each of your test methods. As explained later, the JUnit plugin for Eclipse will completely create the `TestSuite` subclass for you. You just need to be sure to run the JUnit wizard to create it! If you are not using Eclipse, please refer to the JUnit documentation to see how to define your `TestSuite` subclass.

Testing exception throwing

It is important to test that methods throw exceptions at the right time. To build a JUnit test method for this, you use an idiom like the following:

```
public void testConstructorShouldThrow() {
    try {
        new DensePolynomial (3, -5);
        fail ("Expected exception because exponent is negative");
    } catch (Exception e) {
    }
}
```

Since we expect the exception to be thrown, we expect the `fail` statement to be skipped, executing the (empty) `catch` statement instead. If it doesn't throw the exception, we reach the `fail` method call instead. `fail` just forces JUnit to report a test failure unconditionally.

More information on JUnit

There is information below about how to use JUnit within Eclipse.

For more information on JUnit, particularly if you are not using Eclipse, visit <http://junit.sourceforge.net/junit3.8.1/index.html>. The FAQ provides useful information on running JUnit tests from the command line. The javadoc API for JUnit can be found at <http://www.junit.org/junit/javadoc/3.8.1/index.htm>. (Note that there is a new release of JUnit, JUnit 4.0, which was only recently integrated with Eclipse. This homework describes the more widely available 3.8.1.)

Assignment

An earlier homework assignment had you create a class and use `assert` statements to check preconditions, invariants, and postconditions. Hopefully, you also wrote some type of test program that gave you at least some confidence that your program is correct. For this assignment, you will do more rigorous testing of the same code. In particular, the goal of this assignment is to do thorough white box testing. You should strive for complete statement and branch coverage and good loop coverage. Also, test boundary conditions.

One goal of testing is to make it as easy as possible for the person running the test to determine if the program is behaving correctly. On the first assignment, you may have displayed output on the screen that you expected someone to read to determine correctness.

This strategy does not scale well. Instead, we usually want our test program to check for correctness and make it blatantly obvious when a test has failed. To accomplish this, you will need to write the `equals` method for `DensePolynomial`. It should report that two polynomials are equal if they represent the same abstract value, independent of whether the parameter passed in is a `SparsePolynomial` or a `DensePolynomial`.

To do your testing, you should use the JUnit test framework. Fortunately, Eclipse comes with a JUnit plug-in, simplifying the process of creating and running JUnit tests.

Eclipse

Creating a JUnit TestCase

Eclipse has a JUnit wizard that can help you create test cases and test suites. To use the wizard, open the File menu, select New and JUnit Test Case. This should open a new window that helps you define the general structure of a JUnit test case.

In the new window, select "New JUnit 3 test". The source folder should be the name of your project. The Class Under Test should be `DensePolynomial`. This may generate a name for the test case of `DensePolynomialTest`. If so, that is fine. If not, you should probably enter that as the name. It should automatically set the superclass to `junit.framework.TestCase`. You can then decide which method(s) you want to override (probably none of them).

Click Next. The wizard then gives you an opportunity to decide which methods that are defined in `DensePolynomial` you want to write test methods for. It also shows you everything that is inherited. You should select all of the methods in `DensePolynomial` except the constructor(s) and `wellFormed`. Be sure to use all the constructors when creating values in your test cases.

Press Finish and a new JUnit test case class is created with the stubs appropriately initialized.

Creating a JUnit TestSuite

The JUnit wizard can also be used to create a test suite. To use the wizard, open the File menu, select New and then Other... In the left column, open Java by clicking on the triangle next to Java. Select Junit and then select `JUnit TestSuite`. The folder should be the name of the project containing `DensePolynomial`. By default, it will call the test suite `AllTests`. That is appropriate if you run all your tests there (recommended). Select your test case class from the list. Press Finish.

You should not need to modify the class that the wizard generates for you.

Telling the compiler about JUnit

While Eclipse knows about JUnit, it does not automatically place `junit.jar` on your classpath. To add JUnit to your class path do the following:

1. Select your project in the Project Explorer.
2. Open the Project menu and select Properties.
3. Select Java BuildPath.
4. Click on the Libraries tab.
5. Click Add Variable...
6. Select `JUNIT_HOME`. Verify that is defined to be `junit 3.8`. (`junit_3.8` should be part of the variable's value, not `junit_4`)

7. Click Extend...
8. Select junit.jar.

Running JUnit tests

To run a JUnit test, select the test class in the package explorer. Click on the triangle to the right of the run button to open a menu. Select Run As and then JUnit Test. JUnit will create a new tab in one of the two panels. It will either be on the left with the package explorer or on the bottom with the console. Click on that tab to find out the results of running JUnit.

If there are errors, the JUnit tab will list each test method that failed. If you click on a test method, it will show you a stack. Clicking on a line in the stack will take you to the corresponding line of source code (if it's in your project source code). So, a good line to click on is one that lists one of your test methods. You will then be able to tell exactly which test failed.

Grading

The primary focus of this homework is on use of JUnit as a way of testing your code. The thoroughness of your tests is important. I will be looking for statement and branch coverage, good testing of loops, and testing of boundary conditions.

Correctness plays a larger role in this assignment. Between writing assert statements in Homework 3 and the test cases in this homework, you should have had a good opportunity to uncover and fix any problems your code might have had.

Everything you did last time should continue to exist: comments, assert statements, good style, etc.

equals	10 points
Statement coverage	15 points
Branch coverage	5 points
Loop testing	5 points
Boundary conditions	5 points
Style	10 points
Comments	10 points
Correctness	40 points

Turning in your work

To turn in your work, create a jar file that contains your Polynomial project, including the `DensePolynomial` source code and your testing code. Email this jar file to me. In your email message, include any instructions I need to know to run your program. My expectation is that I can run your code as a JUnit test from within Eclipse. If this isn't the case, please tell me.