CISC 275: Introduction to Software Engineering

Lab 2:
Unit Testing with

JUnit

Charlie Greenbacker
University of Delaware
Fall 2011

Overview

- What is Unit Testing?
- JUnit at a Glance
- Setting Up JUnit
- Basic Example of Running a Test
- More Methods
- Lab Exercise

What is Unit Testing?

- You already know the answer...
- Isolate & test individual units of code
  - A unit is the smallest testable part of a program
  - In OOP, individual methods would be the units
- Tests show individual parts are correct
  - Multiple tests can check larger parts of programs
- Can begin testing before entire program is done
- Properly designed tests demonstrate proper functionality of code, but bugs may still exist!

JUnit at a Glance

- Unit testing framework for Java language
- Free & Open Source under CPL
- Lets programmers write & run repeatable tests
- Key Features:
  - Assertions for checking expected results
  - Fixtures for sharing test data
  - Framework for running tests

Setting Up JUnit

- As always, there are multiple options...
- JUnit can be run from command line (tricky)
- Eclipse comes with JUnit built-in
  - This is the option we'll use
- Or install Java SDK, Ant build tool, & JUnit

Basic Example – Math.java

- Let's say we want to test a method in a class:

```
public class Math {
   public static int add(int a, int b) {
      return a + b;
   }
}
```

- Looks good, right?

Basic Example – JUnit test

- Create a new JUnit test case in Eclipse
  - Select File/New/JUnit Test Case
  - Select "New JUnit 4 test" radio button
  - Enter Name (usually <classToBeTested>Test)
  - Add JUnit 4 library to the build path
  - Click Finish
- Eclipse will create a skeleton test case .java file for you to start filling in

Basic Example –

- Here's our very simple test case:

```
import static org.junit.Assert.*;
import org.junit.Test;

public class MathTest {
   @Test
   public void testAdd() {
      int sum = Math.add(3, 2);
      AssertEquals(5, sum);
   }
}
```

Basic Example – Running a

- So we've got Math.java and TestMath.java
- Now we can run our test:
  - We can do this from right inside Eclipse
  - With the focus on TestMath.java, select Run/Run As/JUnit Test
- The JUnit test will be executed, exercising your code by running the test cases
  - Results will be displayed in system message panel at bottom of screen
  - Success: green bar; otherwise: red bar & messages

More Methods

- Assertion statements:
  - assertEquals(expected, actual)
  - assertEquals(message, expected, actual)
  - assertTrue(message, condition)   [or assertFalse]
  - assertNull(message, object)   [assertNotNull]
  - assertSame(expected, actual)   [assertNotSame]
  - etc.
- Using messages can help clarify what went wrong when complicated/compound tests fail

Lab Exercise – Overview

- Objective: get practice writing JUnit test cases
- On your own (or with a partner), you will write & run test cases for a family history parser class
  - This class contains methods for parsing family history event listings (e.g. births, marriages, etc.) into an output format specifying the event type, one or more participants' names, & the date
  - Many of the tests have been written for you; you will write tests for a single, simple helper method

Lab Exercise – Preparation

- Open Eclipse & start a new Java project named "Familiar"
- Create a new Java class named "Parser"
  - www.cis.udel.edu/~charlieg/labs/Parser.java
- Create a new JUnit test case named "ParserTest"
  - www.cis.udel.edu/~charlieg/labs/ParserTest.java
- Carefully read & understand the Parser class & methods, as well as the existing test cases

Lab Exercise – Writing Test Cases

- Most methods already have test cases
- You only need to write tests for capitalizeFirst()
  - This method takes a String & returns a new String with the first letter of each word capitalized
- Examine the existing test cases to get ideas about what kinds of tests to write
  - Be sure to include some with good input, "weird" input, test out the boundaries & null input too
  - Your testCapitalizeFirst() should contain <u>at least 4</u> assertions <u>at a minimum</u>
  - Only need to use assertEquals()

Lab Exercise (cont...)

- Links to lots of useful additional information about JUnit are available on my website...
  - Cookbook, tutorial, using JUnit with Eclipse, etc.
- File containing several examples of "good" input for the parser:
  - www.cis.udel.edu/~charlieg/labs/ParserSamples.txt
- Email your test cases (the ParserTest.java file) to charlieg@cis.udel.edu by Tuesday, Sept. 13
  - Be sure your name[s] (2 people max) are in the email and in all attached files