JUnit
Automated Software Testing Framework
Paul Ammann & Jeff Offutt
http://www.cs.gmu.edu/~offutt/softwaretest/

What is JUnit?
• Open source Java testing framework used to write and run repeatable automated tests
• JUnit is open source (junit.org)
• A structure for writing test drivers
• JUnit features include:
  – Assertions for testing expected results
  – Test features for sharing common test data
  – Test suites for easily organizing and running tests
  – Graphical and textual test runners
• JUnit is widely used in industry
• JUnit can be used as stand alone Java programs (from the command line) or within an IDE such as Eclipse

JUnit Tests
• JUnit can be used to test …
  – … an entire object
  – … part of an object – a method or some interacting methods
  – … interaction between several objects
• It is primarily for unit and integration testing, not system testing
• Each test is embedded into one test method
• A test class contains one or more test methods
• Test classes include :
  – A test runner to run the tests (main())
  – A collection of test methods
  – Methods to set up the state before and update the state after each test and before and after all tests
• Get started at junit.org

Writing Tests for JUnit
• Need to use the methods of the junit.framework.Assert class
  – javadoc gives a complete description of its capabilities
• Each test method checks a condition (assertion) and reports to the test runner whether the test failed or succeeded
• The test runner uses the result to report to the user (in command line mode) or update the display (in an IDE)
• All of the methods return void
• A few representative methods of junit.framework.Assert
  – assertEquals (Object, Object)
  – assertNull (Object)
  – Fail (String)
Example JUnit Test Case

```java
public class Calc {
    public long add (int a, int b) {
        return a + b;
    }
}
```

```java
import org.junit.Test
import static org.junit.Assert.*;
public class calcTest
{
    private Calc calc;
    @Test public void testAdd() {
        calc = new Calc();
        assertEquals ((long) 5, calc.add (2, 3));
    }
}
```

Sample Assertions

- static void assertEquals (boolean expected, boolean actual)
  Asserts that two booleans are equal
- static void assertEquals (byte expected, byte actual)
  Asserts that two bytes are equal
- static void assertEquals (char expected, char actual)
  Asserts that two chars are equal
- static void assertEquals (double expected, double actual, double delta)
  Asserts that two doubles are equal, within a delta
- static void assertEquals (float expected, float actual, float delta)
  Asserts that two floats are equal, within a delta
- static void assertEquals (int expected, int actual)
  Asserts that two ints are equal
- For a complete list, see
  – http://junit.sourceforge.net/javadoc/org/junit/Assert.html

JUnit Test Fixtures

- A test fixture is the state of the test
  - Objects and variables that are used by more than one test
  - Initializations (prefix values)
  - Reset values (postfix values)
- Different tests can use the objects without sharing the state
- Objects used in test fixtures should be declared as instance variables
- They should be initialized in a @Before method
  - JUnit runs them before every @Test method
- Can be deallocated or reset in an @After method
  - JUnit runs them after every @Test method

Testing the Immutable Stack Class

```java
public class Stack {
    public String toString() {
        // EFFECTS: Returns the String representation
        // of this Stack from the top to the bottom.
        StringBuffer buf = new StringBuffer ("{");
        for (int i = size-1; i >= 0; i--)
        {
            if (i < (size-1))
                buf.append (", ");
            buf.append (elements[i].toString());
        }
        buf.append ("}"");
        return buf.toString();
    }
    public boolean repOk() {
        if (elements == null) return false;
        if (size != elements.length) return false;
        for (int i = 0; i < size; i++) {
            if (elements[i] == null)  return false;
        }
        return true;
    }
}
```
Stack Test Class

- Classes to import:
  ```java
  import org.junit.After;
  import org.junit.Before;
  import org.junit.Test;
  import static org.junit.Assert.assertEquals;
  import junit.framework.JUnit4TestAdapter;
  ```

- Pre-test setup method (prefix):
  ```java
  private Stack stack;
  // set up method using @Before syntax
  // @Before methods are run before each test
  @Before public void runBeforeEachTest()
  {
      stack = new Stack();
  }
  ```

- Post-test teardown method (postfix):
  ```java
  // tear-down method using @After
  // @After methods are run after each test
  @After public void runAfterEachTest()
  {
      stack = null;
  }
  ```

Stack Test Cases

- Test public void testToString()
  ```java
  stack = stack.push (new Integer (1));
  stack = stack.push (new Integer (2));
  assertEquals ("{2, 1}", stack.toString());
  ```

- Test public void testRepOk()
  ```java
  boolean result = stack.repOk();
  assertEquals (true, result);
  ```

Running from a Command Line

- This is all we need to run JUnit in an IDE (like Eclipse)
- We need a main() for command line execution ...
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import junit.framework.JUnit4TestAdapter;

// This section declares all of the test classes in the program.
@RunWith (Suite.class)
@Suite.SuiteClasses ({ StackTest.class })  // Add test classes here.
public class AllTests {
  // Execution begins at main(). In this test class, we will execute
  // a text test runner that will tell you if any of your tests fail.
  public static void main (String[] args) {
    junit.textui.TestRunner.run (suite());
  }
  // The suite() method is helpful when using JUnit 3 Test Runners or Ant.
  public static junit.framework.Test suite() {
    return new JUnit4TestAdapter (AllTests.class);
  }
}

How to Run Tests

• JUnit provides test drivers
  – Character-based test driver runs from the command line
  – GUI-based test driver-junit.swingui.TestRunner
    • Allows programmer to specify the test class to run
    • Creates a “Run” button

• If a test fails, JUnit gives the location of the failure and any exceptions that were thrown

JUnit Resources

• Some JUnit tutorials
  – http://open.ncsu.edu/se/tutorials/junit/ (Laurie Williams, Dright Ho, and Sarah Smith)
  – http://www.laliluna.de/eclipse-junit-testing-tutorial.html (Sascha Wolski and Sebastian Hennebrueder)
  – http://www.clarkware.com/articles/JUnitPrimer.html (Clarkware consulting)
• JUnit: Download, Documentation
  – http://www.junit.org/