Testing with JUnit

`JUnit` is a modern testing tool designed by Eric Gamma and Kent Beck (promoter of "Extreme Programming") that is available from `www.junit.org`. JUnit is a tool that reflects the philosophy of test-driven development. Test-driven development involves writing "unit" tests first, and only then writing the actual code. Unit testing is easier and more effective (at least in the early stages of programming) than writing tests of the entire project. The goal is that you never integrate a class into a project until you are convinced that it works properly. Moreover, because we often modify classes, we keep the unit test so that we can rerun them after any changes are made. While anyone can do such testing, JUnit is a tool that helps you set up the tests and then runs them all automatically for you – a huge advantage!

We will only provide a brief overview here. You can learn more on your own. JUnit is integrated into Eclipse, so it is especially easy to use it when using Eclipse. Please read the tutorials that are available from the course web page for details, as you will be using it in lab this week. Because we will be using JUnit from within Eclipse, be sure to look at the JUnit documentation for Eclipse.

To create a test case for an existing project:

1. Create a class extending junit.framework.TestCase. The best way to do this is go select `New` and then JUnit Test Case.
   (a) When the New JUnit Test Case window pops up, click on the choice button on top to select New JUnit 4 test.
   (b) Type in the name of your test class in the name field. If the class you will be testing already exists, enter it in the `Class under Test` field (you can also use the `browse` button to find it if you'd like). Click on the `Next` button.
   (c) You will get a window showing the methods of the class you will be testing (if you selected one). Click in the checkboxes to have method headers automatically generated for each selected method. Click on `Finish` and your class will be displayed. (The system may inform you that JUnit 4 is not on the build path. If so, select "Add JUnit 4 library to the build path" and select OK.)
2. The generated class will import `org.junit.Assert.*` and `org.junit.Test`. Also all of the methods that have stubs created will have an annotation `@Test`. The names of all of these methods begin with `test`, take no parameters and return `void`. They all start with a default body `fail("Not yet implemented")`.

   If you wish to add your own test methods make sure that they follow the same template. You may need some instance variables to be declared and initialized to run your tests. Declare the instance variables as usual, and initialize them in a method with header `public void setUp()` (which may have been added by the wizard). That method should start with the annotation `@Before`. (If the compiler objects to that annotation, add `import org.junit.Before;`.) If you need to put away resources (e.g., close files) after a test, add the method
3. Write the body for each method. The tests should use methods like `assertEquals(a,b)` and `assertTrue(msg,cond)`. The `setUp` method will be run before each test.

To run a test case:

1. Select the test class in the Package Explorer pane on the left side of the Eclipse window.

2. Pull down the menu to the right of the triangle denoting program execution and select `run as ...` and then select `JUnit test`.

A JUnit panel should replace the PackageExplorer on the leftmost panel in the window. If the bar across the top is green, then all tests have succeeded. If one or more tests have failed then the bar will be red and each failed test will be listed. Select each to find out why the test failed.

The `JUnit` panel on the left side of the screen can be replaced by the usual `Package Explorer` panel if you just click on that tab on the top of the panel.

Continue to refine and add to your tests (just add a method whose name starts with `test` and code until you are convinced that all of the methods in your class work correctly. At that point you are ready to integrate your class with the other classes in your project and test them. Again, you can write test suites for these using `JUnit`.

The recommended way of working with `JUnit` is always to write your tests before writing the code that uses them. Thus when you start, all of your tests should fail. Your goal is to fix the failed tests one at a time until you get a green bar. If you had a good test suite then your class should be correct. (Though if you had a lousy test suit, you may still have errors!)

At the beginning of Wednesday lab, please bring in a test class for the State class. I have given you a partially written test class. You should include the bodies of methods that test multiplication, subtraction, division, and the clear method.

```
import static org.junit.Assert.*;
import javax.swing.JTextField;
import org.junit.Test;
import org.junit.Before;

/*
 * @author Kim Bruce
 * @version Created on Apr 4, 2004, revised 2/2011
 * Test class for State class in Calculator
 */
public class StateTest extends TestCase {
private JTextField display; // dummy display for testing
        private State st;        // sample state object

    // set up state with 2 on top of stack and 9 below it.
@Before
        public void setUp(){
                display = new JTextField("0");
                st = new State(display);
                st.addDigit(9);
                st.enter();
                st.addDigit(2);
                st.enter();
        }

    // check that if 2 on stack and add digits 7, 5, and then
    // hit enter that display will show "75"
        public void testAddDigit() {
@Test
```

```
                assertEquals("2",display.getText());
                st.addDigit(7);
                assertEquals("7",display.getText());
                st.addDigit(5);
                assertEquals("75",display.getText());
                st.enter();
                assertEquals("75",display.getText());
        }

    // check that if stack has 4, 2, and 9, then executing plus twice
    // gives "6" and then "15".
@Test
        public void testPlus() {
                st.addDigit(4);
                st.enter();
                st.doOp('+');
                assertEquals("6",display.getText());
                st.doOp('+');
                assertEquals("15",display.getText());
        }

@Test
        public void testMult() {
        }


@Test
        public void testMinus() {
        }

        public void testDiv() {
@Test
        }

@Test
        public void testClear() {
        }

    // check if 2 and 9 on stack that if pop once get "9" in display
    // and if pop again have "0" on stack and remains if pop once more.
@Test
        public void testPop() {
                st.pop();
                assertEquals("9",display.getText());
                st.pop();
                assertEquals("0",display.getText());
                st.pop();
                assertEquals("0",display.getText());
        }

// Method exchange should swap top two values so after
// exchange have 9 on top with 2 underneath.
@Test
public void testExchange() {
st.exchange();
assertEquals("9",display.getText());
st.pop();
assertEquals("2",display.getText());
}

}
```

TestSuites You can create a test suite by selecting `New->Other ...` Select JUnit as before, but create a TestSuite this time instead of a TestCase. This will create a new class AllTests that will create a new TestSuite for every class that starts with "Test". When executed, it will run every test for every class.