What is JUnit?

- Open source Java testing framework used to write and run repeatable automated tests
- 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 is used to test …
  - … an entire object
  - … part of an object – a method or some interacting methods
  - … interaction between several objects

- A tester class contains more than one test
  - Each test is written into one test method

- 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 back 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
  - assertTrue (boolean)
  - assertTrue (String, boolean)
  - assertEquals (Object, Object)
  - assertNull (Object)
  - Fail (String)
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 concerning a delta.

- static void assertEquals (float expected, float actual, float delta)
  
  Asserts that two floats are equal concerning a delta.

- static void assertEquals (int expected, int actual)
  
  Asserts that two ints are equal.

- For a complete list, see
  

How to Write A Test Case

- You may occasionally see old versions of JUnit tests
  
  - Major change in syntax and features in JUnit 4.0
  - Backwards compatible (JUnit 3.X tests still work)

- In JUnit 3.X
  1. import junit.framework.*
  2. extend TestCase.
  3. name the test methods with a prefix of ‘test’
  4. validate conditions using one of the several assert methods

- In JUnit 4.0 and later:
  
  - Do not extend from Junit.framework.TestCase
  - Do not prefix the test method with “test”
  - Use one of the assert methods
  - Run the test using JUnit4TestAdapter
  - @NAME syntax introduced

- We will focus entirely on JUnit 4.X
JUnit Test Fixtures

- A test fixture is the state of the test
  - Objects and variables that are used by more than one test
  - Initializations
  - Reset 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
- Can be deallocated or reset in an @After method

Example JUnit Test Case

```java
import org.junit.Test
import static org.junit.Assert.*;
public class CalcTest
{
    @Test public void testAdd()
    {
        // Calc().add() returns long,
        // so we must cast 5
        assertEquals ((long) 5,
        new Calc().add (2,3));
    }
}
```
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();
    }
}
```

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;
  ```

• Setup operations:
  ```java
  private Stack stack;
  // setUp method using @Before syntax
  // @Before methods are run before each test
  @Before public void runBeforeEachTest()
  {
      stack = new Stack();
  }
  ```

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

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

A problem with this test is that it actually combines four separate tests in one method

```java
@Test public void testRepOk()
{
    boolean result = stack.repOk();
    assertEquals (true, result);
    stack  = stack.push (new Integer (1));
    result = stack.repOk();
    assertEquals (true, result);
    stack  = stack.pop();
    result = stack.repOk();
    assertEquals (true, result);
    stack = stack.push (new Integer (1));
    stack.top();
    result = stack.repOk();
    assertEquals (true, result);
}
```

Without automation, large tests have the advantage of reducing costs of running many tests

With automation, small tests allow us to more easily identify failures …

Stack Test Cases (2)

```java
@Test public void testRepOkA()
{
    boolean result = stack.repOk();
    assertEquals (true, result);
}
```

```java
@Test public void testRepOkB()
{
    stack  = stack.push (new Integer (1));
    result = stack.repOk();
    assertEquals (true, result);
}
```

```java
@Test public void testRepOkC()
{
    stack  = stack.pop();
    result = stack.repOk();
    assertEquals (true, result);
}
```

```java
@Test public void testRepOkD()
{
    stack = stack.push (new Integer (1));
    stack.top();
    result = stack.repOk();
    assertEquals (true, result);
}
```
Running from a Command Line

- This is all that is needed to run JUnit in an IDE (like Eclipse)

- We need a main() for command line execution …

AllTests

```java
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/](http://open.ncsu.edu/se/tutorials/junit/)  
    (Laurie Williams, Dright Ho, and Sarah Smith)
  - [http://www.laliluna.de/eclipse-junit-testing-tutorial.html](http://www.laliluna.de/eclipse-junit-testing-tutorial.html)  
    (Sascha Wolski and Sebastian Hennebrueder)
    (Diaspar software)
  - [http://www.clarkware.com/articles/JUnitPrimer.html](http://www.clarkware.com/articles/JUnitPrimer.html)  
    (Clarkware consulting)

- **JUnit: Download, Documentation**
  - [http://www.junit.org/](http://www.junit.org/)
Summary

• The only way to make testing efficient as well as effective is to automate as much as possible
• JUnit provides a very simple way to automate our unit tests
• It is no “silver bullet” however … it does not solve the hard problem of testing:

What tests to run?

• This is the purpose of test criteria
• JUnit also does not help with integration or system testing