Testing using JUnit version 4.4

977-374 Software Validation and Verification

Dr. Adisak Intana

Lecturer

Writing Unit Tests with JUnit

- At the top of the file, include:
  - ☑ **import org.junit.Test**

- The main class of the file must:
  - ☑ **be public**

Writing Unit Tests with JUnit

- Example:

```
import org.junit.Test;

public class CalculatorTest {


}
```

Writing Unit Tests with JUnit

- **Methods of this class to be:**
  - ☑ **be public and not static**
  - ☑ **return void**
  - ☑ **take no arguments**
  - ☑ **have a name beginning with “test”**
  - ☑ **add keyword @Test before test method**

Writing Unit Tests with JUnit

- **Example:**

```
@Test
public void testAddition(){
   Calculator calc = new Calculator();
   int expected = 7;
   int actual = calc.add(3,4);
   assertEquals("adding 3 and 4", expected,
   actual);
}
```

Writing Unit Tests with JUnit

- Test methods in this class can call any of the following methods:
  - ☑ **void assertTrue(message, condition)**
  - ☑ **void assertTrue(condition)**
  - ☑ **void assertFalse(message, condition)**
  - ☑ **void assertFalse(condition)**

which issues an error report with given message if the condition is false (assertTrue) and true (assertFalse).

Writing Unit Tests with JUnit

- Example;
  - ☑ **assertTrue("Error: value1 is not greater than value2", 3>5)**
  - ☑ **assertTrue(3>5)**

Writing Unit Tests with JUnit

- ☑ **void assertEquals(expected, actual)**
- ☑ **void assertEquals(message, expected, actual)**

which issues an error report with the given message if the two integers are not equal. The first int is expected value, and the second int is the actual (tested) value.

Writing Unit Tests with JUnit

Example:

- ☑ **int expected = 370, actual=400;**
- ☑**assertEquals(“Error: Value1 is not equal Value2”, expected, actual)**
- ☑**assertEquals(expected, actual)**

Writing Unit Tests with JUnit

- ☑ **void assertEquals(expected, actual, delta)**
- ☑ **void assertEquals(message, expected, actual, delta)**

which issues an error report with the given message if the two floatings or doubles are not equal. The first int is expected value, and the second int is the actual (tested) value.

Math.abs(expected - actual) < delta

Writing Unit Tests with JUnit

- ☑ **double expected = 370.991, actual=370.99;**
- ☑ **void assertEquals("Error: Value1 is not equal to Value2", expected, actual, 0.0)**
- ☑ **void assertEquals(expected, actual, 0.0)**

- ☑ **double expected = 370.991, actual=370.99;**
- ☑ **void assertEquals("Error: Value1 is not equal to Value2", expected, actual, 0.1)**
- ☑ **void assertEquals(expected, actual, 0.1)**

Writing Unit Tests with JUnit

- ☑ **void fail()**
- ☑ **void fail(message)**

Causes the current test to fail. This is commonly used with exception handling.

Catch the exception and if it isn't thrown call the fail method. Fail signals the failure of a test case.

Writing Unit Tests with JUnit

- Example;

```
public void testDivision(){
    Calculator calc = new Calculator();
    //Divide by zero shouldn't work
  try{
      calc.divide(2, 0);
        fail("Should have thrown an exception !");
  }catch(ArithmeticException e){
          // Good that's what we expect.
   }
}
```

Writing Unit Tests with JUnit

- ☑ void assertNotNull(object)
- ☑ void assertNotNull(message, object)
- ☑ assertNull(object)
- ☑ assertNull(message, object)
- ☑ assertNotSame(expected, actual)
- ☑ assertNotSame(message, expected, actual)
- ☑ assertSame(expected, actual)
- ☑ assertSame(message, expected, actual)

Writing Unit Tests with JUnit

- If there is any common setup work to be done before running each test (such as initializing instance variables), do it in the body of a method with the following contract:
  - ☑ **@Before**
  - ☑ **protected void setUp()**

Writing Unit Tests with JUnit

- Example:

```
@Before
protected void setUp() {
        myList = new java.util.ArrayList();
}
@Test
public void testAddArr()
{ Product p = new Product("001", "P1");
  myList.addItem(p); }
@Test
public void testRemoveArr()
{ Product p = new Product("001", "P1");
  myList.removeItem(p); }
```

Writing Unit Tests with JUnit

- If there is any common clear work after running each test (such as clearing instance variables), do it in the body of a method with the following contract:
  - ☑ **@After**
  - ☑ **protected void tearDown()**

Writing Unit Tests with JUnit

- Example:

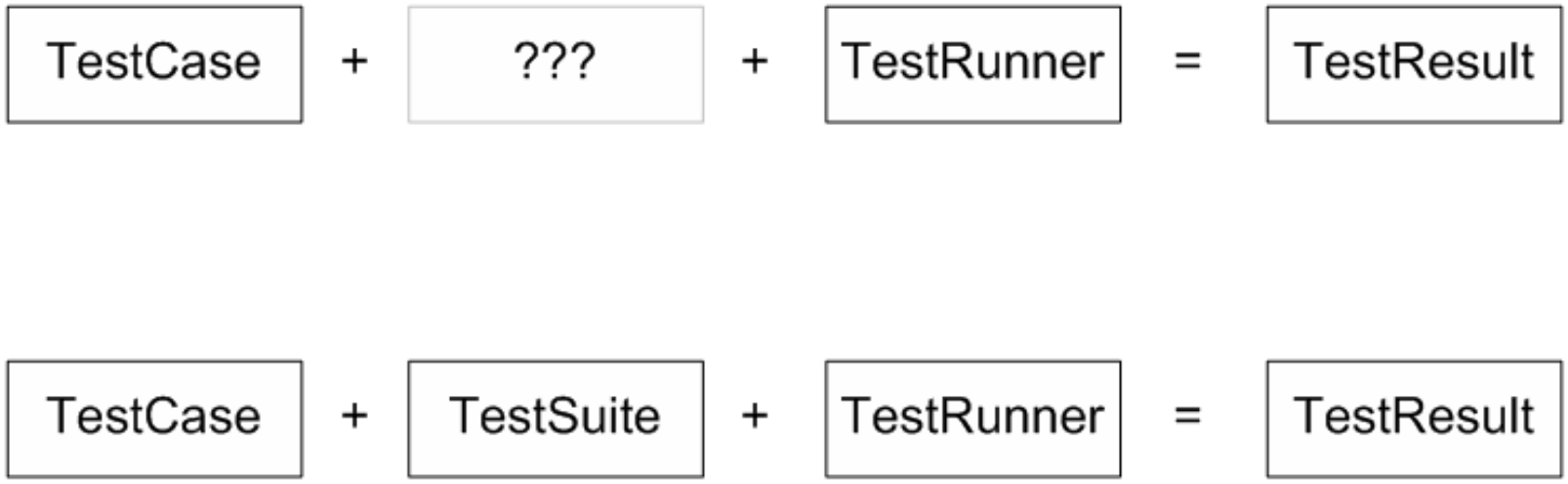
```
@Before
protected void setUp() {
        myList = new java.util.ArrayList();
}

@After
protected void tearDown() {
        myList = null;
}
```

Test Suit

How can we group test cases ??

JUnit Framework

Test Suit


TestCase
+
???
+
TestRunner
=
TestResult
TestCase
+
TestSuite
+
TestRunner
=
TestResult

Writing Unit Tests with JUnit

- Example:

```
public class CalculatorTest {
        @Test
        public void testAddition() {

         }

        @Test
         public void testDivision() {

         }
}
```

Writing Unit Tests with JUnit

- Example:

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses({
  CalculatorTest.class
  })

public class AllTests {
}
```