Testing

Software Testing

• Testing is a critical part of the design and implementation process.
• There are different types of testing with different goals…
• Ideally you want to test all possible ways your program can run and confirm that the output and side effects are all correct.
• Testing individual modules helps identify flaws in the "foundation" that should be fixed before other things are built upon it.
Regression Testing

• What happens if you thoroughly test a module and then decide it needs to be modified at a later time?
  – You might want to retest the entire module.
  – You at least want to retest anything that depends on the things that were changed.

• It would be useful to have a way of "saving" your testing scenarios and the checks used to confirm that the output and side effects are correct.

Module Testing Approaches

• There are many approaches..
  – Manually run through scenarios and check results.
  – Write test drivers and various input and output files for comparison of actual to expected output.
  – Use one of the xUnit family of tools such as…
    • JUnit in Java
    • NUnit in .NET
    • CppUnit in C++
    • etc.

• There are research projects exploring new and different testing approaches (such as the GUITAR and ICE projects here at UM).
JUnit Testing (I)

We will now see a resource available in Java called JUnit testing.

Each test is a method of the form:

```java
public void testTest() {
    //Code and tests here…
}
```

JUnit Testing (II)

Two basic test assertions are:

```java
assertTrue(boolean_expression);
assertFalse(boolean_expression);
```

These can appear anywhere in the body of a testing method.

If the assertion test succeeds, execution of the body continues. If it fails, the test fails and execution of the body of that test stops but if there are more test methods they are run.
JUnit Testing (III)

assertSame(var_1, var_2);

This checks whether these two variables contain the same values. For primitives this acts in a slightly unusual way behind the scene but acts as expected. For object references, this essentially checks whether the two object references "point" to the same memory location.

JUnit Testing (IV)

assertEquals(var_1, var_2);

For primitives this acts as expected. For object references this is an "interesting" one because it uses a very specific form of the equals operator to check whether these two variables are equal. If the class of the objects passed in implement the exact equals operator Java expects (which our self-written Student class doesn't yet) it works as expected.

NOTE: We will explore this more later…
Add a JUnit Test Case

import junit.framework.TestCase;

public class MyTestCases extends TestCase {
    //You put your test cases here...
}

A Test Set is a Class
Constructors, Setters and Getters

```java
public void test1name() {
    String n = "Fred";
    StudentTwo s = new StudentTwo(n, 6, 1234568);
    assertTrue(s.getName().equals(n));
}
```

Other Methods

```java
public void test1ident() {
    StudentTwo s =
        new StudentTwo("Fred", 6, 1234508);
    assertTrue(s.getLastFourIdentDigits() == 4508);
}
```
Test Scenarios (I)

```java
public void test1tokens() {
    StudentTwo s =
        new StudentTwo("Fred", 6, 1234568);

    for (int i=0; i<4; i++) {
        s.useToken();
    }

    assertTrue(s.getTokenLevel() == 2);
}
```

Test Scenarios (II)

```java
public void test2tokens() {
    StudentTwo s =
        new StudentTwo("Fred", 3, 1234568);

    for (int i=0; i<4; i++) {
        s.useToken();
    }

    assertTrue(s.getTokenLevel() == 0);
}
Test Scenarios (III)

public void test3tokens() {
    StudentTwo s =
        new StudentTwo("Fred", 3, 1234568);

    for (int i=0; i<4; i++) {
        s.useToken();
    }

    assertFalse(s.useToken());
}

What to test? How to design them?

• "Everything"
  – Make sure that every method is tested (both public ones and others).
  – Think about the "corner cases".
  – Try some random combinations of scenarios.
  – It would be good if you could make sure that every decision branch is tested.

• It can be useful to actually write your tests (or at least some of them) based on your specs rather than writing them after you implement a module.
Copyright © 2010-2011 : Evan Golub