White box testing

Advanced Programming

White-box testing

- White-box testing is a verification technique software engineers can use to examine if their code works as expected

Types of WBT

- Unit testing (we focus on unit testing in the following)
- Regression testing
- Integration testing
WBT with unit testing

- Each time you write a code module, you should write test cases for it
  - A possible exception to this recommendation is the accessor methods (i.e., getters and setters) of your projects
- You should concentrate your testing effort on code that could easily be broken
  - Generally, accessor methods will be written error-free

Scaffolding

- Scaffolding is a method for unit testing
- It uses
  - Drivers
  - Stubs
  - Harnesses
  - Oracles

Driver

- Driver: is a software module used to invoke a module under test and, often, provide test inputs, control and monitor execution, and report test results
  - line of code that calls a method and passes that method a value
Example of driver

movePlayer(player1, 2);

• call a method movePlayer and passes player1 and moves it two spaces
• The drivers would likely to be called from main method
• A white-box test case would execute this driver line of code and check Player.getPosition() to make sure the player is now on the expected cell on the chess board

Stub

• A stub is a computer program statement substituting for the body of a software module that is or will be defined elsewhere or
• A dummy component or object used to simulate the behaviour of a real component until that component has been developed

Examples of stub

Simply:
public void movePlayer(Player aPlayer, int diceValue) {
}
or
public void movePlayer(Player aPlayer, int diceValue) {
    player=aPlayer;
    player.setPosition(1);
}

WBT with drivers and stubs

• Developing stubs allows programmers to call a method in the code being developed, even if the method does not yet have the desired behaviour
• Stubs can be “filled in” later to form the actual method

Oracle

• An oracle is a piece of code that compares the output of a driver with the a predicted output
• In JUnit these are assert methods:
  `assertEquals(predicted output, output of a driver)`

Harness

• It is the environment in which to execute the tests
• Typically are frameworks
• Example: JUnit library for eclipse
What are drivers, stubs, and oracles?

```java
public void testCenterLine() {
    Formatter f = new Formatter();
    f.setLineWidth(10);
    assertEquals("   word   ", f.center("word"));
}

public void testOddCenterLine() {
    Formatter f = new Formatter();
    f.setLineWidth(10);
    assertEquals("  hello    ", f.center("hello"));
}

public String center(String line) {
    return "a";
}
```

What are drivers, stubs, and oracles?

TESTING WITH JUNIT
Installing JUnit

- Download from [http://junit.org](http://junit.org) the jar file
- Unzip or import the jar file
- Add to classpath of your project
  - Project ➔ Properties ➔ Java Build Path
  - Libraries tab
  - Add library

Installing JUnit

Right click on the project folder in package explorer
- BuildPath ➔ add Libraries tab
- Or
- BuildPath ➔ in the library tab select add library
**TestCase in JUnit**

- Any class that contains test methods should subclass the TestCase class
- testXXX() methods are defined in this class
- When you want to check the expected and actual test results, you invoke assert() and pass a boolean expression which will indicate if the test case is passed or not

**Fixtures:**

- When a test class contains multiple testXXX() methods, you can use the setup() and tearDown() methods to initialize and release any common objects under test

---

**Writing a Test Case (1/2)**

A. Import the Junit harness:
   ```java
   import junit.framework.*;
   ```
B. Define a subclass of TestCase
C. Override the setUp() method to initialize object(s) under test (optional)
D. Override the tearDown() method to release any permanent resources you allocated in setUp() (optional)
Writing a Test Case (2/2)

D. Define one or more testXxx() methods that exercise the objects under test
E. Define a suite() method that creates a TestSuite containing all the testXxx() methods of a TestCase
F. Define a main() method that runs the TestCase

Let’s see the JUnit code

- Main classes
  - junit.framework.TestCase
  - junit.framework.TestSuite
  - junit.textui.TestRunner
  - junit.ui.TestRunner
  - junit.ui.LoadingTestRunner

Let’s see the JUnit code

- Steps:
  - Subclass TestCase
  - Write test cases, methods that start with “test”
  - setup() - initialize variables
  - teardown() - release resources, reset
  - use a TestRunner to run tests
  - Example in the shell
    ```java
    java junit.ui.LoadingTestRunner TestClass
    org.junit.runner.JUnitCore.runClasses(TestClass1.class, ...); or
    java org.junit.runner.JUnitCore TestClass [ ...other test classes ...]
    ```
  - TestRunner expects the name of a TestCase class as argument.
Let’s see the JUnit code

TestSuite

- TestCase instances can be composed into TestSuite hierarchies that automatically invoke all the testXxxx() methods defined in each TestCase instance
- A TestSuite is a composite of:
  - Other tests
  - TestCase instances (using addTest() )
  - Other TestSuite instances (using addTest() )

Organizing Tests

- Create test cases in the same or a lateral package as the code under test
  - Has all the application level classes and test cases for those components
- For each Java package, define a TestSuite class that contains all the test for verifying code in the package
**Example the class**

```java
import java.util.*;

public class BrowsableList {
    private Vector items;

    public BrowsableList() {items = new Vector();}
    public void add(Browsable element) {items.add(element);}
}
```

**Example the class**

```java
public Browsable first() throws NoSuchElementException {
    return (Browsable)items.firstElement();
}
```

```java
public boolean isEmpty() {return items.isEmpty();}
```

**Example the test class**

```java
import junit.framework.*;
import junit.textui.*;

public class BrowsableListTest extends TestCase {
    public static void main(String[] args) {
        // run the test from the main of JUnitCore in the package runner
        org.junit.runner.JUnitCore.main(BrowsableListTest.class.getName());
    }
    private BrowsableList items;
    private TestItem alpha;
    private TestItem beta;

    public void setup() {
        items = new BrowsableList();
        alpha = new TestItem("alpha");
        beta = new TestItem("beta");
    }
}
```
Example the test class

```java
public void tearDown() {}  
public BrowsableListTest(String name) {
    super(name);
}
public void testIsEmpty() {
    assert(items.isEmpty());
}
public void testAddElement() {
    items.add(alpha);
    Browsable first = items.first();
    TestItem expected = new TestItem("alpha");
    assertEquals(first, expected);
    assert(!items.isEmpty());
}
public static Test suite() {
    return new TestSuite(BrowsableListTest.class);
}
```
Simple Test Suite

```java
public void testAdd() {
    double result = fValue1 + fValue2;
    // forced failure result == 5
    assertTrue(result == 6);
}

public void testDivideByZero() {
    int zero = 0;
    int result = 8 / zero;
    result++;
}

public void testEquals() {
    assertEquals(12, 12);
    assertEquals(12L, 12L);
    assertEquals(new Long(12), new Long(12));
    assertEquals("Size", 12, 13);
    assertEquals("Capacity", 12.0, 11.99, 0.0);
}
```

```java
public static void main(String[] args) {
    junit.textui.TestRunner.run(suite());
}
```

Setting up Composite Test Suites

```java
package junit.samples;
import junit.framework.Test;
import junit.framework.TestSuite;

/**
 * TestSuite that runs all the sample tests
 */
public class AllTests {
    public static void main(String[] args) {
        junit.textui.TestRunner.run(suite());
    }

    public static Test suite() {
        TestSuite suite = new TestSuite("All JUnit Tests");
        suite.addTest(SimpleTest.suite());
        suite.addTest(new TestSuite(junit.samples.money.MoneyTest.class));
        suite.addTest(junit.tests.AllTests.suite());
        return suite;
    }
}
```

Running a test with annotation

- When a class is annotated with @RunWith or extends a class annotated with @RunWith, JUnit will invoke the class it references to run the tests in that class instead of the runner built into JUnit
The test suite class

```java
public class AllTest{
    import org.junit.runner.RunWith;
    import org.junit.runners.Suite;
    import org.junit.runners.Suite.SuiteClasses;

    @RunWith(Suite.class)
    @SuiteClasses(ABTest.class, BTest.class, CTest.class)
    public class ABCSuite {
    }
}
```

Create the test source folder

The JUnit Build Path
JUnit Testing tips

- If you find yourself debugging using `System.out.println()`, write a test case instead – if possible ☺
- When a bug is reported, write a test case to expose the bug
- Don’t deliver code that doesn’t pass all the tests
- Write test cases to ensure method, statement, condition coverage – if possible