Java Object-Oriented Concepts Unit

Lesson 2: JUnit and Drills

The Learning House

427 S 4th Street #300

Louisville KY 40202

Lesson 2: JUnit and Drills

Overview

This document is a quick review of the code organization approaches and techniques covered in Java Basics and the beginning of Java OO Concepts. The latter part of the document discusses techniques in creating classes and testing them from JUnit test classes. The last section outlines a basic approach to unit testing your code.

One Class, All Code in Main

The first technique that we covered was writing all of our code in the main method. This is the technique used in Hello, World and the first two problem sets of Programming by Doing. For example:

```
package modulusanimationsymbols;
public class ModulusAnimationSymbols {
    public static void main( String[] args ) throws Exception {
            int repeats = 5;
            int steps_per_second = 10;

            for ( int i=0; i<repeats*11 ; i++ ) {
                  if ( i%11 == 0 )
                        System.out.print(" .oOo..... \r");
                  else if ( i%11 == 1 )
                        System.out.print(" ..oOo.... \r");
                  else if ( i%11 == 2 )
                        System.out.print(" ...oOo... \r");
                  else if ( i%11 == 3 )
                        System.out.print(" ....oOo.. \r");
                  else if ( i%11 == 4 )
                        System.out.print(" .....oOo. \r");
                  else if ( i%11 == 5 )
                        System.out.print(" ......oOo \r");
                  else if ( i%11 == 6 )
                        System.out.print(" .......oO \r");
                  else if ( i%11 == 7 )
                        System.out.print(" o.......o \r");
                  else if ( i%11 == 8 )
                        System.out.print(" Oo....... \r");
                  else if ( i%11 == 9 )
                        System.out.print(" oOo..... \r");
                  else if ( i%11 == 10 )
                        System.out.print(" .oOo..... \r");

                  Thread.sleep(1000/steps_per_second);
            }
      }
}
```

One Class, Multiple Static Methods Called from Main

The second technique that we covered was creating additional methods in our class and then calling them from main. The critical concept presented with this technique was that additional methods must be declared with the **static** keyword if they are to be called from main. For example:

```
package methodexample;

public class MethodExample {

        public static void main(String[] args) {
         silly(5);
         silly(1008);
         int foo = sillyReturnPlus(5);
         System.out.println(foo);
         int myFive = get5();
         System.out.println(myFive);
         System.out.println(get5());
    }

    public static int get5() {
        return 5;
    }

    public static int sillyReturnPlus(int i) {
        i = i + 1;
        System.out.println(i);
        return i;
    }

    public static void silly(int i) {
        System.out.println("My parameter is: " + i );
    }

    public static void doit() {
        System.out.println("Hello");
    }

}
```

One Class, Non-static Methods Called From A Test Class

The third technique that we covered in Unit 01 was creating non-static methods in a class that did not contain a main method and then calling those methods from a JUnit Test Class. The key concept presented here was that we must first **instantiate** (in the Test Call) the class that contains the method we wish to call and then **invoke** or **call** that method from our test code. For example:

Class to be tested (GreatParty)

```
package logicdrills;

public class GreatParty {

    // When squirrels get together for a party, they like to have cigars. A squirrel party
    // is successful when the number of cigars is between 40 and 60, inclusive. Unless it
    // is the weekend, in which case there is no upper bound on the number of cigars. Return
    // true if the party with the given values is successful, or false otherwise.

    // greatParty(30, false) → false
    // greatParty(50, false) → true
    // greatParty(70, true) → true
    public boolean greatParty(int cigars, boolean isWeekend) {
        if (isWeekend) {
            if (cigars >= 40) {
                return true;
            } else {
                return false;
            }
        } else {
            if (cigars >= 40 && cigars <=60) {
                return true;
            } else {
                return false;
            }
        }
    }

}
```

Class Create to Test GreatParty (GreatPartyTest)

```
package logicdrills;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

public class GreatPartyTest {

    public GreatPartyTest() {
    }
    @BeforeClass
    public static void setUpClass() {
    }
    @AfterClass
    public static void tearDownClass() {
    }
    @Before
    public void setUp() {
    }
    @After
    public void tearDown() {
    }
    @Test
    public void greatPartyTest() {
        GreatParty myParty = new GreatParty();
        // greatParty(30, false) → false
        boolean result;
        result = myParty.greatParty(30, false);
        assertFalse(result);
        // greatParty(50, false) → true
        result = myParty.greatParty(50, false);
        assertTrue(result);
        // greatParty(70, true) → true
        result = myParty.greatParty(70, true);
        assertTrue(result);
    }
}
```

Process for Drills Exercises

1. Look at problem statement and solve the given problem on paper.
2. Create a new class for the problem.
3. Copy the problem statement (comment that out) and the method definition into your new class.
4. Implement the solution.
5. Create a Test Class for your new code.
6. Implement the test code using the given test cases.
7. Modify your problem solution until all test cases pass.

Key Concepts for Calling Code from JUnit

1. Create the Test Class from the NetBeans menu.
2. Make sure you specify the **package** for your Test Class (it should be in the same package as the class you are testing).
3. Use the built in JUnit ‘assert’ methods to evaluate your test results:
4. assertTrue(...)
5. assertFalse(...)
6. assertEqual(...)
7. assertNull(...)