

Refactoring: JUnit

**CPSC 501: Advanced Programming Techniques
Fall 2020**

Jonathan Hudson, Ph.D
Instructor
Department of Computer Science
University of Calgary

Tuesday, August 4, 2020



Unit Testing

Unit Testing

- A **unit test** is a technique for testing the correctness of a module of source code
 - You create separate test cases for every nontrivial method in the module
 - Unlike most other tests, is done by developers as they code
 - Is a form of “bottom-up” testing

Benefits of Unit Testing

- Benefits of unit testing:
 - Reduces the time spent on debugging
 - **Catches bugs early**
 - Eases integration
 - Bottom-up testing allows you to build a large system on a reliable “foundation” of working low-level code
 - Documents the intent of the code
 - **Encourages refactoring**
 - Tests are rerun to make sure no new bugs are introduced
 - Is a form of regression testing

Goal of Unit Testing

- The goal of unit testing is to determine if the code:
 1. Does what is intended
 2. Works correctly under all conditions
 - Including exceptional conditions like bad input, full disks, dropped network connections, etc., etc.
 3. Is dependable

Usage of Unit Testing

- Your test code is for internal use only
 - Is separate from production code and is **not shipped**
 - Production code must be “unaware” of the test code that exercises it
 - **However, you may have to refactor poorly structured code to make it testable**

Unit Testing Frameworks

- Unit testing frameworks make it easy to build and run tests
 - Open source frameworks include:
 - JUnit for Java
 - NUnit for C#
 - pytest for python

JUnit Example

JUnit Example – Largest Integer Method

- We will test the following method:
 - Note: contains some bugs

```
public class Largest {  
    public static int largest(int[] list) {  
        int i, max = Integer.MAX_VALUE;  
        for (i = 0; i < list.length - 1; i++) {  
            if (list[i] > max) {  
                max = list[i];  
            }  
        }  
        return max;  
    }  
}
```

JUnit Example – JUnit Test

- Create a test class with an initial test:

```
import org.junit.Test;
import static org.junit.Assert.*;
public class LargestTest {

    @Test
    public void testLargest() {
        int[] list = {8,9,7};
        int expectedResult = 9;
        int result = Largest.largest(list);
        assertEquals(expectedResult, result);
    }
}
```

JUnit Example - Details

- Your test class can be named anything
- Test methods must be annotated with **@Test**
 - Will be invoked automatically by the test runner
- The **assertEquals()** will abort if the **largest()** method does not return a **9**
 - 9 is the largest element in the list 8, 9, 7
- Save the file
- Compile using: **javac *.java**

JUnit Example - Running

- Run the test
- Use: **java org.junit.runner.JUnitCore LargestTest**
 - The classpath must be set correctly for this to work
 - Is a textual UI
 - Most IDEs can run tests within their GUI

JUnit Example – Failing Test

```
public class Largest {
    public static int largest(int[] list) {
        int i, max = Integer.MAX_VALUE;
        for (i = 0; i < list.length - 1; i++) {
            if (list[i] > max) {
                max = list[i];
            }
        }
        return max;
    }
}
```

JUnit Example – Multiple Asserts

- Create a new test testOrder():

```
@Test
public void testOrder() {
    assertEquals(9, Largest.largest(new int[]{8, 9, 7}));
    assertEquals(9, Largest.largest(new int[]{9, 8, 7}));
    assertEquals(9, Largest.largest(new int[]{7, 8, 9}));
}
```

- Tests for the largest element in all 3 positions
- Recompile and retest

JUnit Example – Failing Again

```
public class Largest {  
    public static int largest(int[] list) {  
        int i, max = 0;  
        for (i = 0; i < list.length - 1; i++) {  
            if (list[i] > max) {  
                max = list[i];  
            }  
        }  
        return max;  
    }  
}
```

JUnit Example – Fix Bug

- We find another error:
- Is an “off by one” bug:
 - Change loop for correct termination
- Recompile and retest
 - Should report: OK (2 tests)

JUnit Example – More Tests

- Add methods to test for duplicates and a list of size one:

```
@Test
public void testDuplicates() {
    assertEquals(9, Largest.largest(new int[]{9, 7, 9, 8}));
}
```

```
@Test
public void testOne() {
    assertEquals(1, Largest.largest(new int[]{1}));
}
```

- Recompile and retest
 - Should report: OK (4 tests)




JUnit Example – Negative Numbers

- Add a method to test negative numbers:

```
@Test
public void testNegative() {
    assertEquals(-7, Largest.largest(new int[]{-9, -8, -7}));
}
```

- Retesting reveals another bug:

4 tests passed, 1 test failed. (0.052 s)

  LargestTest Failed
  testNegative Failed: expected:<-7> but was:<0>

- Fix by initializing `max = Integer.MIN_VALUE;`
- Retest

JUnit Example – Exceptions?

- What should happen if the list is empty?
 - Throw an exception

```
if (list.length == 0) {  
    throw new RuntimeException("largest: Empty list");  
}
```

JUnit Example – Exceptions Expected

- Add a test for this:

```
@Test(expected = RuntimeException.class)
public void testEmpty() {
    Largest.largest(new int[]{});
}
```

JUnit Example – Null?

- What if our function should crash on null input?

```
@Test(expected = NullPointerException.class)
public void testNull() {
    Largest.largest(null);
}
```

JUnit Versions

JUnit Versions

- There are three main JUnit revisions active
- JUnit 3 (offered as choice by eclipse)
- JUnit 4 (default in eclipse? or was, might change soon if is)
- JUnit 5 (AKA Jupiter, default in Netbeans)

JUnit Versions

- There are three main JUnit revisions active
- JUnit 3 (offered as step down choice by eclipse)
 - JDK 1.2+
- JUnit 4
 - JDK 1.5+
- JUnit 5
 - JDK 1.8+
 - Has JUnit Vintage for running Junit 3/4 Tests
- Recommend using JUnit5 and an IDE environment like eclipse

JUnit 3

- There are three main JUnit revisions active
- JUnit 3 (offered as step down choice by eclipse)
 - **Had to use testXXX pattern for methods**
 - **Had to catch exception manually**
 - **Had to comment out tests to ignore them**
 - **Can't test timeouts**
 - **Can't make methods that run once before test class executes**

JUnit 4

- There are three main JUnit revisions active
- JUnit 4 (default in eclipse/netbeans)
 - Most common (JUnit 5 adds features that are nice but less of a big deal)
 - **@Test to designate tests**
 - **@BeforeClass/@AfterClass for methods to run once for test class**
 - **@Before/@After for methods to run around each test**
 - **Can test for exceptions**
 - Can @Ignore tests
 - **Can test with timeouts @Test(timeout=1000)**
 - @Category of tests
 - Can add fail messages to asserts

JUnit 5

- There are three main JUnit revisions active
- JUnit 5 (AKA JUnit Jupiter)
 - Tag name changes (same functionality)
 - **Messages moved to end of assert (makes copy-paste code trickier b/w versions)**
 - **Can @Nested tests to only run if outer passes**

JUnit Framework

JUnit Framework

- The JUnit framework does the following:
 - Sets up conditions needed for testing
 - Creates objects, allocates resources, etc.
 - Calls the method
 - Verifies the method worked as expected
 - Cleans up
 - Deallocates resources, etc.

JUnit Framework

- All test methods must be annotated with `@Test`
- Are invoked automatically by the framework

- Each method uses various **assert** helper methods
 - Aborts the test method if the assertion fails
 - Reports failures to the user

JUnit Asserts

- JUnit asserts: (JUnit4 and JUnit5 will swap message front/end of parameters)
 - **assertEquals**([String message], expected, actual)
 - message is optional
 - **assertEquals**([String message], expected, actual, **tolerance**)
 - Useful for imprecise f.p. numbers
 - **assertNull**([String message], Object object)
 - Asserts that the object is null
 - Also: `assertNotNull()`

JUnit Asserts

- JUnit asserts: (JUnit4 and JUnit5 will swap message front/end of parameters)
 - `assertSame([String message], expected, actual)`
 - Asserts that `expected` and `actual` point to the same object
 - Also: `assertNotSame()`
 - `assertTrue([String message], boolean condition)`
 - Also: `assertFalse()`
 - `fail([String message])`
 - Fails the test immediately
 - Used to mark code that should not be reached

JUnit After/Before

- Use **@Before** to mark a method used to initialize the testing environment before every test in test class
 - E.g. Allocate resources, initialize state
- Use **@After** to mark a method used to clean up after every test in test class
 - E.g. Deallocate resources
- **Are invoked before and after every test method is run**
- Incredibly useful to make objects re-used across multiple tests
- **Tests should be designed to be run independently, and in any order**
 - (JUnit doesn't follow your source code order)

JUnit AfterClass/BeforeClass

- Like @Before/@After, but once for the whole test class (instead of each function)
- Good for static setups, like database connections
- Use **@BeforeClass** to mark a method used to initialize the testing environment when test class is initialized
 - E.g. Allocate resources, initialize state
- Use **@AfterClass** to mark a method used to clean up after every test in test class is complete
 - E.g. Deallocate resources

JUnit Before/After Examples

Junit: Before and after

- **BeforeClass** – things you need for multiple tests (connections to resources, constants), shouldn't be changed by tests
- **AfterClass** – cleanup things related to BeforeClass
- **Before** – things used for multiple tests, often changed by tests
- **After** – clean up stuff related to Before

```
import org.junit.*;
import static org.junit.Assert.*;
public class LargestTest {

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }

}
```

JUnit After/Before - Example

```
private Connection dbConn;
@Before
public void setUp() {
    dbConn = new Connection(...);
    dbConn.connect();
}
@After
public void tearDown() {
    dbConn.disconnect();
}
@Test
public void testReadDB() {
    // uses dbConn
}
@Test
public void testWriteDB() {
    // uses dbConn
}
```

JUnit How Suite it Is

JUnit Suites

- By default, the test runner uses all methods of a class annotated with `@Test`
 - A suite of tests is created for you automatically
- You can combine tests from several classes into a suite using the `@RunWith` and `@SuiteClasses` annotations
 - E.g. Combine tests from `TestLargest` and `TestSmallest` classes

JUnit Suite - Example

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

@RunWith(Suite.class)
@SuiteClasses({TestLargest.class, TestSmallest.class})
public class AllTests {
}
```


JUnit Creating Tests

Testing

- Use the mnemonic Right-BICEP
 - Are the results **right**?
 - i.e. Does the code do what is intended?
 - Check **B**oundary conditions
 - Most bugs live at the “edges”
 - Examples:
 - Beginnings and ends of lists, files, etc.
 - Badly formatted data
 - Empty or missing values
 - ... (next page)

Testing

- Out of range values
 - Duplicates in lists that should have unique data
 - Ordered lists that aren't, and vice-versa
 - Things that arrive out of order
 - Etc.
-
- **Check Inverse relationships**
 - E.g. Square the result of a square root method
 - **Cross-check by other means**
 - E.g. Use a proven sorting algorithm to check the results of a new sorting method

Testing

- **Force Error conditions**
 - Incorrect input parameters
 - Problems in the environment like:
 - Running out of RAM
 - Running out of disk space
 - Dropped network connections
 - High system load
 - Etc.
 - Can be simulated using mock objects
 - i.e. objects that stand in for system resources

Testing

- **Check Performance characteristics**
 - Make sure performance doesn't degrade
 - With large inputs
 - When adding new functionality
- Always ask yourself:
 - **What else can go wrong?**

Onward to ... continuous integration & deployment.

Jonathan Hudson
jwhudson@ucalgary.ca
<https://pages.cpsc.ucalgary.ca/~hudsonj/>



UNIVERSITY OF
CALGARY