

Software Development: JUnit

**CPSC 219: Introduction to Computer Science for Multidisciplinary
Studies II
Fall 2023**

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

Wednesday, 22 September 2023

Copyright © 2023



UNIVERSITY OF
CALGARY

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 right now)

```
public class Largest {  
  
    public static int largest1(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.jupiter.api.MethodOrderer;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestMethodOrder;

import static org.junit.jupiter.api.Assertions.*;

@TestMethodOrder(MethodOrderer.MethodName.class)
class LargestTest {

    @Test
    void testLargest1Basic() {
        int[] list = {8, 9, 7};
        int expectedResult = 9;
        int result = Largest.largest1(list);
        assertEquals(expectedResult, result, message: "Largest value in list {8,9,7} should be 9");
    }
}
```

This is our function we are testing

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 **largest1()** 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

```
org.opentest4j.AssertionFailedError: Largest value in list {8,9,7} should be 9 ==>
Expected :9
Actual   :2147483647
<Click to see difference>
```

```
⊕ <4 internal lines>
⊕ at LargestTest.testLargest11Basic(LargestTest.java:15) <31 internal lines>
⊕ at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>
⊕ at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <23 internal lines>
```

```
public static int largest2(int[] list) {
    int i, max = 0;
    for (i = 0; i < list.length - 1; i++) {
        if (list[i] > max) {
            max = list[i];
        }
    }
    return max;
}
```

Let's try max=0 instead

JUnit Example – Multiple Asserts

- Create a new test testOrder():

```
@Test
void testLargest22Order() {
    assertEquals( expected: 9, Largest.largest2(new int[]{8, 9, 7}), message: "Largest value in list {8,9,7} should be 9");
    assertEquals( expected: 9, Largest.largest2(new int[]{9, 8, 7}), message: "Largest value in list {9,8,7} should be 9");
    assertEquals( expected: 9, Largest.largest2(new int[]{7, 8, 9}), message: "Largest value in list {7,8,9} should be 9");
}
```

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

JUnit Example – Failing Again

```
org.opentest4j.AssertionFailedError: Largest value in list {7,8,9} should be 9 ==>
Expected :9
Actual   :8
```

```
public static int largest3(int[] list) {
    int i, max = 0;
    for (i = 0; i < list.length; i++) {
        if (list[i] > max) {
            max = list[i];
        }
    }
    return max;
}
```

We had off by one error

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
void testLargest33Duplicates() {
    assertEquals( expected: 9, Largest.largest3(new int[]{9, 7, 8, 9}), message: "Largest value in list {9,7,8,9} should be 9");
}

@Test
void testLargest340ne() {
    assertEquals( expected: 9, Largest.largest3(new int[]{9}), message: "Largest value in list {9} should be 9");
}
```

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

JUnit Example – Negative Numbers

- Add a method to test negative numbers:

```
@Test
void testLargest35Negative() {
    assertEquals( expected: -7, Largest.largest3(new int[]{-9, -8, -7}), message: "Largest value in list {-7,-8,-9} should be -7");
}
```

- Retesting reveals another bug:

```
org.opentest4j.AssertionFailedError: Largest value in list {-7,-8,-9} should be -7 ==>
Expected :-7
Actual   :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
void testLargest46Empty() {
    RuntimeException e = assertThrows(RuntimeException.class, () -> {
        Largest.largest4(new int[]{});
    });
    assertEquals("expected: 'largest: empty list', e.getMessage()", "message: 'Expect RuntimeException for empty list usage.'");
}
```

JUnit Example – Null?

- What if our function should crash on null input?

```
if (list == null) {  
    throw new NullPointerException("largest: null list");  
}
```

```
@Test  
void testLargest47Null() {  
    NullPointerException e = assertThrows(NullPointerException.class, () -> {  
        Largest.largest4(null);  
    });  
    assertEquals("largest: null list", e.getMessage(), "Expect NullPointerException for null list usage.");  
}
```

Result

- Final Function

```
public static int largest5(int[] list) {
    if (list == null) {
        throw new NullPointerException("largest: null list");
    }
    if (list.length == 0) {
        throw new RuntimeException("largest: empty list");
    }
    int i, max = Integer.MIN_VALUE;
    for (i = 0; i < list.length; i++) {
        if (list[i] > max) {
            max = list[i];
        }
    }
    return max;
}
```

JUnit Versions

JUnit Versions

- There are three main JUnit revisions active
- JUnit 3 (old)
- JUnit 4 (common to find examples, not recommended)
- JUnit 5 (AKA Jupiter, default in most IDEs)

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+ (Java 8 and higher)
 - Has JUnit Vintage for running JUnit 3/4 Tests
- Recommend using JUnit5 and an IDE environment like eclipse

JUnit 4

- JUnit 4 (included in eclipse/netbeans)
 - Was 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

- JUnit 5 (AKA JUnit Jupiter)
 - Tag name changes (same functionality)
 - **Messages moved to end of assert (makes copy-paste code trickier b/w versions)**
 - **@BeforeAll/@BeforeEach/@AfterAll/@AfterEach (same function, clearer name)**
 - **Can create test order**
 - **Can @Nested tests to only run if outer passes**
 - **AssertThrows better than @Test (expected==Exception.class)**

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)
- <https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/Assertions.html>
 - **assertEquals**(expected, actual, [String message])
 - message is optional
 - **assertEquals**(expected, actual, **tolerance**, [String message])
 - Useful for imprecise f.p. numbers
 - **assertNull**(Object object, [String message])
 - Asserts that the object is null
 - Also: **assertNotNull**()

JUnit Asserts

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

JUnit Before/After Examples

JUnit AfterAll/BeforeAll

- Use **@BeforeAll** to mark a method used to initialize the testing environment before every test in test class
 - E.g. Allocate resources, initialize state
- Use **@AfterAll** 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 DOES NOT follow your source code order)**

JUnit AfterEach/BeforeEach

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

Junit: Before and after

- **BeforeAll** – things you need for multiple tests (connections to resources, constants), shouldn't be changed by tests
- **AfterAll** – cleanup things related to BeforeClass
- Issue here?

```
static int[] list1;

@BeforeAll
public static void setup_class(){
    list1 = new int[]{8,9,7};
}

@AfterAll
public static void teardown_class(){
    list1 = null;
}
```

Junit: Before and after

- **BeforeAll** – things you need for multiple tests (connections to resources, constants), shouldn't be changed by tests
- **AfterAll** – cleanup things related to BeforeClass

```
static int[] list1;

@BeforeAll
public static void setup_class(){
    list1 = new int[]{8,9,7};
}

@AfterAll
public static void teardown_class(){
    list1 = null;
}
```

```
@Test
void testLargest1() {
    int expectedResult = 9;
    int result = Largest.largest5(list1);
    assertEquals(expectedResult, result, message: "Largest value in
list1[0] = 100;
}

@Test
void testLargest2() {
    int expectedResult = 9;
    int result = Largest.largest5(list1);
    assertEquals(expectedResult, result, message: "Largest value in
list1[0] = 100;
}
```

Junit: Before and after

- **BeforeAll** – things you need for multiple tests (connections to resources, constants), shouldn't be changed by tests
- **AfterAll** – cleanup things related to BeforeClass
- Best used when you need some sort of infrastructure through-out the whole test, like a connection

```
static DBConn = null

@BeforeAll
public static void setup_class(){
    DBConn = new DBConn(...);
}

@AfterAll
public static void teardown_class(){
    DBConn.disconnect();
    DBConn = null;
}
```

Junit: Before and after

- **BeforeEach** – things used for multiple tests, often changed by tests
- **AfterEach** – clean up stuff related to Before
- Proper usage for setting up an object, especially if you want to re-use it for multiple tests
- Great if you have a large amount of related classes to setup before a test can begin operating
- Ex. A lecture object connected with a list of student

```
int[] list1;

@BeforeEach
public void setup_test() {
    list1 = new int[]{8, 9, 7};
}

@AfterEach
public void teardown_test() {
    list1 = null;
}
```

Junit: Before and after

- **BeforeEach** – things used for multiple tests, often changed by tests
- **AfterEach** – clean up stuff related to Before

```
int[] list1;

@BeforeEach
public void setup_test() {
    list1 = new int[]{8, 9, 7};
}

@AfterEach
public void teardown_test() {
    list1 = null;
}
```

```
@Test
void testLargest1() {
    int expectedResult = 9;
    int result = Largest.largest5(list1);
    assertEquals(expectedResult, result, message: "Largest
list1[0] = 100;
}

@Test
void testLargest2() {
    int expectedResult = 9;
    int result = Largest.largest5(list1);
    assertEquals(expectedResult, result, message: "Largest
list1[0] = 100;
}
```


Onward to ... Command Line, Files, and Exceptions

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



UNIVERSITY OF
CALGARY