Software Development (CS2500)

Lecture 17: Fixing Bugs with `JUnit` Tests

M. R. C. van Dongen

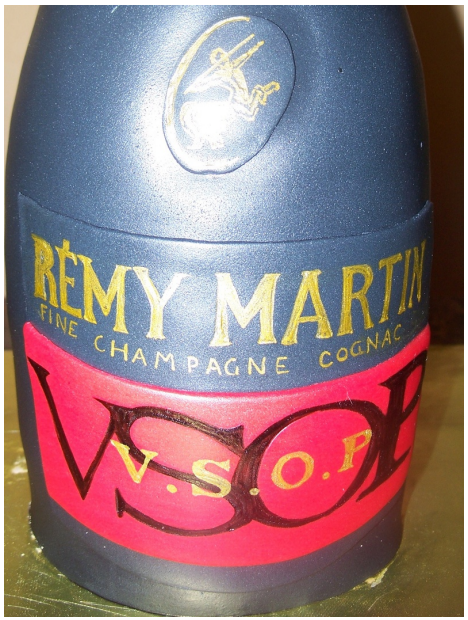
OCtober 30, 2013

Bonus Assignment

- 30×1 mark;
- 18×2 marks;
- 1×3 marks.

Who Won the Prize?

Here's a Clue



And Another One

And Another One (Sort Of)

M. R. C. van Dongen

Congratulations Martin Bullman

Objectives

- This lecture is about `JUnit` testing.
- Not examinable for written examination.
- Lecture is mainly based on:
  - [Furguson Smart 2008, Chapter 10],
  - [Hunt, and Thomas 2007], and
  - [Sommerville 2007, Chapter 23].
- More information on `http://en.wikipedia.org/wiki/Unit_test`.

Assertions

Java

```
public class TestAssert {
    public static void main( String[] args ) {
        int value = 1;
        assert( value == 1 );
        value = 2; // Simulate error.
        assert( value == 1 );
        System.out.println( "value = " + value );
    }
}
```

Assertions

Unix Session

```
$
```

Assertions

Unix Session

```
$ java -ea TestAssert
```

Assertions

Unix Session

```
$ java -ea TestAssert
Exception in thread "main" java.lang.AssertionError
        at TestAssert.main(TestAssert.java:6)
$
```

Assertions

Unix Session

```
$ java -ea TestAssert
Exception in thread "main" java.lang.AssertionError
        at TestAssert.main(TestAssert.java:6)
$
```

Assertions

Unix Session

```
$ java -ea TestAssert
Exception in thread "main" java.lang.AssertionError
        at TestAssert.main(TestAssert.java:6)
$
```

Assertions

Unix Session

```
$ java -ea TestAssert
Exception in thread "main" java.lang.AssertionError
        at TestAssert.main(TestAssert.java:6)
$
```

Assertions

Unix Session

```
$
```

Assertions

Unix Session

```
$ java TestAssert
```

Assertions

Unix Session

```
$ java TestAssert
value = 2
$
```

Advantages and Disadvantages of Assertions

- Easy mechanism to integrate tests and code.
- Easy to turn tests on and off.
- Form of documentation.
- Easy support for *integrated* testing.
- Does not facilitate method testing in *isolation*.

What is Unit Testing?

- *Unit* (component): smallest testable part of application.
- A *unit test* verifies the individual units work properly.
- Unit testing is also known as *component testing*.
- Many companies have specialised *test engineers*.
  - They test software developed by software developers.
- Unit tests are written by the software developers themselves.
- Goal of unit testing is to show the individual units are correct.
- Unit testing is done right from the early stage of development.

Why Unit Testing

Defect testing: Unit testing is a *defect testing process*:
- Goal is to expose faults in the components.

Provides confidence: Eliminates uncertainty about the units.

Automates testing: Unit tests can be automated.

Regression testing: Supports repeating of previous tests.

Robustness to changes: Tests remain valid after changing code.

Simplifies integration: Bugs are eliminated at an early stage.

Extra effort saves *much* work later.

Documentation: Provides "documentation" of unit API.

Contract: The test is a contract that the unit must satisfy.

Drives design: Writing test *first* drives code design.

Separates testing: Testing can be done in *isolation*.

What to Test

RIGHT-BICEP: your key to successful unit testing

RIGHT results:	Are the results right?
Boundary conditions:	Are the boundary conditions correct?
Inverse relationships:	Can you check inverse relationships?
Cross-checking:	Can you cross-checks results?
Error conditions:	Can you force error-conditions to happen?
Performance:	Is the performance satisfactory?

Are the Results **RIGHT**?

- These are simple tests to verify if the results are correct.
- Given the input: what would be the right result.
  - Given numbers 2, 4, and 1: which is the larger?
  - Given numbers 45, and 2345: which is the larger?
  - Given number 1, what is the absolute value?
  - Given number -4, what is the absolute value?

Are the **B**oundary Conditions **CORRECT**?

Range:

- Wrong file extension.
- Bogus input: surname is `cw4vr@:.`
- Missing values in array declarations.
- Unreasonable input: a `10000` years old person.
- Ordered lists aren't sorted.
- Negative/positive numbers.
- Sequencing errors.

Boundary conditions:

- Fence-post problems.
- Other off-by one errors.
- Empty lists.
- Division by zero.

Are the Boundary Conditions CORRECT?

Conformance: Does the value conform to an expected format?

Ordering: Are the values correctly ordered?

- Should result be independent of ordering?

Range: Is the value's range OK?

Reference: Are there references to objects in other classes?

- If yes, are the conditions for referencing them right?

Existence: Is the value non-zero, is it present in a set, ...?

Is the string non-empty?

Is the reference non-null?

Cardinality: Is the number of things correct?

Are there off-by-one errors?

Time: Are things happening at the right time?

Are they happening in the right order?

Can you check Inverse Relationships?

- Some results can be cross-checked using *inverse relationships*.
  - $x = \sqrt{x}^2$, provided x is non-negative.
  - $x = -(-x)$.
  - $x = (2x)/2$.
- Excellent for *many* tests with pseudo-random data in a `for`-loop.

Other Relationships

- There may be recurrence relations:
  - $f_{n+2} = f_{n+1} + f_n$,
  - $m! = (m-1)! \times m$,
  - ....

Java

```
for (int n = 0; n <= MAX_N; n++) {
    final int fib2 = fibonacci( n + 2 );
    final int fib1 = fibonacci( n + 1 );
    final int fib0 = fibonacci( n + 0 );
    assertTrue( fib2 == fib1 + fib0 );
}
```

- Solutions may be known given certain boundary conditions.
  - You may know the answer if the input is even.

Java

```
for (int n = -MAX_N; n <= MAX_N; n++) {
    final int newAnswer  = myNewMethod( 2 * n );
    final int evenAnswer = myAnswerForEvenNumbers( 2 * n );
    assertTrue( newAnswer == evenAnswer );
}
```

  - You may know the answer if the input is a power of 2.
  - ...

Can you Cross-check Results using Other Means?

- Do you know a less efficient way to compute the result?
  - For example, you have to compute $f(n) = \sum_{i=0}^{n} i$.
  - You implement a clever way to compute it:
    - $f(n) = n(n+1)/2$.
  - You can check the result using a simple `for`-loop.
- Can you check against a previous release?
- Is there a database with test-cases?

First JUnit Tests

Java

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

First JUnit Tests

All JUnit Test Methods are Instance Methods

Java

```
import static org.junit.Assert.*;
import org.junit.Test;

public class TestLargest {
    @Test
    public void orderTest( ) {
        final int[] ints = new int[] {7,6,8,9};
        assertEquals( 9, Largest.largest( ints ) );
    }

    @Test
    public void success( ) {
        // Simulate successful test.
    }
}
```

Running the Test

Unix Session

```
$
```

Running the Test

Unix Session

```
$ CLASSPATH=${CLASSPATH}:/usr/share/java/junit4.jar:.
```

Running the Test

Unix Session

```
$ CLASSPATH=${CLASSPATH}:/usr/share/java/junit4.jar:.
$
```

Running the Test

Unix Session

```
$ CLASSPATH=${CLASSPATH}:/usr/share/java/junit4.jar:.
$ export CLASSPATH
```

Unix Session

```
$
```

Running the Test

Unix Session

```
$ CLASSPATH=${CLASSPATH}:/usr/share/java/junit4.jar:.
$ export CLASSPATH
```

Unix Session

```
$ javac Largest.java TestLargest.java
```

Running the Test

Unix Session

```
$ CLASSPATH=${CLASSPATH}:/usr/share/java/junit4.jar:.
$ export CLASSPATH
```

Unix Session

```
$ javac Largest.java TestLargest.java
$
```

Running the Test

Unix Session

```
$ CLASSPATH=${CLASSPATH}:/usr/share/java/junit4.jar:.
$ export CLASSPATH
```

Unix Session

```
$ javac Largest.java TestLargest.java
$ java org.junit.runner.JUnitCore TestLargest
```

Running the Test

Unix Session

```
$ CLASSPATH=${CLASSPATH}:/usr/share/java/junit4.jar:.
$ export CLASSPATH
```

Unix Session

```
$ javac Largest.java TestLargest.java
$ java org.junit.runner.JUnitCore TestLargest
...
Time: 0.007
JUnit version 4.3.1
.E
Time: 0.009
There was 1 failure:
1) orderTest(TestLargest)
java.lang.AssertionError: expected:<9> but was:<2147483647>
...
FAILURES!!!
Tests run: 2,  Failures: 1
$
```

Fixing the Bug

Java

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

Fixing the Bug

Java

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

Fixing the Bug

Java

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

Fixing the Bug (Continued)

Unix Session

```
$
```

Fixing the Bug (Continued)

Unix Session

```
$ javac Largest.java TestLargest.java
```

Fixing the Bug (Continued)

Unix Session

```
$ javac Largest.java TestLargest.java
$
```

Fixing the Bug (Continued)

Unix Session

```
$ javac Largest.java TestLargest.java
$ java org.junit.runner.JUnitCore TestLargest
```

Fixing the Bug (Continued)

Unix Session

```
$ javac Largest.java TestLargest.java
$ java org.junit.runner.JUnitCore TestLargest
...
Time: 0.007
JUnit version 4.3.1
.E.
Time: 0.012
There was 1 failure:
1) orderTest(TestLargest)
java.lang.AssertionError: expected:<9> but was:<8>
...
FAILURES!!!
Tests run: 2,  Failures: 1
$
```

Fixing the Bug (The Never-Ending Saga)

`java.lang.AssertionError: expected:<9> but was:<8>`

Java

```
import static org.junit.Assert.*;
import org.junit.Test;

public class TestLargest {
    @Test
    public void orderTest( ) {
        int[] ints = new int[] {7,6,8,9};
        assertEquals( 9, Largest.largest( ints ) );
    }

    @Test
    public void success( ) {
        // Simulate successful test.
    }
}
```

Fixing the Bug (The Never-Ending Saga)

`java.lang.AssertionError:` expected:<9> but was:<8>

Java

```
import static org.junit.Assert.*;
import org.junit.Test;

public class TestLargest {
    @Test
    public void orderTest( ) {
        int[] ints = new int[] {7,6,8,9};
        assertEquals( 9, Largest.largest( ints ) );
    }

    @Test
    public void success( ) {
        // Simulate successful test.
    }
}
```

Fixing the Bug (The Never-Ending Saga)

`java.lang.AssertionError:` expected:<9> but was:<8>

Java

```
import static org.junit.Assert.*;
import org.junit.Test;

public class TestLargest {
    @Test
    public void orderTest( ) {
        int[] ints = new int[] {7,6,8,9};
        assertEquals( 9, Largest.largest( ints ) );
    }

    @Test
    public void success( ) {
        // Simulate successful test.
    }
}
```

Fixing the Bug (The Never-Ending Saga)

Java

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

Fixing the Bug (The Never-Ending Saga)

Java

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

Fixing the Bug (The Never-Ending Saga)

Java

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

Fixing the Bug (The Never-Ending Saga)

Unix Session

```
$
```

Fixing the Bug (The Never-Ending Saga)

Unix Session

```
$ javac Largest.java TestLargest.java
```

Fixing the Bug (The Never-Ending Saga)

Unix Session

```
$ javac Largest.java TestLargest.java
$
```

Fixing the Bug (The Never-Ending Saga)

Unix Session

```
$ javac Largest.java TestLargest.java
$ java org.junit.runner.JUnitCore TestLargest
```

Fixing the Bug (The Never-Ending Saga)

Unix Session

```
$ javac Largest.java TestLargest.java
$ java org.junit.runner.JUnitCore TestLargest
...
JUnit version 4.3.1
..
Time: 0.011

OK (2 tests)
$
```

JUnit Assertions

- `assertEquals( expected, actual )`
  - Compares primitive and object types.
- `assertEquals( expected, actual, tolerance )`
  - For comparing floating point numbers.
- `assertNull( object )`
  - Asserts that `object == null`.
- `assertNotNull( object )`
  - Asserts that `object != null`.
- `assertSame( expected, actual )`
  - Asserts that `object` and `actual` are aliases.
- `assertNotSame( expected, actual )`
  - Asserts `object` and `actual` aren't aliases.
- `assertTrue( condition )`
  - Asserts `condition` is `true`.
- `assertFalse( condition )`
  - Asserts `condition` is `false`.
- `fail( )`
  - Fails immediately.

JUnit Assertions: First Optional Argument

Java

```
assertEquals( "Should be 3 1/3", 3.33, 10.0/3.0, 0.01 );
```

Per-Test Set Up and Tear Down

- Many tests require per-test set up and tear down.
- For example:
  - Test may assume open database connection.
- For test to work:
  - The database connection is opened before the test.
  - The database connection is closed after the test.

@Before and @After

Java

```
import org.junit.Before;
import org.junit.After;
import org.junit.Test;

public class TestBeforeAfter {
    private int value;

    @Before
    public void initialise( ) {
        System.out.println( "Initialising" );
        value = 1;
    }

    @After
    public void tearDown( ) {
        System.out.println( "Tearing down. Value is: " + value );
    }

    @Test
    public void test1( ) {
        value += 3;
    }

    @Test
    public void test2( ) {
        value += 4;
    }
```

Running the Test

Unix Session

```
$
```

Running the Test

Unix Session

```
$ javac TestBeforeAfter.java
$
```

Running the Test

Unix Session

```
$ javac TestBeforeAfter.java
$ java org.junit.runner.JUnitCore  TestBeforeAfter
JUnit version 4.3.1
.Initialising
Tearing down. Value is: 4
.Initialising
Tearing down. Value is: 5

Time: 0.04

OK (2 tests)
$
```

Per-Class Set Up and Tear Down

- Per-class setting up and tearing down is also supported.
- Useful for expensive resources.

@Before and @After

Java

```
import org.junit.BeforeClass;
import org.junit.AfterClass;
import org.junit.Test;

public class TestBeforeAfterClass {
    private static int value;

    @BeforeClass
    public static void initialiseClass( ) { value = 1; }

    @Test public void test1( ) { value += 3; }
    @Test public void test2( ) { value += 4; }

    @AfterClass
    public static void tearClassDown( ) {
        System.out.println( "Tearing down. Value is: " + value );
    }
}
```

Running the Test

Unix Session

```
$
```

Running the Test

Unix Session

```
$ javac TestBeforeAfterClass.java
$
```

Running the Test

Unix Session

```
$ javac TestBeforeAfterClass.java
$ java org.junit.runner.JUnitCore  TestBeforeAfterClass
```

Running the Test

Unix Session

```
$ javac TestBeforeAfterClass.java
$ java org.junit.runner.JUnitCore  TestBeforeAfterClass

JUnit version 4.3.1
..Tearing down. Value is: 8

Time: 0.012

OK (2 tests)

$
```

Tests with Timeout Parameters

- Many programs critically depend on time.
- For example, user interfaces should be responsive.
- Other computations cannot take forever.
- Testing with a maximum computation makes sense.
  - Catches errors due to slow response time;
  - May catch infinite loops;
  - ....

Example

Java

```
import org.junit.Test;

public class TestTimeout {
    @Test(timeout=10)
    public void failure( ) {
        for (int index = 0; ; ) ;
    }

    @Test(timeout=1)
    public void success( ) {
    }
}
```

Example (Continued)

Unix Session

```
$
```

Example (Continued)

Unix Session

```
$ javac TestTimeout.java
$
```

Example (Continued)

Unix Session

```
$ javac TestTimeout.java
$ java org.junit.runner.JUnitCore  TestTimeout
```

Example (Continued)

Unix Session

```
$ javac TestTimeout.java
$ java org.junit.runner.JUnitCore  TestTimeout
JUnit version 4.3.1
.E.
Time: 0.04
There was 1 failure:
1) failure(TestTimeout)
java.lang.Exception: test timed out after 10 milliseconds
:

FAILURES!!!
Tests run: 2,  Failures: 1
$
```

Acknowledgements

- This lecture is based on [Furguson Smart 2008, Chapter 10], [Hunt, and Thomas 2007], and [Sommerville 2007, Chapter 23].
- Further information about `JUnit` testing may be found at `http://en.wikipedia.org/wiki/Unit_test`.

Bibliography

- Furguson Smart, John [2008]. *Java Power Tools*. O'Reilly. ISBN: 978-0-596-52793-8.
- Hunt, Andrew, and David Thomas [2007]. *Pragmatic Unit Testing In Java with JUnit*. The Pragmatic Programmers. ISBN: 978-09745140-1-7.
- Sommerville, Ian [2007]. *Software Engineering*. Eight Edition. Addison Wesley. ISBN: 978-0-321-31379-9.

For Friday

- Study the lecture notes and study Chapter 6.

About this Document

- This document was created with `pdflatex`.
- The LaTeX document class is `beamer`.