Unit Testing in Java

by Kip Irvine

COP 4814

Revision date: Nov 29, 2009

(c) 2008-2009 Kip Irvine, all rights reserved. You may modify and copy this slide show for your personal use as long as this copyright statement is included.
Overview

• What is a Unit Test?
• Why do Unit Testing?
• Goals
• Test-Driven Approach
• Common Excuses
• Example 1: Largest
• Running JUnit Tests
• Static methods & asserts
• Example 2: Stack
• Reference Sources
Goals of Software Testing

1. Does the application code accomplish the basic goals of the application?
2. Does the code work with a variety of inputs?
3. Will it stand up under extreme types of input?
4. Does the code document my intent?
What is a Unit Test?

• A method that tests results produced by another method
  • proves (or disproves) that individual methods work as expected

• Called a *structural test*, or *white box test* because it requires knowledge of the source code details
Two Classes

• Class being tested
  • called the *class under test*, or *target class*
  • contains methods that do the actual work

• Test class
  • also called a *Test Fixture*
  • contains methods that test the Application class methods
Here's What Unit Testing Can Do

- Reduces the time you spend debugging, particularly at the end of a project
- Improves reliability of application code
- Minimizes bug fixes that affect existing code
  (run regression tests after fixing a bug)
Test-Driven Approach

1. Write several unit tests before writing the target application code
2. The test should document the application code's intent
3. Run the tests and watch them fail
4. Write application code that permits the tests to pass
I'm already busy writing code! Why should I write even more?

REAL Programmers code in BINARY.
Common Excuse for not testing:

Writing tests takes up too much time

Rebuttal:

• Unit Testing saves time—why?
  • It's easier to write tests while you still remember the code
• Waiting until the end means dealing with greater complexity
  • reworking defective code under pressure
  • tracking down bugs to their source in a large program
  • you might create new bugs when fixing old ones
Task completion (x) vs Time (y)

- Code, then test
- TDD
Java Unit Tests
Example 1: getLargest

- ArrayUtils class contains a method named `getLargest` that returns the largest element in an array of integers
- Create the `TestArrayUtils` class to test methods in the ArrayUtils class
  - `testGetLargest_1` calls `ArrayUtils.getLargest`
  - `testGetLargest_2` calls `ArrayUtils.getLargest`
  - etc.

See: ArrayUtils.java, TestArrayUtils.java
JUnit Framework

- Download from http://JUnit.org
- Created by Kent Beck and Erich Gamma
- Current version: 4.x

View the Course topics page
Running Tests

- In Eclipse, select the test class and execute it as a JUnit application.

- Alternatives:
  - Graphical TestRunner application:
    
    ```java
    java junit.swingui.TestRunner
    ```
  
  - Text-based TestRunner application:
    
    ```java
    java junit.textui.TestRunner classname
    ```
JUnit Static Methods

- Compare primitive and object types:
  ```java
  assertEquals( [String message], expected, actual )
  ```

- Compare primitive and object arrays:
  ```java
  assertArrayEquals( [String message], expected[], actual[] )
  ```

- For floating-point values, use:
  ```java
  assertEquals( [String message], expected, actual, tolerance )
  ```

- Set tolerance to a small value, such as 0.001
assertNull( [String message], object )
assertNotNull( [String message], object )

assertSame( [String message], expected, actual )
// Returns true if expected and actual refer to the same // object.

assertNotSame( ... )

assertTrue( [String message], boolean condition)
assertFalse( ... )

fail( [String message] );
// Forces the current test to fail when this statement // executes.
Controlling the Order of Execution

@Before
method executes before each test

@After
method executes after each test

@Ignore( String message )
do not test this method
Example 2: Stack Class

• Holds strings
• Methods:
  • push, pop, empty full, clear, stackSize, count
Some Tests We Can Run

1. Does count() return the correct values before and after calling push()?
2. Does count() return the correct values before and after calling pop()?
3. Does full() return true when count() equals stackSize()?
4. Is an exception thrown when calling push() on an empty stack?
5. Is an exception thrown when calling pop() on an empty stack?
6. Is an exception thrown when calling push() on a full stack?
7. Does the full() method return false after one or more values have been pushed on the stack?
More Tests

8. After a value is pushed on the stack, is the same value returned by pop()?

9. After multiple values are pushed, is the first value returned by pop() equal to the last value pushed?

10. Does empty() return true after all values have been popped from the stack?
Code Coverage

• Determine if the unit tests have covered a large percentage of the code under test.

• Criteria:
  • has every method been called?
  • has every statement been executed?
  • has each branch been executed?
  • has each boolean expression been evaluated to both true and false?
  • has each state in a finite-state machine been reached and explored?
  • have all common values for parameters been considered?
Adding EclEmma to Eclipse 4

- From the Help menu, select *Install New Software*
- Click the *Add* button:

![Image of EclEmma installation window]
Adding EclEmma

- Enter the following into the dialog and click OK.

![Add Repository dialog](image)
Click the Checkbox and click the Next button
More Steps...

**Install Details**
Review the items to be installed.

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>EclEmma Java Code Coverage</td>
<td>2.2.1.20130609</td>
<td>com.mountainminds.eclEmma.fea...</td>
</tr>
</tbody>
</table>

Size: 1,204 KB

**Review Licenses**
Licenses must be reviewed and accepted before the software can be installed.

License text (for EclEmma Java Code Coverage 2.2.1.201306092145):

- Eclipse Public License - v 1.0
- THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC

- I accept the terms of the license agreement
- I do not accept the terms of the license agreement

Finish
Cancel
Finishing the Installation

- You will be prompted to restart Eclipse.
- When Eclipse restarts, go to next slide
Showing the Coverage View

- Select *Show View* from the Window menu
- Select *Other*...
- Select *Java*
- Select *Coverage*, click OK.
- Run your unit tests by clicking the new green and red launch button just left of the debug button in the toolbar.
/*
 * public void clear()
 * {
 *    nextIndex = 0;
 * }
 */

/**
 * Pops and returns the item at the top of the stack.
 */
public String pop()
{
    return stack[--nextIndex];
}

/**
 * Pushes a new item onto the stack.
 */
public void push(String aString)
{
    stack[nextIndex++] = aString;
}

/**
 * Returns True if the stack is empty.
 */
public boolean empty()
{
    return nextIndex == 0;
}
## Coverage View Details

### UnitTestExample (Nov 6, 2013 11:08:45 PM)

<table>
<thead>
<tr>
<th>Element</th>
<th>Coverage</th>
<th>Covered Instructions</th>
<th>Missed Instructions</th>
<th>Total Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnitTestExample</td>
<td>62.4 %</td>
<td>433</td>
<td>261</td>
<td>694</td>
</tr>
<tr>
<td>src</td>
<td>62.4 %</td>
<td>433</td>
<td>261</td>
<td>694</td>
</tr>
<tr>
<td>ArrayTest</td>
<td>39.2 %</td>
<td>130</td>
<td>202</td>
<td>332</td>
</tr>
<tr>
<td>StackTest</td>
<td>83.7 %</td>
<td>303</td>
<td>59</td>
<td>362</td>
</tr>
<tr>
<td>StackTester.java</td>
<td>80.4 %</td>
<td>242</td>
<td>59</td>
<td>301</td>
</tr>
<tr>
<td>StackTester</td>
<td>80.4 %</td>
<td>242</td>
<td>59</td>
<td>301</td>
</tr>
<tr>
<td>TestCheckPushPerformance()</td>
<td>0.0 %</td>
<td>0</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>TestPushWhenStackIsFull()</td>
<td>0.0 %</td>
<td>0</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>TestPopOnEmptyStack()</td>
<td>0.0 %</td>
<td>0</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>TestCountAndStackSize()</td>
<td>87.1 %</td>
<td>27</td>
<td>4</td>
<td>31</td>
</tr>
<tr>
<td>setUp()</td>
<td>100.0 %</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>tearDown()</td>
<td>100.0 %</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>TestClearAndCount()</td>
<td>100.0 %</td>
<td>25</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>TestMatchMultiple_pushes_and_pop()</td>
<td>100.0 %</td>
<td>40</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>TestMatchPushAndPop()</td>
<td>100.0 %</td>
<td>29</td>
<td>0</td>
<td>29</td>
</tr>
<tr>
<td>TestPushAndCount()</td>
<td>100.0 %</td>
<td>31</td>
<td>0</td>
<td>31</td>
</tr>
<tr>
<td>TestPushAndFull()</td>
<td>100.0 %</td>
<td>20</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>TestPushAndNotEmpty()</td>
<td>100.0 %</td>
<td>41</td>
<td>0</td>
<td>41</td>
</tr>
<tr>
<td>TestStackSize()</td>
<td>100.0 %</td>
<td>18</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>Stack.java</td>
<td>100.0 %</td>
<td>61</td>
<td>0</td>
<td>61</td>
</tr>
<tr>
<td>Stack</td>
<td>100.0 %</td>
<td>61</td>
<td>0</td>
<td>61</td>
</tr>
<tr>
<td>Stack(int)</td>
<td>100.0 %</td>
<td>10</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>clear()</td>
<td>100.0 %</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>count()</td>
<td>100.0 %</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>empty()</td>
<td>100.0 %</td>
<td>7</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>full()</td>
<td>100.0 %</td>
<td>10</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>pop()</td>
<td>100.0 %</td>
<td>11</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>push(String)</td>
<td>100.0 %</td>
<td>12</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>stackSize()</td>
<td>100.0 %</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>
Reference Sources

• JUnit organization: http://junit.org/
• JUnit Cookbook (Kent Beck, Erich Gamma, )
• Get Acquainted with JUnit 4
• JUnit FAQ (SourceForge article)
• Using JUnit with Eclipse
• Next-Generation Java Testing (Cedric Beust)
• EclEmma Code Coverage tool
The End