Unit Testing with JUnit: A Very Brief Introduction

Originals of slides and source code for examples: http://courses.coreservlets.com/Course-Materials/java.html

Also see Java 8 tutorial: http://www.coreservlets.com/java-8-tutorial/ and many other Java EE tutorials: http://www.coreservlets.com/

Customized Java training courses (onsite or at public venues): http://courses.coreservlets.com/java-training.html

Topics in This Section

- **Static imports**
- **JUnit overview**
- **Modern style**
  - assertThat(value, matcher(...))
  - is, equalTo, nullValue, hasItem, not, anyOf, allOf, etc
- **Traditional style**
  - assertEquals, assertTrue, assertFalse

Static Imports

Big Idea

- **Motivation**
  - Shortens code by letting you refer to static methods without the class name
- **Syntax**
  - import static package.Class.method;
  - import static package.Class.*;
- **Example**

```
import static java.lang.Math.*;
…
double d1 = cos(…);   // Instead of Math.cos(…)
double d2 = sin(…);   // Instead of Math.sin(…)
double d3 = random(); // Instead of Math.random()
```

JUnit Overview

Motivation

- **Unit testing in general**
  - Testing individual methods or small pieces of functionality. Testing overall behavior often not sufficient because not all code cases are used in tests.
  - Whenever you modify code, you can rerun the test cases to verify you are still getting same answer
- **JUnit in particular**
  - Most popular and widely used unit testing framework in Java world. Easy to learn basics.
    - Not the only unit testing framework, or even necessarily the best for all situations. Almost all Java newcomers should start with JUnit first.
  - Not part of official Java SE
  - Integrated with Eclipse and other IDEs

Using JUnit in Eclipse: Simple Usage (New Style)

- **Put @Test above any zero-arg method**
  - Eclipse will prompt you to include the JUnit library and will automatically import org.junit.*;
- **Use import static org.junit.Assert.*; and import static org.hamcrest.CoreMatchers.*;**
  - Lets you use assertThat, etc. without class name
- **Test with assertThat**
  - Make tests with assertThat(someValue, someMatcher)
- **R-click in code, Run As → JUnit Test**
  - Check results printed by Eclipse

Using JUnit in Eclipse: Simple Usage (Traditional Style)

- **Put @Test above any zero-arg method**
  - Eclipse will prompt you to include the JUnit library
- **Use import static org.junit.Assert.*;**
  - Lets you use assertTrue, etc. without class name
- **Test with assertTrue, assertEquals, etc.**
  - Make tests with assertTrue(value), assertFalse(value), assertEquals(val1, val2)
- **R-click in code, Run As → JUnit Test**
  - Check results printed by Eclipse

Documentation

- **Home page**
  - http://junit.org/
    - Many more options than the simple ones shown here
- **Assertions**
  - Modern style
    - https://github.com/junit-team/junit/wiki/Matchers-and-assertthat
  - Traditional style
    - https://github.com/junit-team/junit/wiki/Assertions
- **JavaDoc**
  - http://junit.org/javadoc/latest/
    - For the new style, see especially CoreMatchers
  - http://hamcrest.org/JavaHamcrest/javadoc/1.3/

Modern Approach

Overview

- **Setup**
  - Use imports
    - import org.junit.*;
    - import static org.junit.Assert.*;
    - import static org.hamcrest.CoreMatchers.*;
  - Make zero-arg method marked with @Test
- **Create tests with assertThat(val, matcher)**

```
int n = someCalculation();
assertThat(n, is(equalTo(17)));
String s = someOtherCalculation();
assertThat(s, containsString("blah"));
```

- **Run in Eclipse**
  - R-click in code, Run As → JUnit Test
  - Eclipse will show pass (green) or fail (red) results

Core Builtin Matcher: is

- **With simple value, synonymous to equalTo**
  - – assertThat(num, is(12));
  - – assertThat(num, is(equalTo(12));
- **With matcher, just syntactic sugar**
  - – So omitting “is” has no effect except for readability
    - assertThat(someString, is(equalTo("blah"));
    - assertThat(someString, equalTo("blah");

    - assertThat(someObject, is(nullValue()));
    - assertThat(someObject, nullValue());

Other Core Matcher Types

- **Testing numbers**
  - – equalTo, closeTo
    - To use closeTo, you must load the full hamcrest library and import static org.hamcrest.number.IsCloseTo.*;
- **Testing object values**
  - – equalTo, instanceOf, nullValue, notNullValue, sameInstance
- **Strings and lists**
  - – containsString, startsWith, endsWith, hasItem, hasItems
- **Combining tests**
  - – not, anyOf, allOf
    - not takes one matcher
    - anyOf and allOf take multiple matchers

Mini Examples

- **Equality**
  - assertThat(foo, is(equalTo(bar)))
- **Boolean true**
  - assertThat(foo, is(true))
- **Boolean false**
  - assertThat(foo, is(not(true)))
- **Contains strings**
  - assertThat(string1, containsString(string2))
- **Contains elements**
  - assertThat(list1, hasItem(blah))
- **Combined tests**
  - assertThat(string1, anyOf(nullValue(), startsWith("q")))
  - assertThat(list1, allOf(hasItem("foo"), hasItem("bar")));

Testing Example

- **reverseString**
  - Should reverse a string, preserving case
- **isPalindrome**
  - Should return true if and only if the string reads the same backward and forward, ignoring case differences
- **Examples taken from**
  - File IO section

Testing Example: Current Implementation

```
public class StringUtils {

  /** Returns a reversed copy of a non-null String. */

  public static String reverseString(String s) {
    return(new StringBuilder(s).reverse().toString());
  }

  /** Checks if a String is a palindrome. Accepts
   *  zero-length or one-length strings, but not null.
   */
  public static boolean isPalindrome(String s) {
    return(s.equalsIgnoreCase(reverseString(s)));
  }

  private StringUtils() {}
}
```

JUnit Test (Part 1)

```
package coreservlets;

import org.junit.*;
import static org.junit.Assert.*;
import static org.hamcrest.CoreMatchers.*;
```

Lets you use assertThat without the class name.

See JavaDocs for CoreMatchers for details on matchers like is, hasItem, anyOf, etc.

JUnit Test (Part 2)

Eclipse knows to run this when you R-click
and choose Run As → JUnit Test

```
public class StringUtilsTester {
  @Test
  public void testReverse() {
    assertThat("oof",
               is(equalTo(StringUtils.reverseString("foo"))));
    assertThat("rab",
               is(equalTo(StringUtils.reverseString("bar"))));
    assertThat("!zaB",
               is(equalTo(StringUtils.reverseString("Baz!"))));
  }
```

If any of the tests fail, you get red error message in the Eclipse JUnit window.

Slightly longer than using assertEquals, the traditional approach shown later. But:

- More readable
- If you prefer, you can shorten is(equalTo(blah)) to is(blah)
- Typesafe: won't compile if argument to equalTo is of wrong type

JUnit Test (Part 3)

```
  @Test
  public void testPalindromes() {
    String[] matches =
      { "a", "aba", "Aba", "abba", "AbBa",
        "abcdeffedcba", "abcdEffedcba" };
    String[] misMatches =
      { "ax", "axba", "Axba", "abbax", "xAbBa",
        "abcdeffedcdax", "axbcdEffedcda" };
    for(String s: matches) {
      assertThat(StringUtils.isPalindrome(s), is(true));
    }
    for(String s: misMatches) {
      assertThat(StringUtils.isPalindrome(s), is(false));
    }
  }
}
```

Slightly longer than assertTrue and assertFalse (traditional approach). But

- More readable
- There are often more specific tests such as startsWith
- When combining tests with not, anyOf, or allOf, the result is much more readable

Traditional Approach

Overview

- **Setup**
  - Use imports
    - import org.junit.*;
    - import static org.junit.Assert.*;
  - Make zero-arg method marked with @Test
- **Create tests with assertTrue, assertFalse, assertEquals**

```
int n = someCalculation();
assertEquals(n, 17);
String s = someOtherCalculation();
assertTrue(s.containsString("blah"));
```

- **Run in Eclipse**
  - R-click in code, Run As → JUnit Test
  - Eclipse will show pass (green) or fail (red) results

Traditional Approach: Summary

- **assertEquals**
  - assertEquals("some string", someMethodCall(…))
  - assertEquals(var1, var2);
- **assertTrue**
  - assertTrue(someString.contains(someSubstring))
  - assertTrue(someList.contains(someItem))
  - assertTrue(someBoolean)
- **assertFalse**
  - assertFalse(someString.contains(someSubstring))
  - assertFalse(someList.contains(someItem))
  - assertFalse(someBoolean)

JUnit Test (Part 1)

```
package coreservlets.java8;

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


public class StringUtilsTester {
  @Test
  public void testReverse() {
    assertEquals("oof", StringUtils.reverseString("foo"));
    assertEquals("rab", StringUtils.reverseString("bar"));
    assertEquals("!zaB", StringUtils.reverseString("Baz!"));
  }
```

Lets you use assertEquals instead of Assert.assertEquals

If any of the pairs are not equal, you will get error message in the Eclipse JUnit window

JUnit Test (Part 2)

```
  @Test
  public void testPalindromes() {
    String[] matches =
      { "a", "aba", "Aba", "abba", "AbBa",
        "abcdeffedcba", "abcdEffedcba" };
    String[] misMatches =
      { "ax", "axba", "Axba", "abbax", "xAbBa",
        "abcdeffedcdax", "axbcdEffedcda" };
    for(String s: matches) {
      assertTrue(StringUtils.isPalindrome(s));
    }
    for(String s: misMatches) {
      assertFalse(StringUtils.isPalindrome(s));
    }
  }
}
```

If any of the arguments fail to evaluate to true (assertTrue) or false (assertFalse), you will get error message in the Eclipse JUnit window

Wrap-Up

Summary

- **Write unit tests from beginning**
  - Rerun whenever you change the code
- **Example usage**

```
@Test
public void someMethod() {
  String blah = someFancyComputation();
  assertThat(blah, allOf(notNullValue(),
                         startsWith("q"),
                         not(contains("z"))));
  List<String> items = someMethod();
  assertThat(items, hasItem("foobar"));
  Blah b1 = doComputationOneWay();
  Blah b2 = doComputationAnotherWay();
  assertThat(b1, is(equalTo(b2)));
}
```

Questions?

More info:
http://courses.coreservlets.com/Course-Materials/java.html – General Java programming tutorial
http://www.coreservlets.com/java-8-tutorial/ – Java 8 tutorial
http://courses.coreservlets.com/java-training.html – Customized Java training courses, at public venues or onsite at *your* organization
http://coreservlets.com/ – JSF 2, PrimeFaces, Java 7 or 8, Ajax, jQuery, Hadoop, RESTful Web Services, Android, HTML5, Spring, Hibernate, Servlets, JSP, GWT, and other Java EE training
Many additional free tutorials at coreservlets.com (JSF, Android, Ajax, Hadoop, and lots more)

Customized Java EE Training: http://courses.coreservlets.com/
Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.