Unit Testing with JUnit

Why Testing Is Important
and
How to Test Effectively

Gregg Lippa
Senior Technical Analyst
Themis Education
Themis, Inc.

glippa@themisinc.com
Visit us at: www.themisinc.com
Also: www.themisinc.com/webinars

Themis
Leaders in IT Education

Questions Addressed in ThisWebinar

- What is unit testing is and why it is important?
- How does JUnit work? How does Eclipse support it?
- What is a JUnit Test Case class?
- What kinds of things that should be tested?
- How do you use JUnit to test those things?
- How do you execute a JUnit test case and review results?
- What is a TestSuite and how do you create one?
- What are the best practices for JUnit testing?
- What is on the horizon for JUnit in 2017?
- We will have very little time for questions
  - Please submit questions via email to glippa@themisinc.com

Themis
Leaders in IT Education

Unit Testing Objectives

- Create high **quality code**
  - Removal of defects increases quality
  - Testing is easier with well written code
    - So is enhancing and maintaining the code
- **Find bugs** early
  - The later in the development process bugs arise, the more expensive it is to correct them
- **Support the requirements** of the interface
  - Testing can assist in the process of refactoring code
    - Often provides a rationale for refactoring
- Increase **developer confidence** in code
  - Improve developer productivity

Developers Often Avoid Unit Testing

- Why?
  - "There isn't time for it"
  - "It's too complicated"
  - "My manager doesn't think it's worth the effort"
  - "It's boring – I'd rather be developing code"
  - "We let others take care of the testing"
- Additional types of testing
  - Integration Testing
  - System Testing
  - User Acceptance Testing

Themis
Leaders in IT Education

How to Make Unit Testing Succeed

- Keep it simple
  - Encourages use and reuse
  - Saves time and reduces resource requirements
- Plan for it
  - Plan for Unit Testing during project inception
  - Allow other team members to assist and comment to create the most comprehensive testing effort possible
- Document it
  - Unit Test Plans document the expected inputs and results
    - All possible sets of circumstances must be tested
  - Regression testing is supported by test cases documentation
  - Standardize Unit Test Plan form for use by all team members

Thorough and Automated Unit Testing

- Test every line of code
  - Positive tests: code works as it should when things go well
  - Negative tests: code fails as it should when things go wrong
- Test for all possible values
  - Numerics: minimum and maximum allowable values
  - Character: minimum and maximum field lengths
- Automate the process
  - Eases burden on developers
    - Makes tests more likely to be run
  - Simplifies regression testing
- Use a testing framework (like JUnit)

Automated Testing Advantages

- Easy to use **GUI interface**
  - Indicates location of problems
- Simplifies **comparison of test results to expected results**
- Utilizes a “test class”
  - Leverages developer knowledge of Java language
  - Eliminates need to repeatedly reenter test data
- Supports **reuse of test cases** for
  - Re-testing after fixing bugs
  - **Regression testing**
- Standardized approach shortens learning curve

Best Practices

- Develop a written test plan
- Use a **separate test case class for each class being tested**
- Create separate **tests for each path through the code**
  - Usually multiple test methods for each method being tested
    - Test each possible positive and negative outcome in a separate method
- Keep test-cases in sync with code as requirements change
- Use **source code control** to version test-case code
  - Along with business code
- Automate as much as possible
- **Regression test** entire class or sets of related classes following significant code changes

JUnit Overview

The JUnit Framework

Code a little
Test a little

- Helps develop and execute repeatable test cases
- Offers ways to **describe how code is expected to work**
- Tests **automatically report their results** using a GUI
- Sets of related unit tests may be collected into test suites
- Proven!
  - Kent Beck and Erich Gamma introduced JUnit
    - *Java Report*, Vol. 3, No. 7, July 1998:
      "Test Infected: Programmers Love Writing Tests"

JUnit website:
http://junit.org

Themis
Leaders in IT Education

Step By Step

Code a little
Test a little

1. Create a **business class**
   - Add stubs for its required methods
2. Create a **test case class** for the business object
   - Parent class is provided by JUnit framework: TestCase
   - Best practice: create a separate source folder with the same package structure to contain the test case
3. Code **test methods in the test case**
   - Test methods will call the business methods of the object being tested and compare actual versus expected results
4. Add code to **implement the business methods**
   - This can be done **after** test methods have been coded
5. **Run the test cases** as JUnit Tests
   - Entire test case
   - Individual test methods
   - Test suites containing groups of test cases

Themis
Leaders in IT Education

Basic JUnit Assert Methods

- TestCase assertion methods provided by the framework
  - Automatically **compare actual against expected** test results
- Assertion methods return void
  - Throw **AssertionFailedError** if assertion fails (i.e. is not true)
    - Caught by framework code for automated reporting
  - Assertion methods are overloaded
    - Support optional first parameter of type String to provide message text to AssertionFailedError if thrown

assertEquals( )

assertSame( )

assertTrue( )

assertFalse( )

assertNull( )

assertNotNull( )

fail( )

JUnit Assert Methods Detail

`assertEquals(expected-value, actual-value)`

- Asserts the two arguments are equal
- Supports objects and primitives

`assertEquals(expected, actual, allowed-difference)`

- Asserts the two **float or double** arguments are equal
- Additional required argument: difference allowed to consider them equal

`assertSame(object-reference-1, object-reference-2)`

- Asserts the two arguments refer to the same object

`assertTrue(single-boolean-argument)`

- Asserts the argument is true

`assertFalse(single-boolean-argument)`

- Asserts the argument is false

JUnit Assert and Fail Methods Detail

```
assertNull(single-object-reference)
```

- Asserts the argument is null

```
assertNotNull(single-object-reference)
```

- Asserts the argument is not null

```
fail()
```

- Always fails a test and throws an AssertionFailedError
  - Displays the method name of the test that failed
- Intended for code that should not be executed for this test
  - E.g., after a call being tested which should throw an exception
- Examples:

```
assertEquals("Jones", emp.getName());
```

```
assertTrue(emp.getSalary() < 50000);
```

Themis
Leaders in IT Education

JUnit Assert Methods – Additional Parameter

Assertion methods are overloaded to support an optional first parameter of type String to provide message text to AssertionFailedError if thrown

```
assertEquals(string-msg, expected-value, actual-value)
```

```
assertEquals(string-msg, expected, actual, allow-diff)
```

```
assertSame(string-msg, object-ref-1, object-ref-2)
```

```
assertTrue(string-msg, single-boolean-argument)
```

```
assertFalse(string-msg, single-boolean-argument)
```

```
assertNull(string-msg, single-object-reference)
```

```
assertNotNull(string-msg, single-object-reference)
```

```
fail(string-msg)
```

Themis
Leaders in IT Education

Additional Options – AssertThat( )

- Newer, **more readable**, assertion mechanism
  - **`AssertThat()`** method structure – subject, verb, object
- Note that **expected and actual are reversed** compared to the other assert methods
  - Actual value is in the first argument
  - Expected value is contained in a **Matcher**
    - Many types implement the Matcher interface
    - Details: http://junit.org/junit4/javadoc/latest/
      - Notable packages: `org.hamcrest` and `org.hamcrest.core`
- A few examples:

```
assertThat(emp.getAge(), is(30));
assertThat(emp.getJob, is(equalTo("Manager")));
assertThat(name, either(containsString("x")).or(startsWith("Z")));
assertThat(nameList, hasItem("Jones"));
```

Themis Leaders in IT Education

org.hamcrest.CoreMatchers

Contains numerous Classes that implement **`Matcher`**

`allOf`
`any`
`anyOf`
`anything`
`both`
`containsString`
`describedAs`
`either`
`endsWith`
`equalTo`
`everyItem`

`hasItem`
`hasItems`
`instanceOf`
`is`
`isA`
`not`
`notNullValue`
`nullValue`
`sameInstance`
`startsWith`
`theInstance`

Also note:
Third party
Matchers exist
to support

`JSON`

`XML/XPath`

`Excel`

Also **`org.hamcrest.core.CombinableMatchers`**

`and`
`both`
`describeTo`

`either`
`matchesSafely`
`or`

You can also
write your own
custom Matchers
by extending
`BaseMatcher`

Themis
Leaders in IT Education

Business Class To Be Tested

```
package com.themis.apps;

public class Product {

    private int id;
    private String name;
    private double cost;

    public Product(int id, String name, double cost) {
        setId(id);
        setName(name);
        setCost(cost);
    }

    public int setName(String name) {
        int returnValue = 0;
        // This code deliberately contains an error
        // Length of name must be greater than 0
        if (name == null || name.length() < 0
            || name.length() > 15) {
            returnValue = -1;
        }
        else {
            this.name = name;
        }

        return returnValue;
    }
                            . . . .
```

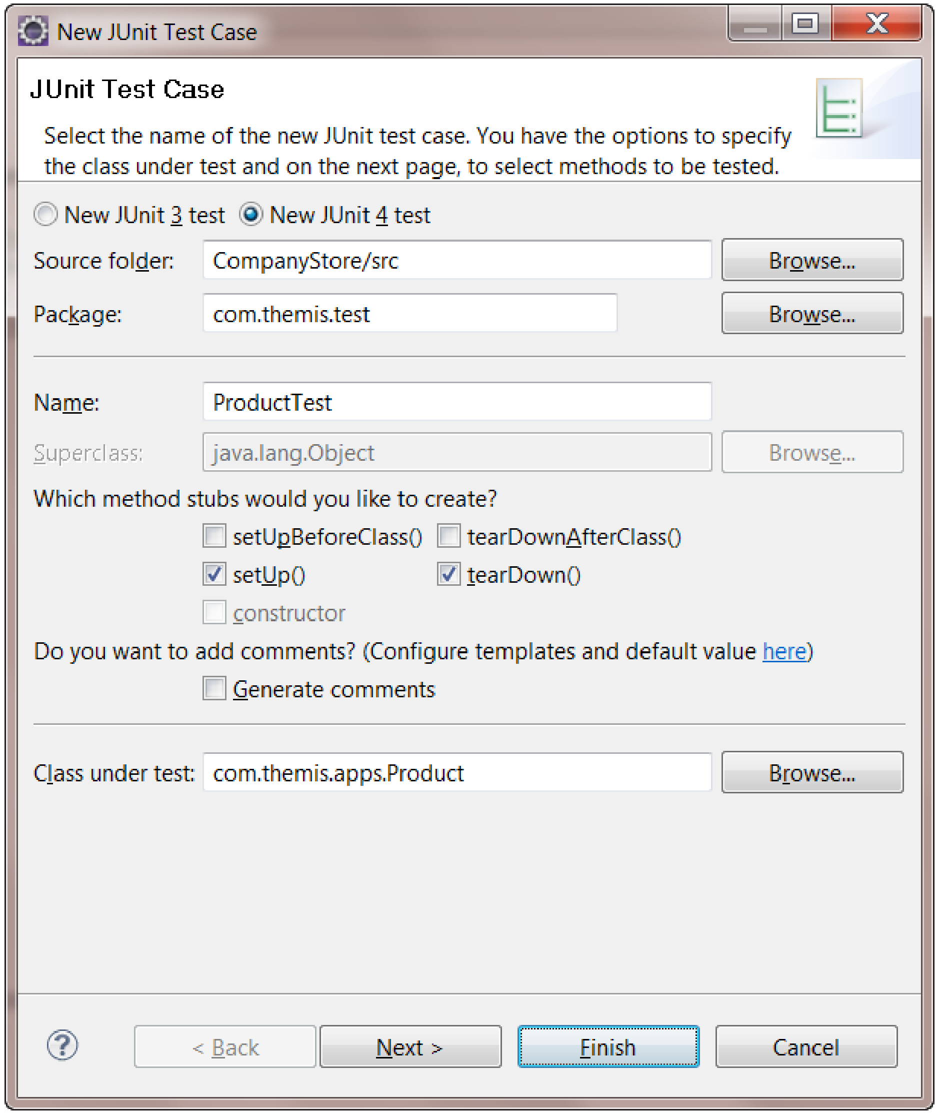
Code defect to be discovered by a suitable test case

Other code in this class is not shown

Themis
Leaders in IT Education

New JUnit 4 Test Case

- From New Dialog:
  - Java | JUnit | JUnit Test case
- Choose radio button: New Junit 4 test
- Change package name from apps to test and check checkboxes shown
- Class under test: com.themis.apps.Product
- Click Next



JUnit Test Case

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

New JUnit 3 test [✓ New JUnit 4 test]

Source folder: CompanyStore/src

Package: com.themis.test

Name: ProductTest

Superclass: java.lang.Object

Which method stubs would you like to create?

- [ ] setUpBeforeClass()
- [ ] tearDownAfterClass()
- [✓] setUp()
- [✓] tearDown()
- [ ] constructor

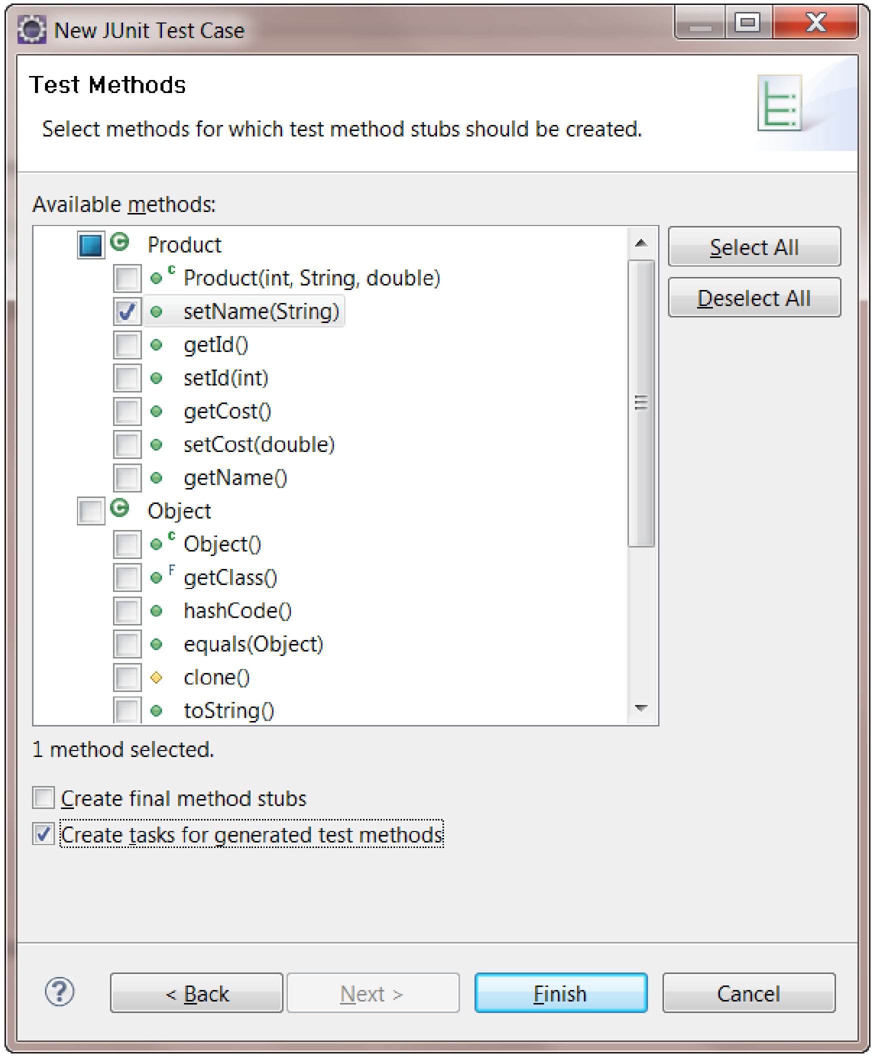
Do you want to add comments? (Configure templates and default value here)

- [ ] Generate comments

Class under test: com.themis.apps.Product

Themis
Leaders in IT Education

New JUnit 4 Test Case
Select Methods to Test



New JUnit Test Case

Test Methods

Select methods for which test method stubs should be created.

Available methods:

- [■] Product
  - [ ] Product(int, String, double)
  - [✓] setName(String)
  - [ ] getId()
  - [ ] setId(int)
  - [ ] getCost()
  - [ ] setCost(double)
  - [ ] getName()
- [ ] Object
  - [ ] Object()
  - [ ] getClass()
  - [ ] hashCode()
  - [ ] equals(Object)
  - [ ] clone()
  - [ ] toString()

1 method selected.

- [ ] Create final method stubs
- [✓] Create tasks for generated test methods

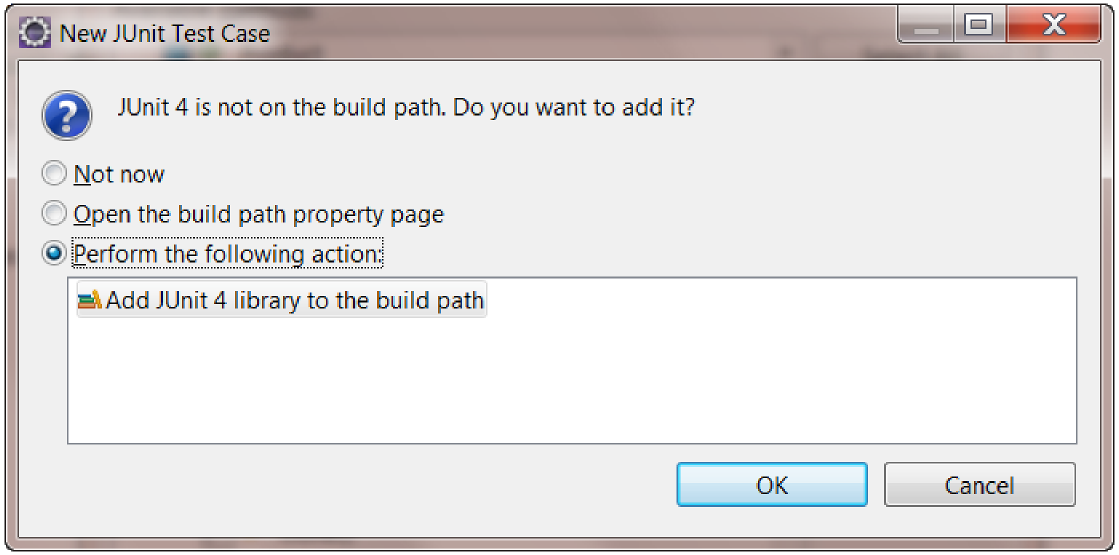
- Click method(s) to be tested
- Check “Create tasks for generated test methods"
- Click Finish

New JUnit 4 Test Case Build Path Fixup

- The window below appears because JUnit needs to be added to the project build path

- Click OK

- Once added to the Project Build Path, this dialog will no longer appear when adding JUnit tests



New JUnit Test Case

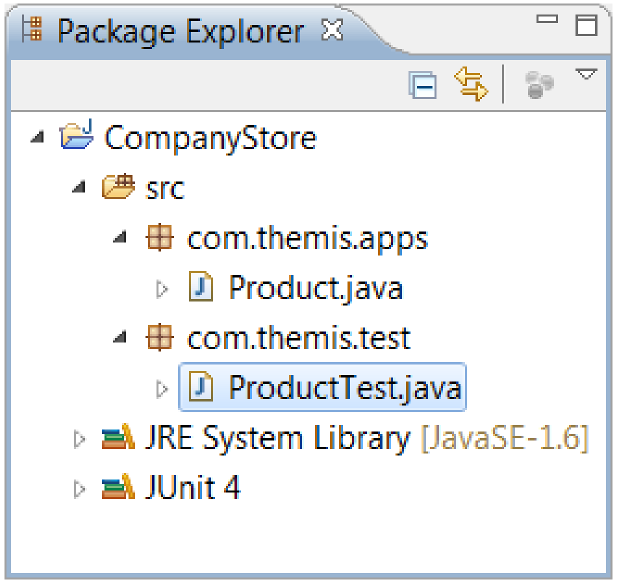
JUnit 4 is not on the build path. Do you want to add it?

( ) Not now
( ) Open the build path property page
(●) Perform the following action:

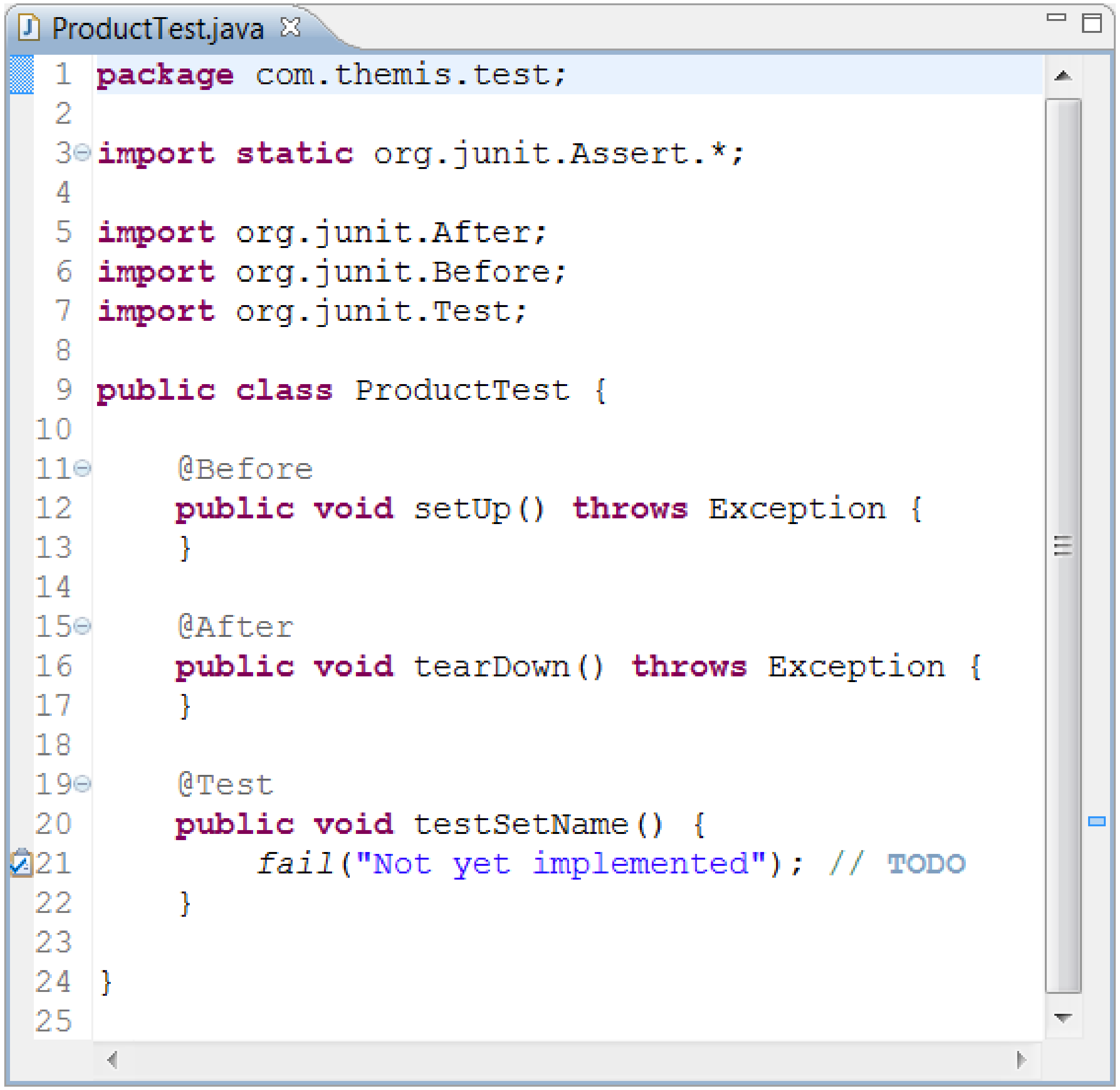
Add JUnit 4 library to the build path

New JUnit 4 Test Case Created

- A new Java class has been created and opened
- Notice the three methods that have been created

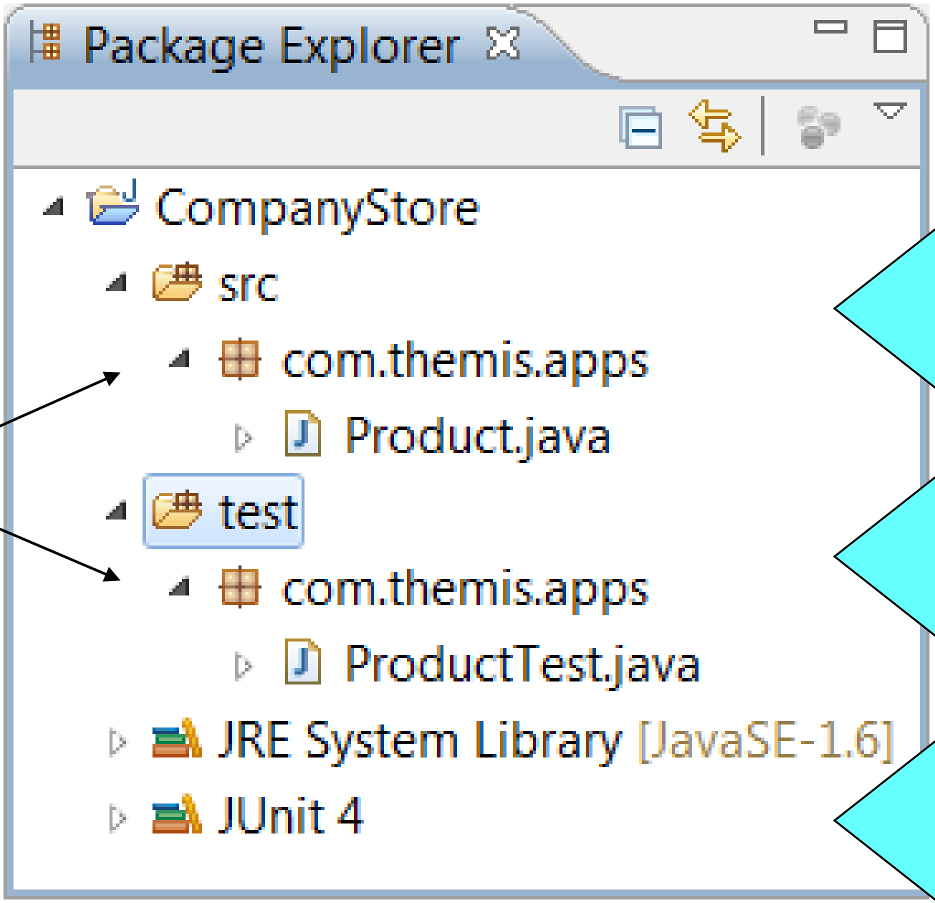


```
CompanyStore
  src
    com.themis.apps
      Product.java
    com.themis.test
      ProductTest.java
  JRE System Library [JavaSE-1.6]
  JUnit 4
```



```java
package com.themis.test;

import static org.junit.Assert.*;

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

public class ProductTest {

    @Before
    public void setUp() throws Exception {
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public void testSetName() {
        fail("Not yet implemented"); // TODO
    }

}
```

Alternative Project Source Folder Structure for Test Cases



```
CompanyStore
  src
    com.themis.apps
      Product.java
  test
    com.themis.apps
      ProductTest.java
  JRE System Library [JavaSE-1.6]
  JUnit 4
```

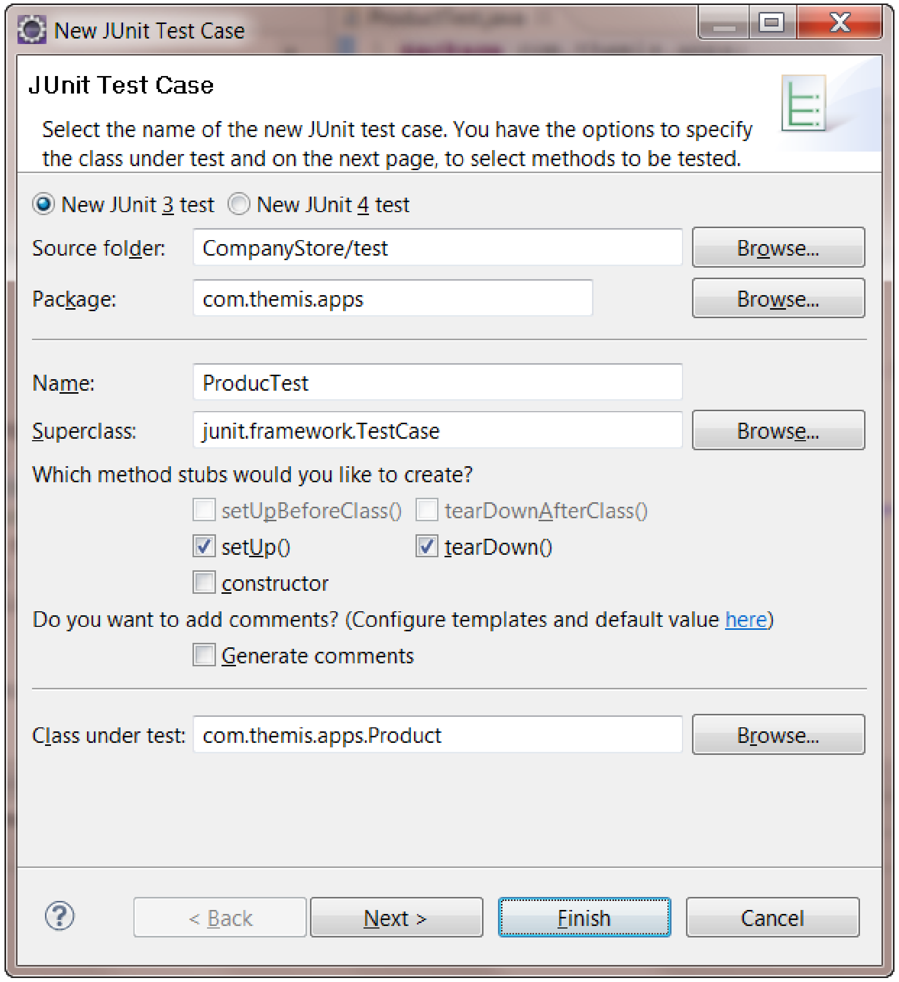
The same package structure exists in each source folder

src folder contains business code to test

test folder contains one TestCase for each business class

Additional library (jar files) needed

JUnit 3 Still Supported



New JUnit Test Case

JUnit Test Case

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

[✓ New JUnit 3 test] New JUnit 4 test

Source folder:	CompanyStore/test
Package:	com.themis.apps
Name:	ProducTest
Superclass:	junit.framework.TestCase

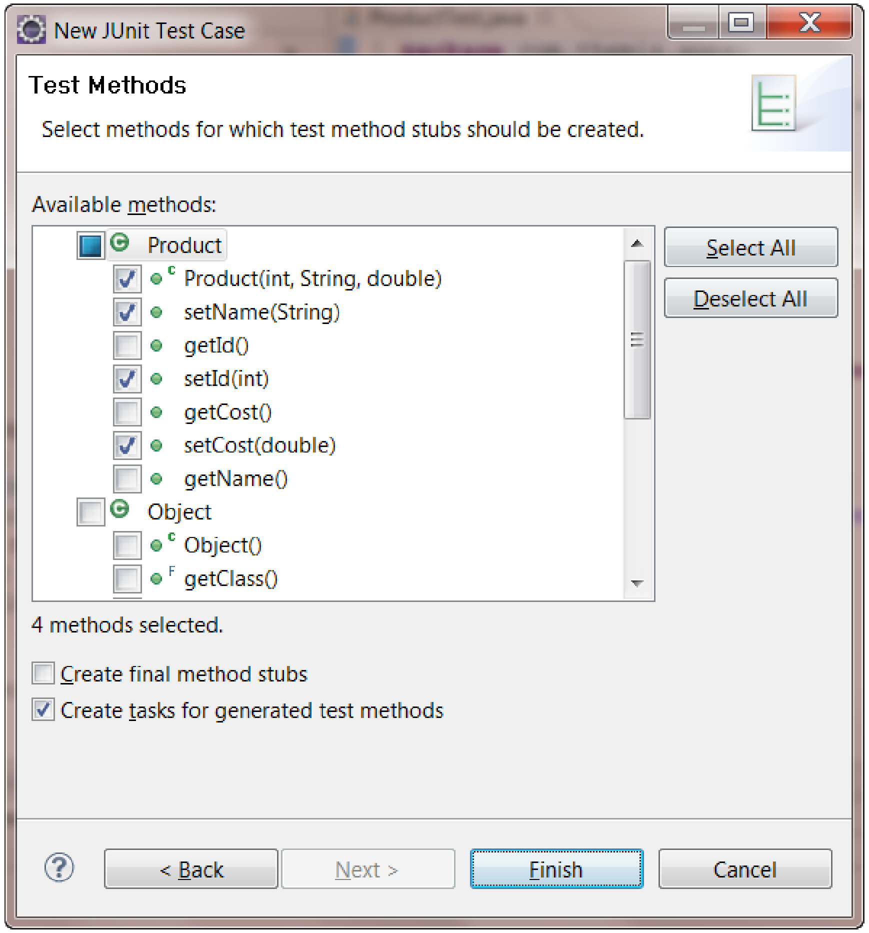
Which method stubs would you like to create?

- [ ] setUpBeforeClass()
- [ ] tearDownAfterClass()
- [x] setUp()
- [x] tearDown()
- [ ] constructor

Do you want to add comments? (Configure templates and default value here)

- [ ] Generate comments

Class under test: com.themis.apps.Product



New JUnit Test Case

Test Methods

Select methods for which test method stubs should be created.

Available methods:

- Product
  - [x] Product(int, String, double)
  - [x] setName(String)
  - [ ] getId()
  - [x] setId(int)
  - [ ] getCost()
  - [x] setCost(double)
  - [ ] getName()
- [ ] Object
  - [ ] Object()
  - [ ] getClass()

4 methods selected.

- [ ] Create final method stubs
- [x] Create tasks for generated test methods

What's In the Test Case?

```
package com.themis.apps;
import junit.framework.TestCase;

public class ProductTest extends TestCase {
   public ProductTest(String arg0) {
      super(arg0);
   }
   protected void setUp() throws Exception {
      super.setUp();
   }
   protected void tearDown() throws Exception {
      super.tearDown();
   }
   public void testProduct() {
      fail("Not yet implemented");
   }
   public void testSetCost() {
      fail("Not yet implemented");
   }
   public void testSetId() {
      fail("Not yet implemented");
   }
   public void testSetName() {
      fail("Not yet implemented");
   }
}
```

JUnit 3.8 Tests extend the superclass TestCase

Runs prior to execution of each test method

Runs following execution of each test method

JUnit 3.8 does not use annotations

- **All test code is placed in "test" methods**
  - **Names must begin with "test"**
- **Each test method should be independent**
  - **The order in which test methods execute is not guaranteed**

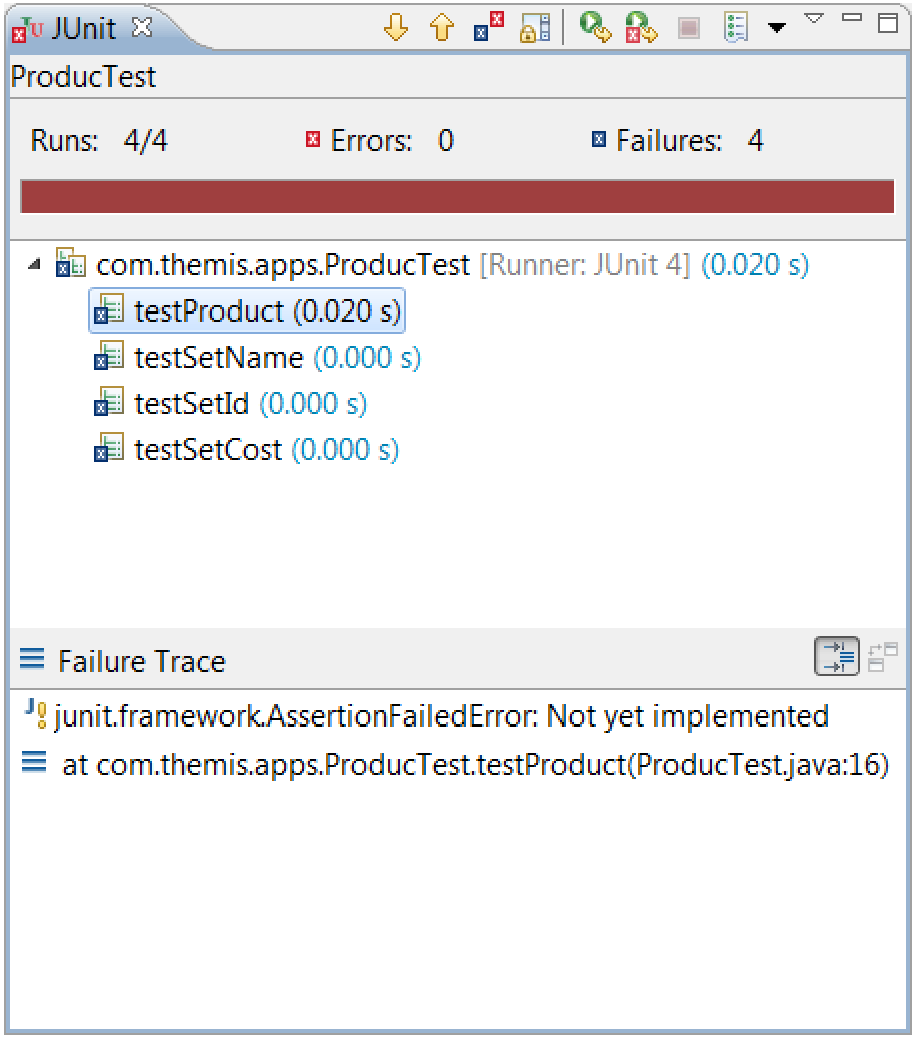
Themis
Leaders in IT Education

Run Unimplemented Test Case

- Run ➔
- Run As ➔
- JUnit Test

Keep the bar green to keep the code clean

Wizard generates code causing all tests to fail



```
ProducTest

Runs:  4/4        Errors:  0        Failures:  4

com.themis.apps.ProducTest [Runner: JUnit 4] (0.020 s)
    testProduct (0.020 s)
    testSetName (0.000 s)
    testSetId (0.000 s)
    testSetCost (0.000 s)

Failure Trace
junit.framework.AssertionFailedError: Not yet implemented
at com.themis.apps.ProducTest.testProduct(ProducTest.java:16)
```

Themis
Leaders in IT Education

Things to Test

- Test any code that does a significant bit of work
  - Test to ensure that it works correctly with **good data**
  - Test to ensure that it works correctly with **bad data**
  - Test with all possible types of good and bad data values
  - Getter methods typically do not need to be tested
- Test the **boundaries**
  - String length, integer value, array size, etc.
  - Test the minimum, the maximum, one less than the minimum, one more than the maximum, and a value in the middle of the range
- Ensure correct values following method call
  - Did the set method assigned the value properly?
  - Are the contents of the array or collection correct after execution?

That's right . . . five tests for each conditional check!

Leaders in IT Education

Constructing a JUnit Test

1. Call the business method being tested
2. Compare expected result versus actual result
   - JUnit framework supplies the assertXXX methods
     - For test failures, reports expected vs. actual value, message (optionally) provided by test developer, and location in code where failure occurred
   - Most commonly used in test code: **assertEquals**
     - assertEquals(expected-result, actual-result)
     - assertEquals("failure message", expected-result, actual-result)
     - Requires additional argument when comparing floats or doubles
       - assertEquals(expected-result, actual-result, allow-diff)
   - assertEquals accepts all primitive data types as well as Object
     - Uses .equals() or == as appropriate depending on types being compared
   - Also common: **AssertThat**
     - assertThat(actual-result, matcher-expression)

Sample Method to be Tested

This simple setter method checks for null values and also checks that the length of the incoming name is within a specified range

```
public int setName(String name) {
    int retVal = 0;
// This code deliberately contains an error
// Length of name must be greater than 0
    if (name == null
    || name.length() < 0
    || name.length() > 15) {
            retVal = -1;
    }
    else {
        this.name = name;
    }

    return retVal;
}
```

testSetName()

```
public void testSetName() {
  int returnCode;
  String actual;

  Product p = new Product(54321, "Printer", 49.99);
  returnCode = p.setName("LengthIs15Chars");
  assertEquals("expect ret val for success", 0, returnCode);
  actual = p.getName();
  assertEquals("set should work", "LengthIs15Chars", actual);

  returnCode = p.setName("LengthNow16Chars");
  assertEquals("16 chars: expect ret val fail", -1, returnCode);
  actual = p.getName();
  assertEquals("16 chars: no chg", "LengthIs15Chars", actual);

  returnCode = p.setName(null);
  assertEquals("null: expect ret val fail", -1, returnCode);
  actual = p.getName();
  assertEquals("null: no change", "LengthIs15Chars", actual);

  returnCode = p.setName("");
  assertEquals("0 chars: expect ret val fail", -1, returnCode);
  actual = p.getName();
  assertEquals("0 chars: no chg", "LengthIs15Chars", actual);

  returnCode = p.setName("1");
  assertEquals("expect ret val for success", 0, returnCode);
  actual = p.getName();
  assertEquals("set should work", "1", actual);
}
```

Note: some testers rename this to testGetSetName

Failure of any assert method terminates the execution of the current test method by throwing an AssertionFailedError

Execution of the JUnit TestCase continues with other test methods that have not yet run

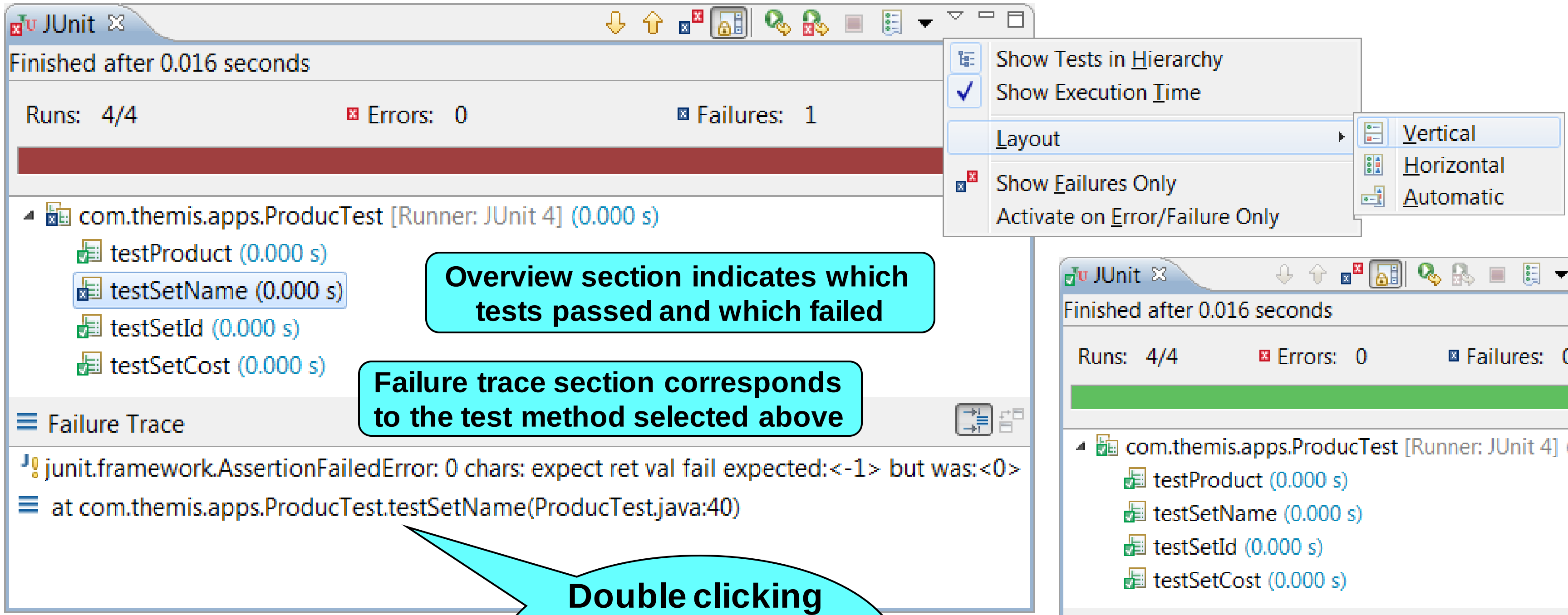
So . . .

It is better to have a separate test method for each scenario being tested

Themis
Leaders in IT Education

Test Case Execution

Run → Run As → JUnit Test



JUnit

Finished after 0.016 seconds

Runs: 4/4 Errors: 0 Failures: 1

com.themis.apps.ProducTest [Runner: JUnit 4] (0.000 s)
testProduct (0.000 s)
testSetName (0.000 s)
testSetId (0.000 s)
testSetCost (0.000 s)

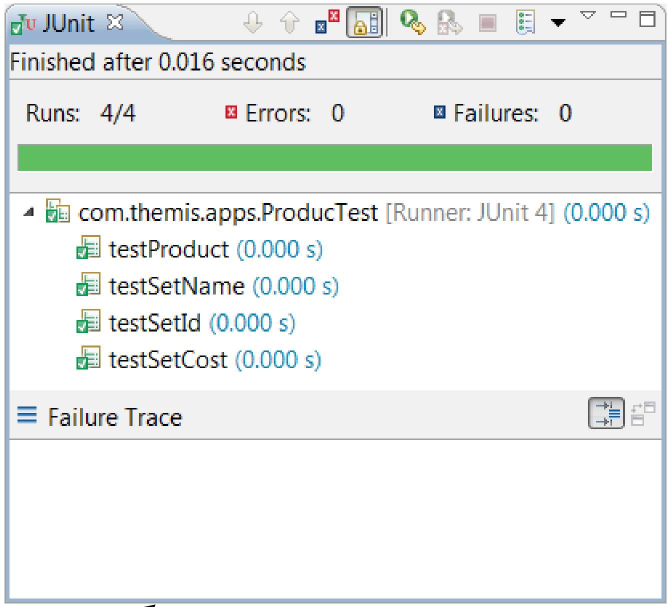
Failure Trace

```
junit.framework.AssertionFailedError: 0 chars: expect ret val fail expected:<-1> but was:<0>
at com.themis.apps.ProducTest.testSetName(ProducTest.java:40)
```

Overview section indicates which tests passed and which failed

Failure trace section corresponds to the test method selected above

Double clicking here takes you to the test case line whose assertion failed



JUnit

Finished after 0.016 seconds

Runs: 4/4 Errors: 0 Failures: 0

com.themis.apps.ProducTest [Runner: JUnit 4] (0.000 s)
testProduct (0.000 s)
testSetName (0.000 s)
testSetId (0.000 s)
testSetCost (0.000 s)

Failure Trace

```
if (name == null || name.length() < 1
    || name.length() > 15) {
    retVal = -1;
}
. . .
```

Fix coding error and try again: success!

Testing setName With Separate Methods

```
public void testSetNameMaxLength() {
  int returnCode;
  String actual;

  Product p = new Product(54321, "Printer", 49.99);
  returnCode = p.setName("LengthIs15Chars");
  assertEquals("expect ret val for success", 0, returnCode);
  actual = p.getName();
  assertEquals("set should work", "LengthIs15Chars", actual);
}
public void testSetNameTooLong() {
  int returnCode;
  String actual;

  Product p = new Product(54321, "Printer", 49.99);
  returnCode = p.setName("LengthNow16Chars");
  assertEquals("16 chars: expect ret val fail", -1, returnCode);
  actual = p.getName();
  assertEquals("16 chars: no chg", "Printer", actual);
}
public void testSetNameNullValue() {
  int returnCode;
  String actual;

  Product p = new Product(54321, "Printer", 49.99);
  returnCode = p.setName(null);
  assertEquals("null: expect ret val fail", -1, returnCode);
  actual = p.getName();
  assertEquals("null: no change", "Printer", actual);
}
```

Notice that the assertEquals calls testing that the value was set correctly now compare to "Printer"

They are no longer dependent on the success of the first test running correctly

Themis
Leaders in IT Education

Testing setName With Separate Methods

```
public void testSetNameEmptyString() {
  int returnCode;
  String actual;

  Product p = new Product(54321, "Printer", 49.99);
  returnCode = p.setName("");
  assertEquals("0 chars: expect ret val fail", -1, returnCode);
  actual = p.getName();
  assertEquals("0 chars: no chg", "Printer", actual);
}
public void testSetNameMinimumLength() {
  int returnCode;
  String actual;

  Product p = new Product(54321, "Printer", 49.99);
  returnCode = p.setName("1");
  assertEquals("expect ret val for success", 0, returnCode);
  actual = p.getName();
  assertEquals("set should work", "1", actual);
}
public void testSetNameInTheMiddle() {
  int returnCode;
  String actual;

  Product p = new Product(54321, "Printer", 49.99);
  returnCode = p.setName("Average Name");
  assertEquals("expect ret val for success", 0, returnCode);
  actual = p.getName();
  assertEquals("set should work", "Average Name", actual);
}
```

Notice that a Product object is currently being instantiated in each of the tests

Themis
Leaders in IT Education

Test Fixtures

Q: How do you test the methods of Product?

A: Instantiate a Product so I have an object to call methods on.

Q: What if Product contains required references to other objects?

A: Instantiate those other objects and provide them to Product.

- Every test of any non-static method in a class requires an instance
- Previous examples instantiated the object in the testXXX method
  - This code appears repeatedly in every testXX method
- TestCase's setup( ) provides a place to instantiate **test fixtures**
  - Objects that are required by many or all of the testXXX methods

1. **`setUp()`** is called before each test method runs
   - Instantiate objects and assign to instance variables (**fixtures**)
2. The next test method is then called
   - Uses the fixtures
3. **`tearDown()`** is called after each test method runs
   - Clean up after the test runs

Test Fixture Example

```
import com.themis.apps.Product;
public class ProductTest {
    private Product p;
    . . .

    @Before
    public void setUp() throws Exception {
        p = new Product(54321, "Printer", 49.99);
    }

    @After
    public void tearDown() throws Exception {

    }

    @Test   // added
    public void testSetNameMaxLength() {

        returnCode = p.setName("LengthIs15Chars");
        assertEquals("expect ret val for success", 0, returnCode);
        actual = p.getName();
        assertEquals("set should work", "LengthIs15Chars", actual);
    }
        // Same approach taken with other testXXX methods
```

Declare fixture

Initialize fixture

This tearDown() is unnecessary

Use the fixture

Use the fixture

Constructor Testing

```
public Product(int id, String Name, String lastName) {
   setId(id);
   setName(name);
   setCost(cost);
}
```

- Constructor code often simply calls setters
  - Do not re-test the setters when testing the constructor
  - Simply check to make sure that:
    - Provided values were assigned correctly
    - Default values were assigned correctly
    - Exceptions are handled appropriately

```
@Test
public void testProduct() {
   Product p = new Product(54321, "Printer", 49.99);
   assertEquals(54321, p.getId());
   assertEquals("Printer", p.getName());
   assertEquals(49.99, p.getCost(), 0.02);   // changed
}
```

Themis
Leaders in IT Education

Testing With Exception Handling

```
public void setName(String name) throws InvalidParmException {
   if (name == null || name.length() < 1 || name.length() > 15) {
      throw new InvalidParmException("Name: " + Name);
   }
   else {
      this.name = name;
   }
}
```

Updated setName( ) now throws InvalidParmException

- Test that a method throws an exception correctly
  - TestCase's fail( ) is instrumental for this purpose

```
public void testSetName() throws InvalidParmException {
   try {
         p.setName("Valid name");
   }
   catch (InvalidParmException success) {
         fail("Valid name should not cause InvalidParmException");
   }
   assertEquals("Valid name", p.getName());

   try {
      p.setName(null);
      fail("Invalid name should cause InvalidParmException");
   }
   catch (InvalidParmException good) {
         assertEquals("Name: null", good.getMessage());
   }
   . . . Test other conditions
```

If InvalidParmException is unexpectedly thrown, JUnit will fail the test method

Should not enter catch block when good data passed

Should enter catch block when bad data passed

Ensure that exception messages are correct

Themis
Leaders in IT Education

Testing With Exception Handling V4

```
public void setName(String name) throws InvalidParmException {
   if (name == null || name.length() < 1 || name.length() > 15) {
      throw new InvalidParmException("Name: " + Name);
   } else {
      this.name = name;
}  }
```

Same updated setName() now throwing InvalidParmException

- JUnit 4 annotation for exception testing:
  - @Test (expected=SomeException.class)

```
@Test(expected=InvalidParmException.class)
public void testSetNameNull() {
     p.setName(null);
}
```

Requires InvalidParmException to be thrown for this test to succeed

- Use @Rule to support checking exception message
  - Also allows checking domain objects affected by a test

No try / catch blocks are required in test methods

```
@Rule
public ExpectedException thrown = ExpectedException.none();
@Test
public void testSetNameNullWithChecking() {
     thrown.expect(InvalidParmException.class);
     thrown.expectMessage("Name: null");
     thrown.expectMessage(startsWith("Name: null"));
     p.setName(null);
}
```

Ensure that exception message is correct

Check exception message using Matcher

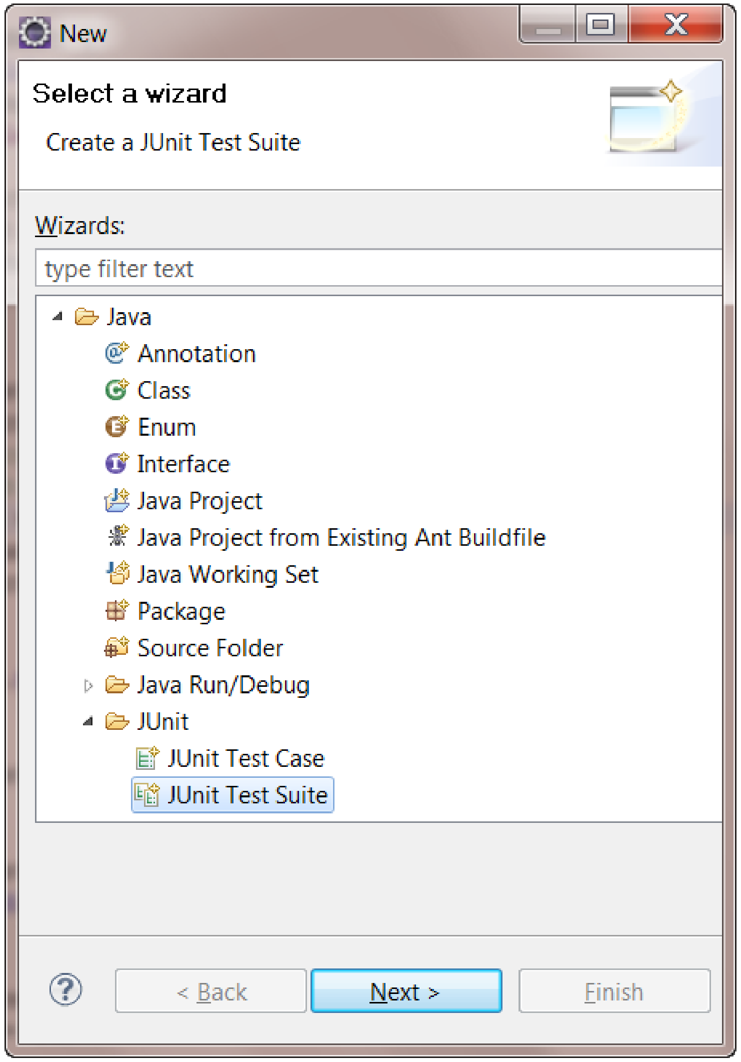
Other JUnit Capabilities

- Temporarily ignore a test with **`@Ignore`**
- Fail a test that runs too long using the **`timeout`** parameter
  - **`@Test(timeout=2000)`** ← **number of milliseconds**
- Make assumptions that must be true for test to pass with **`assumeThat`**
  - **`assumeThat(empList.size(), is(equalTo(20)));`**
  - Failed assumptions are ignored rather than causing the test to fail
- Use **`@Rule`** to alter how a test method is run and reported
  - See Javadoc for the **`org.junit.rules`** package
- Advanced features (out of scope for this presentation)
  - **`@Parameters`** – cross-product of test methods and a set of test data elements
  - **`@Theory`** – capture intended behavior for a large numbers of potential scenarios
  - **`@Category`** – run only certain tests based on categorizing test methods

Test Suites

- Many classes are created to support an application
  - Each business class should have a TestCase to support it
  - Running each TestCase can be a tedious undertaking
- **Group tests cases together** using TestSuites
  - Running TestSuite runs all of its contained TestCases
  - TestSuites may also contain other TestSuites

TestSuite Creation



New

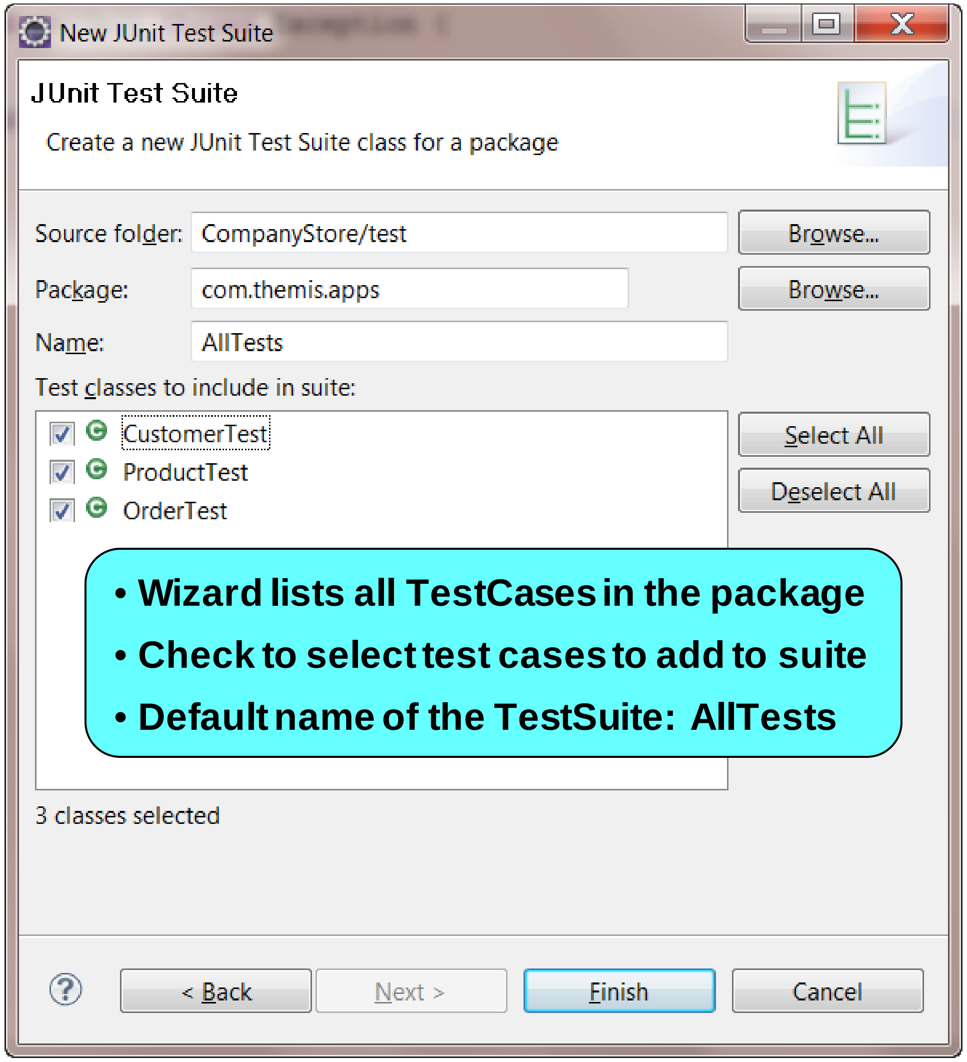
Select a wizard

Create a JUnit Test Suite

Wizards:

type filter text

- Java
  - Annotation
  - Class
  - Enum
  - Interface
  - Java Project
  - Java Project from Existing Ant Buildfile
  - Java Working Set
  - Package
  - Source Folder
  - Java Run/Debug
  - JUnit
    - JUnit Test Case
    - JUnit Test Suite



New JUnit Test Suite

JUnit Test Suite

Create a new JUnit Test Suite class for a package

Source folder: CompanyStore/test

Package: com.themis.apps

Name: AllTests

Test classes to include in suite:

- ✓ CustomerTest
- ✓ ProductTest
- ✓ OrderTest

3 classes selected

- **Wizard lists all TestCases in the package**
- **Check to select test cases to add to suite**
- **Default name of the TestSuite: AllTests**

JUnit 3 TestSuite

- Run the TestSuite exactly the same way a TestCase is run
  - Run → Run As → JUnit Test

```
package com.themis.apps;

import junit.framework.Test;
import junit.framework.TestSuite;

public class AllTests {

    public static Test suite() {
        TestSuite suite = new TestSuite(AllTests.class.getName());
        //$JUnit-BEGIN$
        suite.addTestSuite(CustomerTest.class);
        suite.addTestSuite(ProductTest.class);
        suite.addTestSuite(OrderTest.class);
        //$JUnit-END$
        return suite;
    }
}
```

All this code is generated by the wizard

- Wizard offers to include classes defined with **`extends TestCase`**
- Contents of TestSuite can be adjusted after it has been created
- Right-click over TestSuite in Package Explorer
  - Select "Recreate Test Suite…"

Themis
Leaders in IT Education

TestSuites in JUnit 4

- JUnit 4 TestCases do not include the clause **`extends TestCase`**
  - Different approach to geneerated TestSuite
- Instead, the `@RunWith` and `@SuiteClasses` annotations are used
  - Use `@RunWith(Suite.class)` exactly as shown
  - Use `@SuiteClasses` with each TestCase class entity as shown
    - Comma separated list contained within parentheses and braces

```
package com.themis.apps;

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

@RunWith(Suite.class)
@SuiteClasses({
   CustomerTest.class,
   ProductTest.class,
   OrderTest.class,
})
public class AllTestsV4 { /* empty class */ }
```

JUnit 4 TestSuite configuration is done using JUnit annotations

No change in how to run the JUnit 4 TestSuite

- Run → Run As → JUnit Test

Themis
Leaders in IT Education

Best Practices

- For every application class, create one JUnit TestCase
- Test methods should be small and focused
- Use test fixtures and use **`setUp()`** to instantiate them
  - Minimizes repeating the same code in every test method
- Each test case should be autonomous
  - Never depend on results of another test case
  - The order in which tests run is not assured
- Possible to control order in which tests run (rarely needed)
  - TestSuites can specify which test method to execute
    - Use **`@Category`** on a test method to define a category name
    - Use **`@IncludeCategory`** on the test suite to execute tests in that category

Preview of JUnit 5

- General Availability of JUnit 5.0 due Q3 2017
  - The JUnit 5 team released Milestone 4 in April 2017
- JUnit 5 goals:
  - Support new features in Java 8
  - Enable different styles of testing
- JUnit 5 is composed of modules from three different sub-projects
  - **JUnit 5 = *JUnit Platform* + *JUnit Jupiter* + *JUnit Vintage***
  - **JUnit Platform** – foundation for launching testing frameworks on the JVM
  - **JUnit Jupiter** – combination of the new programming model and extension model for writing tests and extensions in JUnit 5
  - **JUnit Vintage** – provides a TestEngine for running JUnit 3 and JUnit 4 tests
- For more detail: http://junit.org/junit5/docs/current/user-guide/
- There are numerous changes to the set of supported annotations

JUnit 5 requires Java 8 at runtime

Annotation Changes in JUnit 5

- Some JUnit 5 annotations **rename** those in JUnit 4:
  - ***@BeforeEach*** (previously *@Before*) – executed before each test method
  - ***@AfterEach*** (previously *@After*) – executed after each test method
  - ***@BeforeAll*** (previously *@BeforeClass*) – executed before all test methods
  - ***@AfterAll*** (previously *@AfterClass*) – executed after all test methods
  - ***@Disable*** (previously *@Ignore*) – used to disable a test class or method
- Some JUnit 5 annotations are **new** (did not exist in JUnit 4):
  - ***@RepeatedTest*** – allows specifying the *number of times to execute a test*
  - ***@TestFactory*** – supports *dynamic test* generated at runtime by a factory method
  - ***@DisplayName*** – defines custom display name for a test class or a test method
  - ***@Nested*** – denotes that the annotated class is a nested, non-static test class
  - ***@Tag*** – declares tags *for filtering tests* at the class or method level
  - ***@ExtendWith*** – it is used to *register custom extensions* (e.g. SpringExtension)

Eclipse 4.7 (*Oxygen*) has beta support for the JUnit Platform and JUnit Jupiter

Themis
Leaders in IT Education

Where Do I Go From Here?

Themis

Leaders in IT Education

Intermediate Java

Course Information

http://www.themisinc.com/CourseDetail.aspx?id=JA1011

Documentation on JUnit

http://junit.org/junit4/

http://junit.org/junit4/javadoc/latest/

http://junit.org/junit5/docs/current/user-guide/

Themis
Leaders in IT Education

Learning More About Java, OO and Database Interaction

http://www.themisinc.com/CatDetail.aspx?id=600&category=Java

- Java 1 (**Intro** Java) and Java 2 (**Intermediate** Java)
  - These courses spend two weeks in Java Standard Edition
  - Both courses are necessary to get a complete background
- Java **Enterprise** – Servlets and JSPs
  - One week course on these critical components of Java EE
- **Object Oriented Analysis and Design**
  - This course gives the budding OO developer a solid background in OO requirements specification and UML
- Many other courses depending on your environment
  - Enterprise Java Beans (EJBs); Java Persistence API (JPA); Java Messaging Service (JMS); Java Server Faces (JSF); Web Services with Java; Design Patterns; Spring; XML; HTML; Hibernate; Struts; many more

Themis
Leaders in IT Education

Thank you for coming

Themis

Leaders in IT Education

www.themisinc.com

US	1-800-756-3000
Intl.	1-908-233-8900

On-site and Public

Instructor-led

Hands-on Training

Hundreds of IT Courses

Customization Available