Unit testing with JUnit

COMP3111/H tutorial

A Sample Test Case

- For a calendar application, we may have the following test case:
  - The date is set to **11-Nov-2014**
  - The start time is set to **8:00am**
  - The end time is set to **8:15am**
  - The event title is set to **"COMP3111 Test Case 1"**
  - The event is a **one-time** event
- Expected result
  - The one-time event should be successfully created with an appropriate update on the user interface
  - The event should be saved correctly
  - The event should be loaded correctly

Without Automatic Unit Testing

- Based on the previous test case, we need to
  - Launch the calendar application
  - Through the user interface, type in all the values
  - Ensure that the event is successfully created
  - Save and Close
  - Re-launch the application
  - Show that the event is successfully loaded
- What happen if we have 10,000 cases and we need to complete all test cases daily?
  - Impossible!
  - If you can complete 1 test case per minute, you need 6.94 days to complete 10,000 cases

Unit Testing

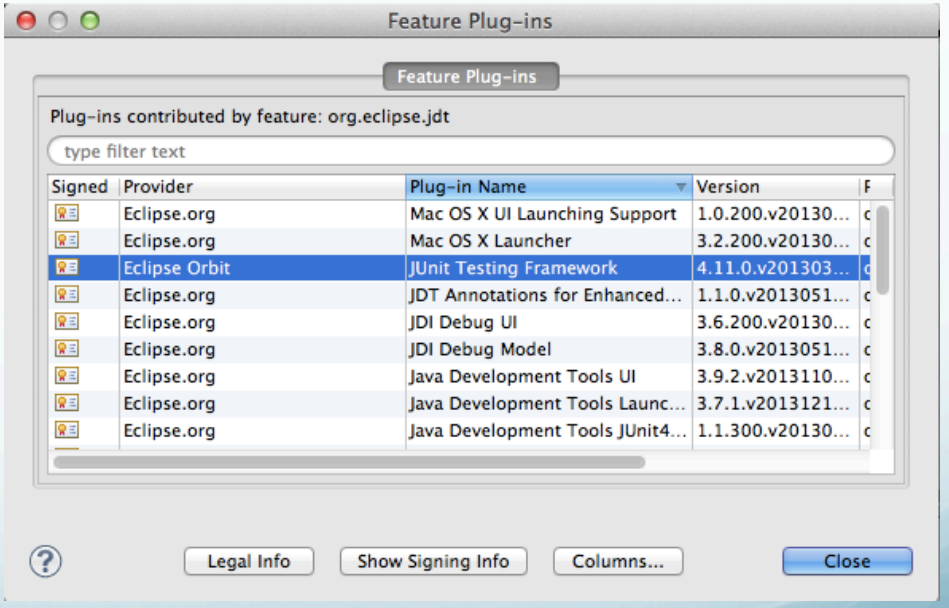
- Unit testing is a software verification and validation method in which a programmer tests if individual units of source code are fit for use
- What is a unit?
  - A unit is the smallest testable part of an application
- A unit test provides a strict, written contract that the piece of code must satisfy
- The goal of unit testing is to isolate each part of the program and show that the individual parts are correct

Automatic Unit Testing

- Tests must be executed automatically without any human participation
  - A tester may simply click the “Play” button
  - The tester will finally receive a report with a number of PASS/FAIL

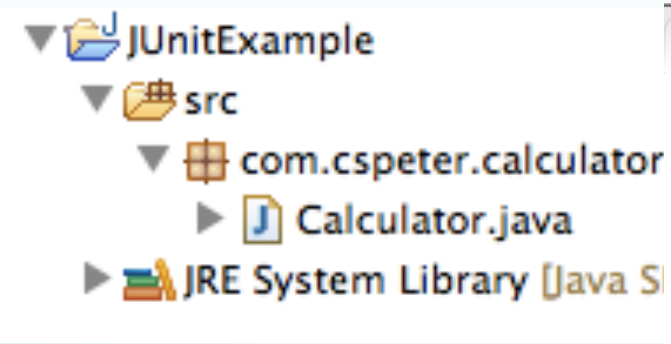
JUnit

- JUnit is a unit testing framework for the Java programming language
  - http://junit.org/index.html
- Eclipse includes JUnit



Example: Using JUnit in Eclipse

- An example project is created with an implementation of Calculator class in a package "com.cspeter.calculator"
- Problem: What are the steps to write a JUnit test?

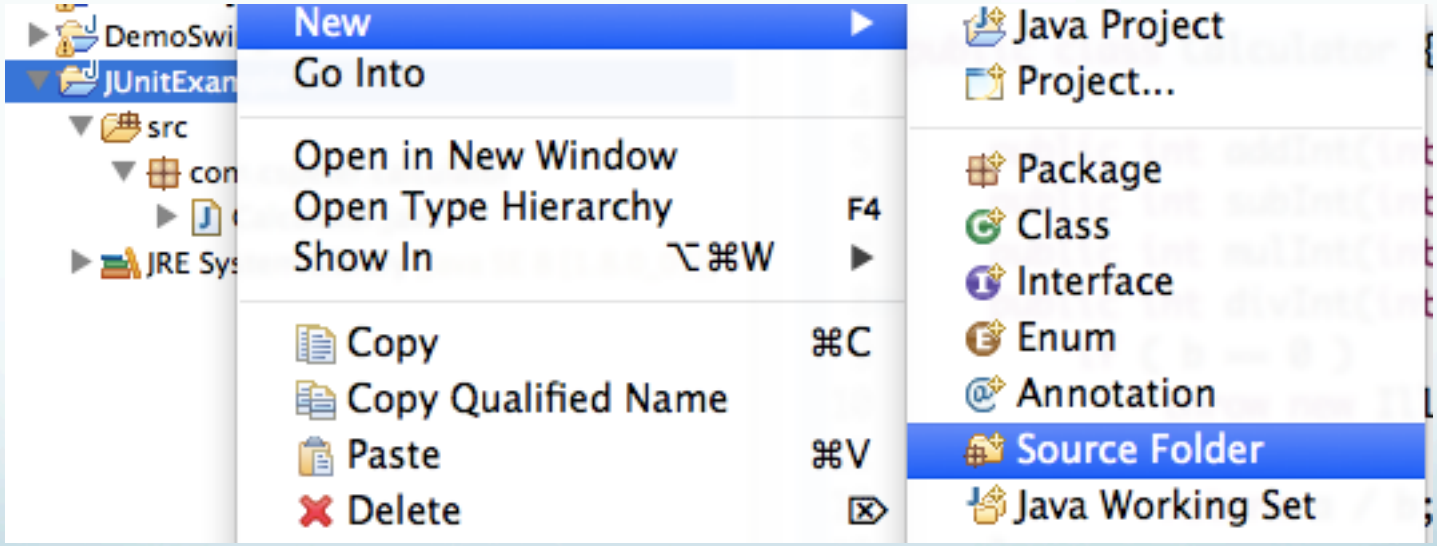


Calculator.java

```
1  package com.cspeter.calculator;
2
3  public class Calculator {
4
5      public int addInt(int a, int b) {return a + b;}
6      public int subInt(int a, int b) {return a - b;}
7      public int mulInt(int a, int b) {return a * b;}
8      public int divInt(int a, int b) throws IllegalArgumentException {
9          if ( b == 0 )
10             throw new IllegalArgumentException("b can't be zero");
11         else
12             return a / b;
13     }
14 }
```

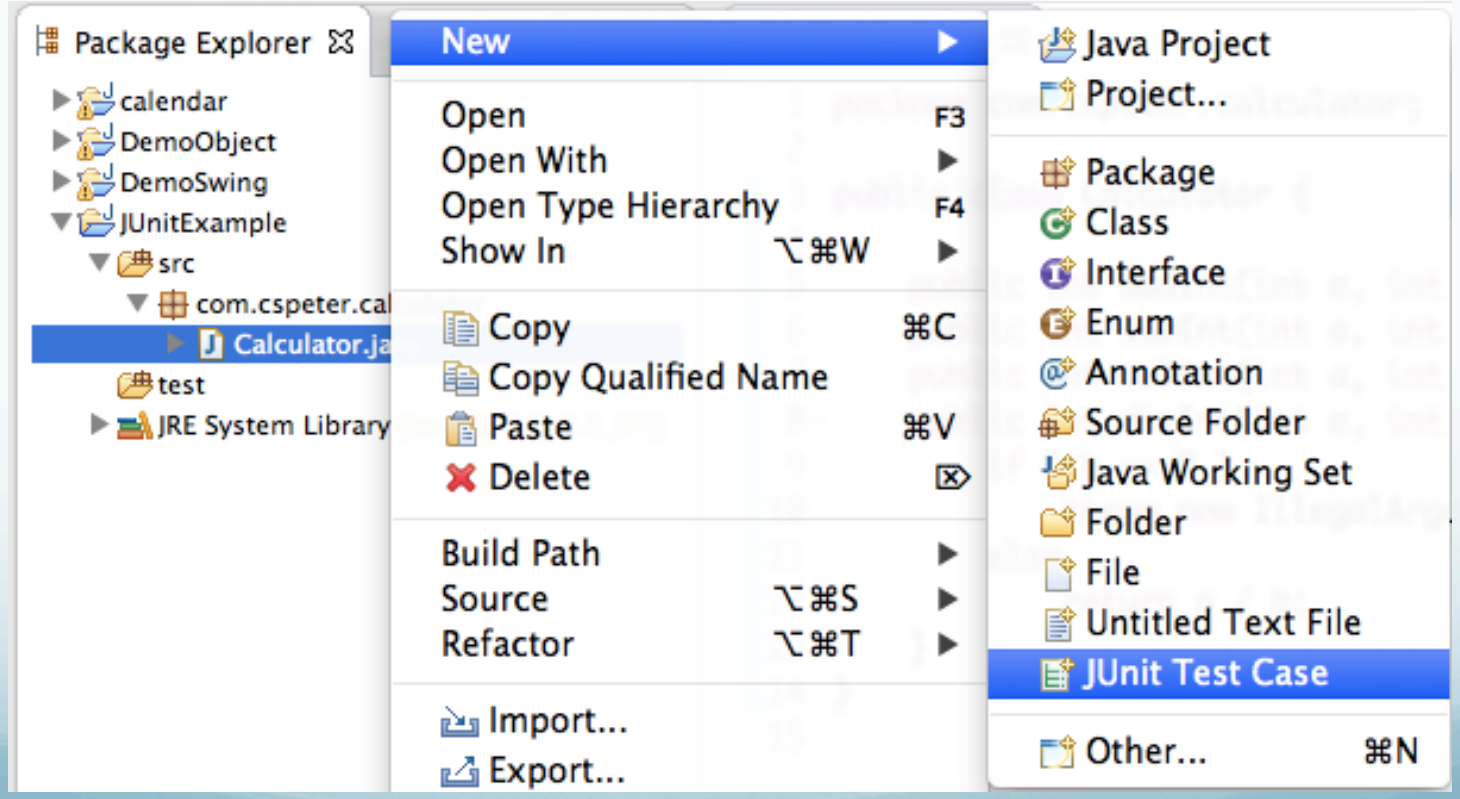
Example: JUnit (Step 1)

- Create a “source folder” named “test”
- It avoids mixing JUnit test cases with the source files



Example: JUnit (Step 2)

- Right-click the class you would like to test (i.e. Calculator) and choose “New > JUnit Test Case”



Example: JUnit (Step 3)

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: JUnitExample/test Browse...

Package: com.cspeter.calculator Browse...

Name: CalculatorTest

Superclass: java.lang.Object Browse...

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.cspeter.calculator.Calculator Browse...

< Back Next > Cancel Finish

Change to the “test” folder

Name the test case (e.g. [className]Test)

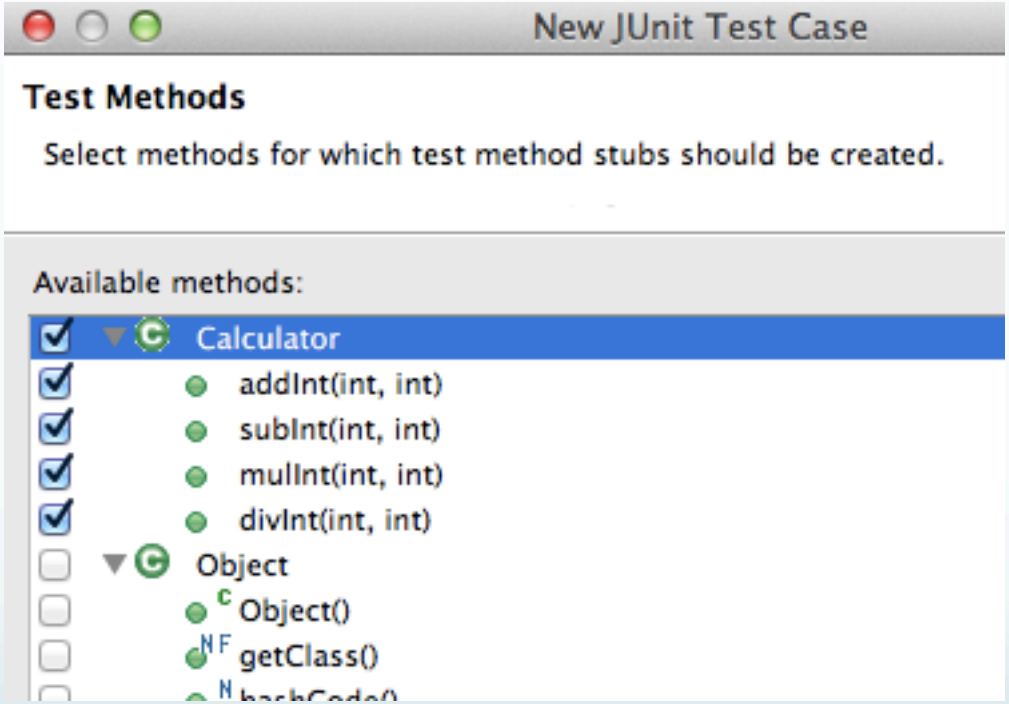
[Optional]
Generate method stubs

Search for the class under test

Click “Next >”

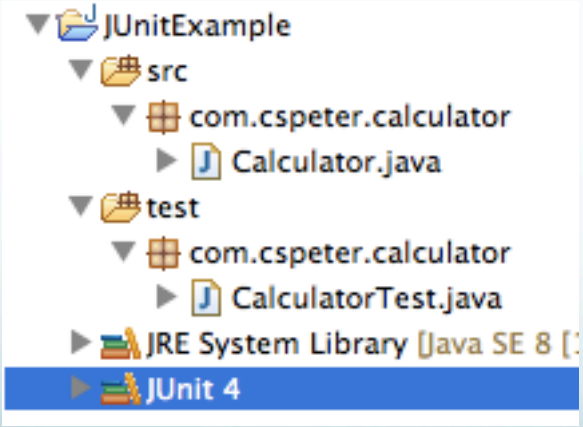
Example: JUnit (Step 4)

- Generate the prototypes of the test cases



Example: JUnit (Step 5)

- A default test case "CalculatorTest" will be generated
- The test case class will be separated from the source folder



*CalculatorTest.java

```
1  package com.cspeter.calculator;
2  import static org.junit.Assert.*;
6  public class CalculatorTest {
7      @Before
8      public void setUp() throws Exception {
9      }
10     @After
11     public void tearDown() throws Exception {
12     }
13     @Test
14     public void testAddInt() {
15         fail("Not yet implemented");
16     }
17     @Test
18     public void testSubInt() {
19         fail("Not yet implemented");
20     }
21     @Test
22     public void testMulInt() {
23         fail("Not yet implemented");
24     }
25     @Test
26     public void testDivInt() {
27         fail("Not yet implemented");
28     }
29 }
```

Writing a test case (Step 1)

- The setUp() method is very useful if you would like to create a single object to be shared among different testing methods in the same test case
- For example, there are 4 testing methods (i.e. testAddInt, testSubInt, testMulInt and testDivInt) and all methods should share the same Calculator object (i.e. testCal)

```
public class CalculatorTest {

    private Calculator testCal;
    @Before
    public void setUp() throws Exception {
        testCal = new Calculator();
    }
```

Assert Methods

- Assert methods are used to complete a test case
- Assert methods include:
  - assertEquals(x,y)
  - assertFalse(boolean)
  - assertTrue(boolean)
  - assertNull(object)
  - assertNotNull(object)
  - assetSame(firstObject, secondObject)
  - assertNotSame(firstObject, secondObject)
  - …
  - Full List: http://junit.sourceforge.net/javadoc/org/junit/Assert.html

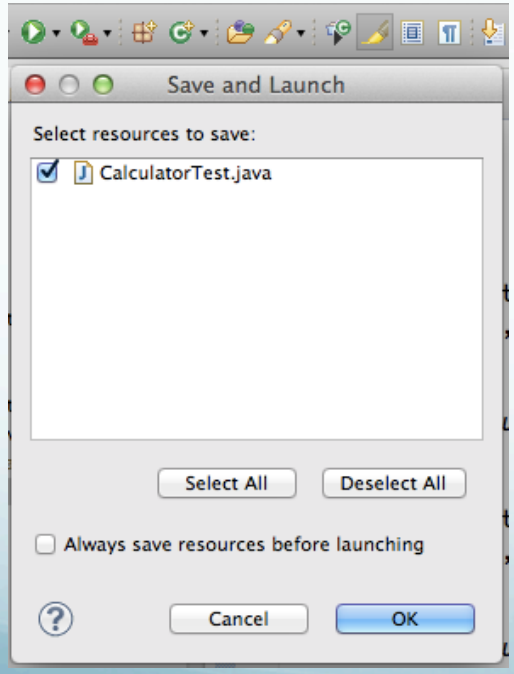
Writing a test case (Step 2)

- Implement all the testing methods using assert methods
- For example:
  - 1 + 0 = 1
  - 0 + 0 = 0
  - 123 + (-123) = 0
- Other assert methods can also be used

```java
@Test
public void testAddInt() {
    int a, b, result, expected;

    a = 1; b = 0; expected = 1;
    result = testCal.addInt(a, b);
    assertEquals(result, expected);

    a = 0; b = 0; expected = 0;
    result = testCal.addInt(a, b);
    assertEquals(result, expected);

    a = 123; b = -123; expected = 0;
    result = testCal.addInt(a, b);
    assertEquals(result, expected);
}
```

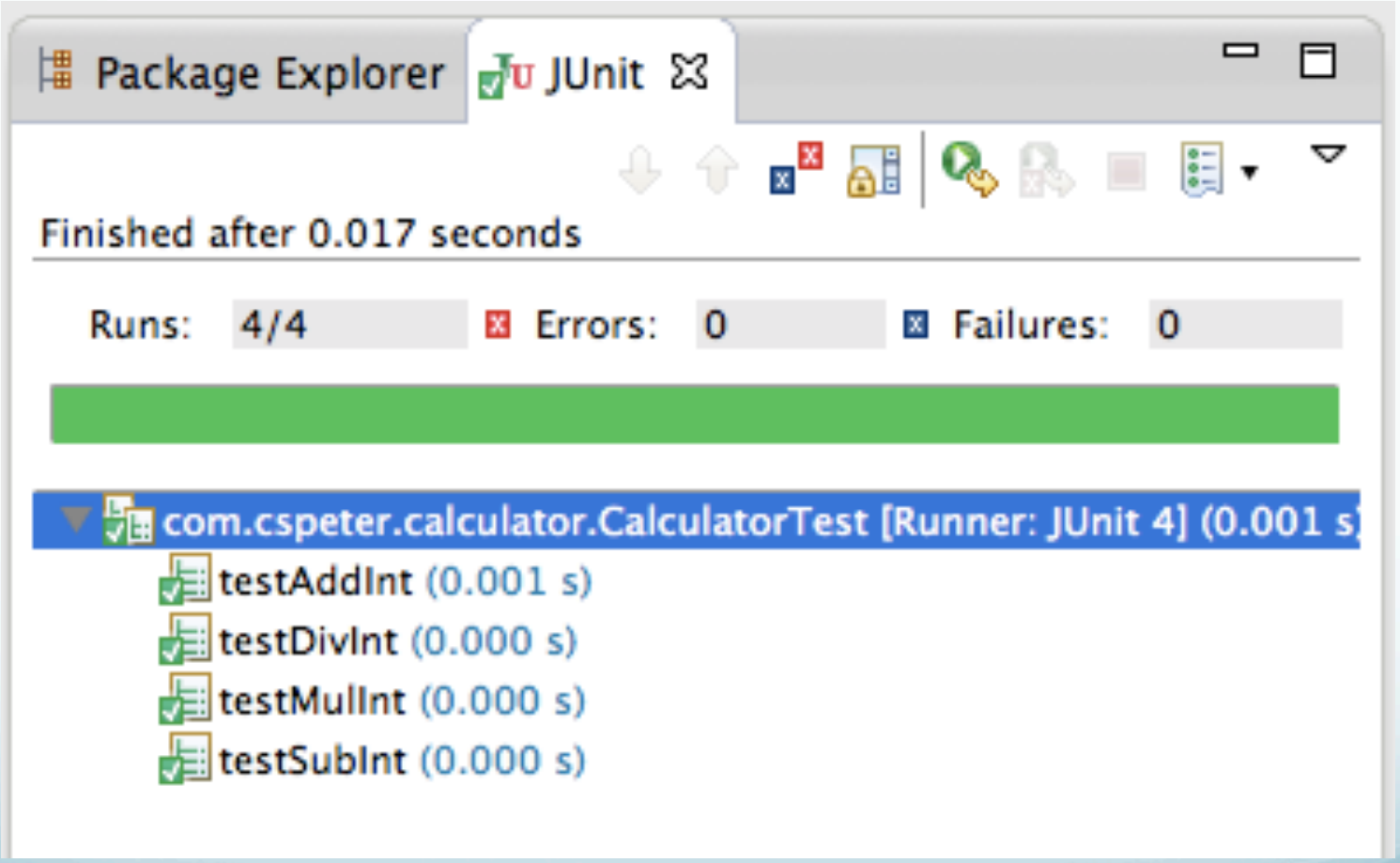
Writing a test case (Step 3)

- Implementing the remaining testing methods
- Click the “Play” button to launch a JUnit test




```
@Test
public void testSubInt() {
    int a, b, result, expected;
    a = 1; b = 0; expected = 1;
    result = testCal.subInt(a, b);
    assertEquals(result, expected);
}
@Test
public void testMulInt() {
    int a, b, result, expected;
    a = 1; b = 0; expected = 0;
    result = testCal.mulInt(a, b);
    assertEquals(result, expected);
}
@Test
public void testDivInt() {
    int a, b, result, expected;
    a = 0; b = 1; expected = 0;
    result = testCal.divInt(a, b);
    assertEquals(result, expected);
}
```

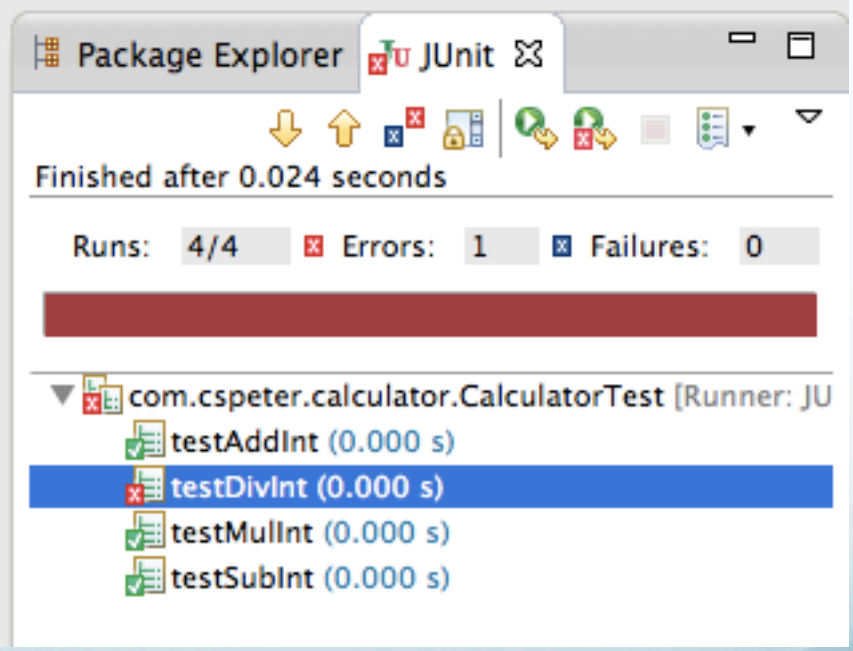
Result


Package Explorer
JUnit
Finished after 0.017 seconds
Runs: 4/4
Errors: 0
Failures: 0
com.cspeter.calculator.CalculatorTest [Runner: JUnit 4] (0.001 s)
testAddInt (0.001 s)
testDivInt (0.000 s)
testMulInt (0.000 s)
testSubInt (0.000 s)

What happens if an Exception occurs?

- Change testIntDiv() as follows and re-run the test
- An error is shown on the JUnit report

```
@Test
public void testDivInt() {
    int a, b, result, expected;
    a = 1; b = 0; expected = 0;
    result = testCal.divInt(a, b);
    assertEquals(result, expected);
}
```

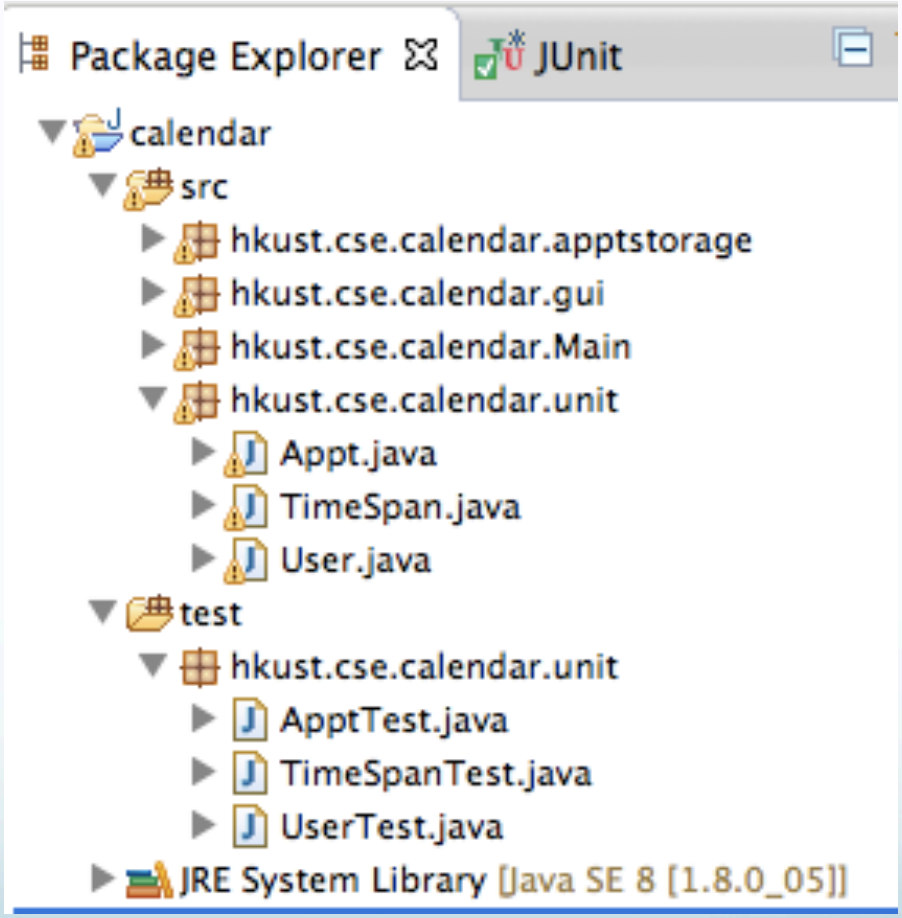


Writing a test case (Step 4)

- testIntDiv() should better be revised as follows:

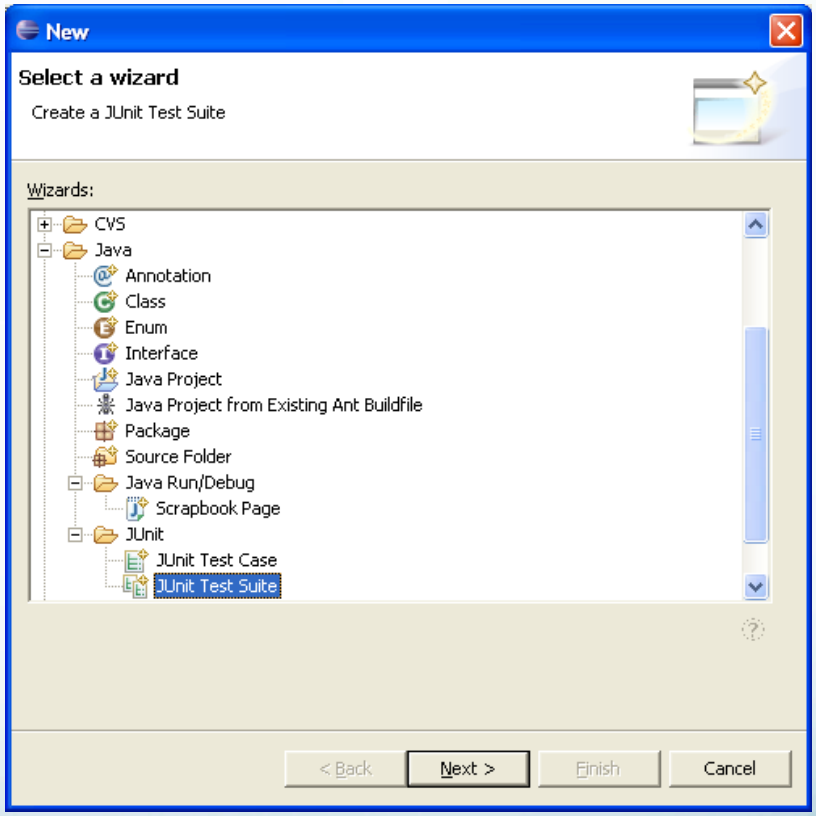
```
@Test
public void testDivInt() {
    int a, b, result, expected;

    a = 5; b = 2; expected = 2;
        result = testCal.divInt(a, b);
        assertEquals(result, expected);

    a = 5; b = 0;
    try {
        result = testCal.divInt(a, b);
        fail("This line should not be executed!");
    } catch (IllegalArgumentException e) {
        assertTrue(e.toString(), true);
    }
}
```

What happen if I have a number of test cases?



JUnit TestSuite

- Usually, a JUnit test case should only test one class
- JUnit TestSuite
  - Run multiple test cases or suites
- To create a TestSuite
  - Select your testing package
  - From the context menu select New > Other… > Java > Junit
- Select JUnit Test Suite



JUnit TestSuite

- You can run multiple test cases in a single TestSuite