**Hands On Lab 1  
[Test Cases and Test Fixtures](https://personales.unican.es/corcuerp/java/Labs/LAB_23.htm" \l "Exercise_1)**

### Eclipse

Many prefer to execute their tests from the convenience of an automated integrated development environment such as Eclipse.  Fortunately, Eclipse includes jUnit as a part of the standard Eclipse Java Development Tools package and provides a clean frontend to defining and running tests.  To see Eclipse's jUnit capabilities in action, create a new Java project and add the Philadelphia class.

**public** **class** Philadelphia {

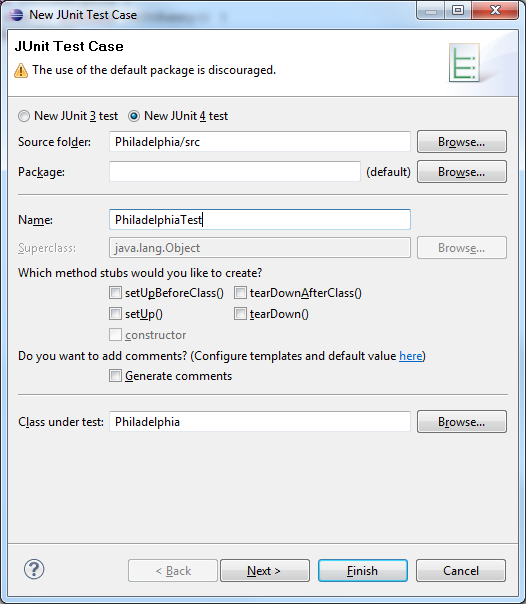
**public** **static** **boolean** isItSunny() {

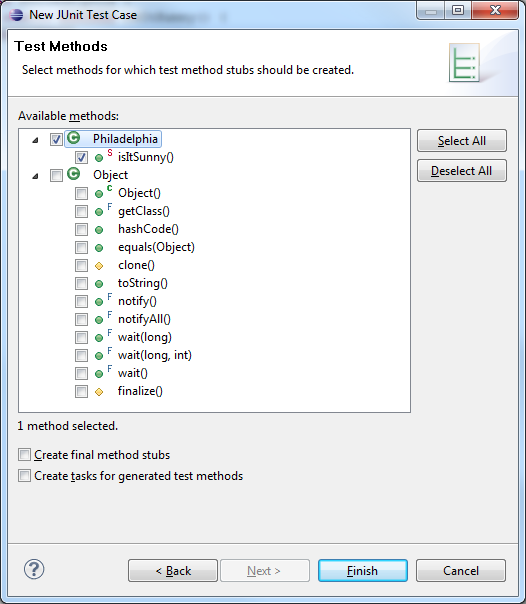
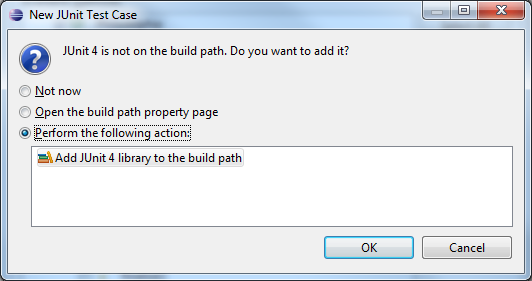
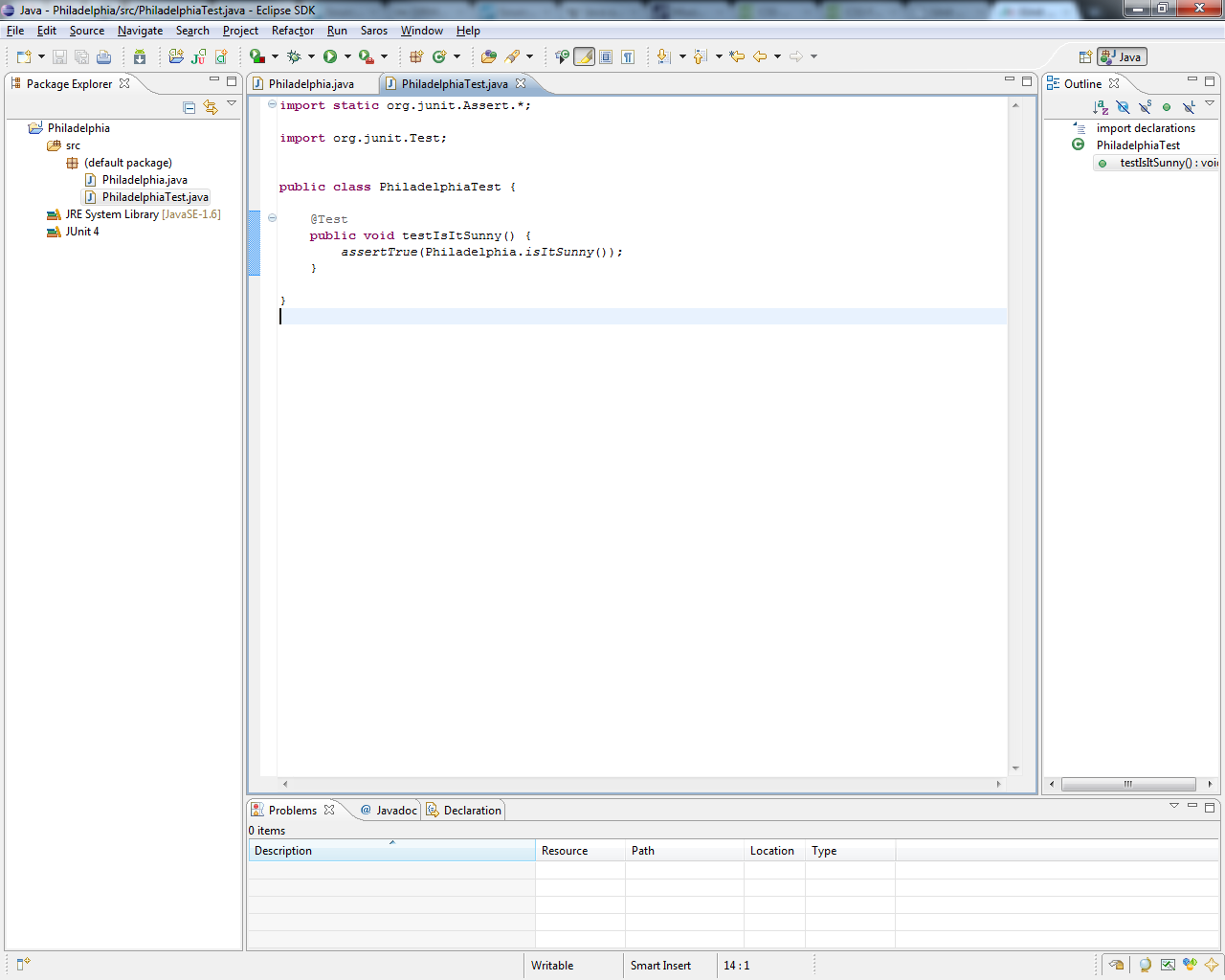
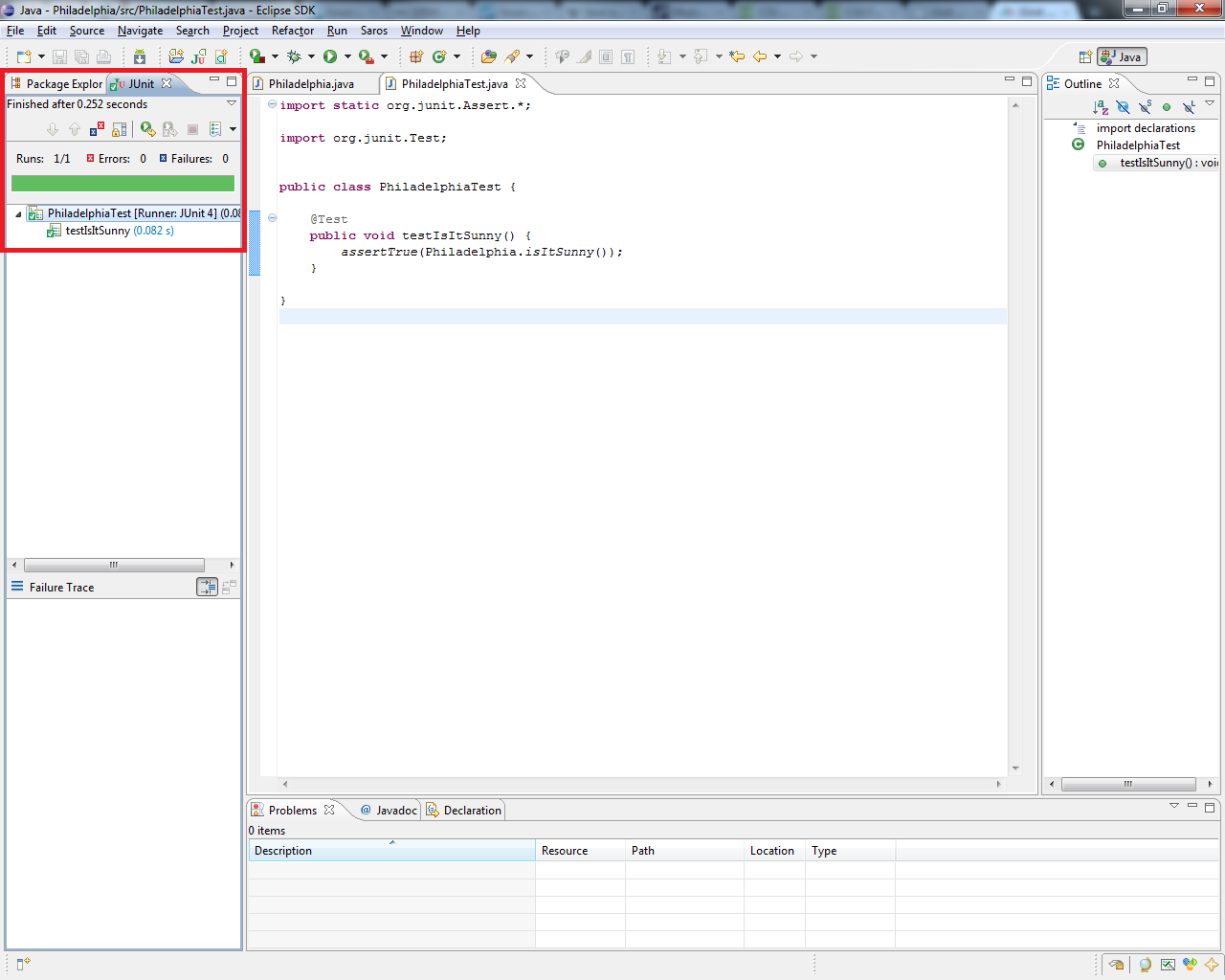
**return** **true**;

}}

Now to define the test class use Eclipse's "File -> New -> JUnit Test Case" menu option.  This will open a "New JUnit Test Case" dialog as seen below.  We want the following options:

* Select "New JUnit 4 test"
* Name: PhiladelphiaTest
* Class under test: Philadelphia



Click "Next" and you'll be given the option to select the methods from the Philadelphia test that should be tested.  We'll be testing the "isItSunny" method, so select the checkbox next to that method as seen below:  
  
Click Finish, and Eclipse will create a new Test class for you.  (Note: the first time you define a test in a project Eclipse will inform you that jUnit is not on the build path and ask if you would like it to be added.  The jUnit library is required to run the tests, so we do indeed want it to be added to the build path).  
  
All that remains is to actually define the test from above.  In the PhiladelphiaTest class that Eclipse created and opened for you, replace the call to "fail" with the original test statement as seen below:  
  
Now all that remains is to run the test.  Eclipse provides a special run configuration for jUnit testing.  To use it, select the "Run -> Run As -> JUnit Test" menu item.  This will open the "JUnit" view that presents the testing progress and results.  


**Exercise**Perform the following tasks to become familiar with jUnit basics (you may use either the command line or Eclipse for this exercise):

* Run the test described above and verify that you get the same output.
* Add an additional test which fails an assertion (this can be as simple as a test containing only an "assertTrue(false);" statement) and run the tests again.  Describe what has changed in the test results.
* Add an additional test that throws an exception and run the tests again.  Describe what has changed in the test results.

## Test Fixtures

Often when testing you'll need to do the same basic set-up or tear-down activities for several tests.  jUnit provides an easy way to aggregate tests and handle the set-up and tear-down activities in separate methods that get called before and after running each test.  To use this capability, you need only group such tests into the same containing class and annotate the set-up and tear-down methods with "org.junit.Before" and "org.junit.After" respectively.  For example, the following class represents a very simple test fixture that exercises some of the methods found in the "java.util.ArrayList" class:

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

import java.util.ArrayList;import java.util.Arrays;

import java.util.List;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

**public** **class** ArrayListTest {

List<Integer> testArray;

/\*\* \* This method is invoked before each test is run to set up the test array \* with a known set of values. \*/

@Before

// Informs JUnit that this method should be run before each test

**public** **void** setUp() {

testArray = **new** ArrayList<Integer>(Arrays.asList(3, 1, 4, 1, 5));

}

/\*\* \* This method is invoked after each test is run to perform tear down \* activities (not needed in this test fixture). \*/

@After

// Informs JUnit that this method should be run after each test

**public** **void** tearDown() {

// No tear down needed for this test

}

/\*\* \* Adds a value to the array and verifies the add was successful. \*/

@Test

**public** **void** testAdd() {

testArray.add(9);

List<Integer> expected =

**new** ArrayList<Integer>(Arrays.asList(3, 1, 4, 1, 5, 9));

assertEquals(testArray, expected);

}

/\*\* \* Removes a value from the array and verifies the remove was successful. \*/

@Test

**public** **void** testRemoveObject() {

testArray.remove(**new** Integer(5));

List<Integer> expected =

**new** ArrayList<Integer>(Arrays.asList(3, 1, 4, 1));

assertEquals(testArray, expected);

}

/\*\* \* Tests the indexOf method and verifies the expected return value. \*/

@Test

**public** **void** testIndexOf() {

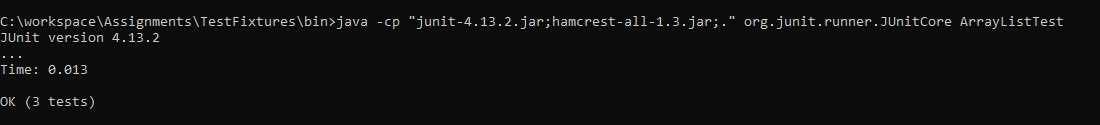
assertEquals(testArray.indexOf(4), 2);

}

}

Each of the tests above works on the same pre-initialized test array named "testArray".  Also, it's important to note that each jUnit test is run independently: actions taken in one test are not seen in any other test.  In other words, the set-up and tear-down are run for each test. Run the tests in this class from the command line. Junit4.jar should be in the same directory or the classpath. Pick it up from the Supporting Materials/commandline folder for Day 5.

Upon running the tests in this class, we'll get the following output:



**Exercise**  
Perform the following tasks to become familiar with jUnit basics:

* Run the test fixture described above and verify that you get the same output.
* Add an additional test which uses the testArray, tests the "clear" method, and verifies that the array is empty.
* Add an additional test which uses the testArray and tests the "contains" method by verifying it returns true when supplied a value that exists in the array.
* Add an additional test which uses the testArray and tests the "contains" method by verifying it returns false when supplied a value that does not exist in the array.
* Add an additional test which uses the testArray and tests the "get" method verifying it returns the correct value for a given index.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*