JUnit testing

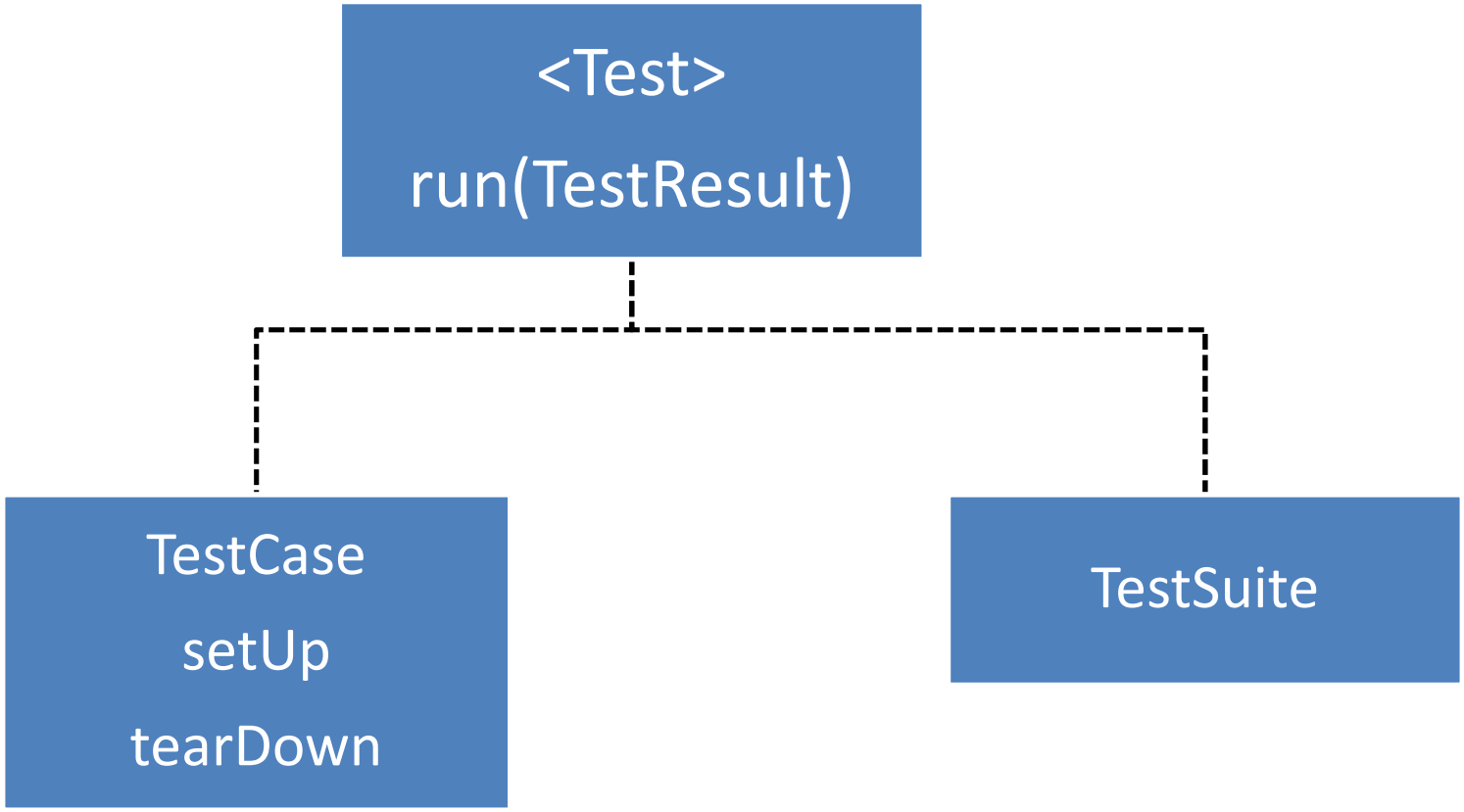
Contents:

JUnit – the Java's Unit Testing Framework

:: Introduction

- In Java, the standard unit testing framework is known as **JUnit**.
- Test Cases and Test Results are Java objects.
- JUnit was created by Erich Gamma and Kent Beck, two authors best known for Design Patterns and eXtreme Programming, respectively.
- Using JUnit you can easily and incrementally build a **test suite** that will help you **measure your progress**, **spot unintended side effects**, and focus your development efforts.

JUnit UML diagram



JUnit – the Java's Unit Testing Framework

:: Key JUnit Notions

- ***Tested Class*** – the class that is being tested.
- ***Tested Method*** – the method that is tested.
- ***Test Case*** – the testing of a class's method against some specified conditions.
- ***Test Case Class*** – a class performing the test cases.
- ***Test Case Method*** – a Test Case Class's method implementing a test case.
- ***Test Suite*** – a collection of test cases that can be tested in a single batch.

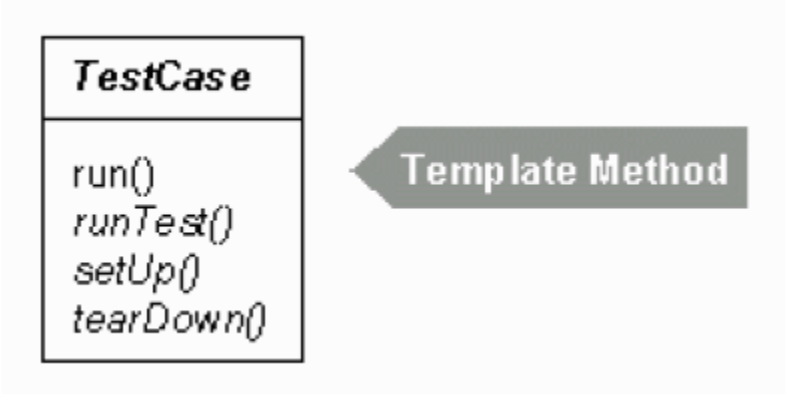
JUnit – the Java's Unit Testing Framework

:: TestCase Class

The class TestCase has four important methods – **run(), setUp(), tearDown()** and **runTest().**

TestCase.run() applies **Template Method** pattern

```
public void run(){
        setUp();
        runTest();
        tearDown();
}
```



The Template Method pattern "defines the skeleton of an algorithm in an operation, deferring some steps to subclasses".

All Test Case classes need to be subclasses to the TestCase class.

Assertions

- Assertions are defined in the JUnit class **`Assert`**
  - If an assertion is true, the method continues executing.
  - If any assertion is false, the method stops executing at that point, and the result for the test case will be fail.
  - If any other exception is thrown during the method, the result for the test case will be error
  - If no assertions were violated for the entire method, the test case will pass.

- All assertion methods are static methods

JUnit Assert class

Method name	Description
assertEquals	Asserts that two objects or primitives are equal. Compares objects using equals method, and compares primitives using == operator.
assertFalse	Asserts that a Boolean condition is false.
assertNotNull	Asserts that an object is not null
assertNotSame	Asserts that two objects do not refer the same object. Compares objects using != operator
assertNull	Asserts that an object is null.
assertSame	Asserts that two objects refer to the same object. Compares objects using == operator.
assertTrue	Asserts that a boolean condition is true.
fail	Fails the test.

JUnit – the Java's Unit Testing Framework

:: Basics - JUnit 3

1. Create the class that you want to test.
2. Build the test class - with the needed imports and extensions for JUnit.
   - Extend this class from junit.framework.TestCase.
   - Name all the test methods with a prefix of 'test'.
3. Code the actual test cases.
   - Validate conditions and invariants using one of the several assert methods.

```
import junit.framework.*;
public class TestFailure extends TestCase {

  public void testSquareRootException() {
    try {
      SquareRoot.sqrt(-4, 1);
      fail("Should raise an exception");
    }
    catch (Exception success) { … }
  }
}
```

Test Case Class

Test Case Method

Tested Class and Method

Assertion Statement

JUnit – the Java's Unit Testing Framework

:: Basics – JUnit 4

- Tests are identified by an **@Test** annotation and we no longer need to prefix our test methods with "test".
- This lets us follow the naming convention that best fits our application.

```
import junit.framework.*;
import org.junit.Test;
public class TestAddition extends TestCase {

        private int x = 1;
        private int y = 1;
        @Test public void addition()
        {
                int z = x + y;
                assertEquals(2, z);
        }
}
```

Annotations

- The @Ignore annotation says to not run a test

```
@Ignore("I don't want Dave to know this doesn't work")
@Test
public void add() {
    assertEquals(4, program.sum(2, 2));
}
```

- You shouldn't use @Ignore without a very good reason!

JUnit – the Java's Unit Testing Framework

:: TestCase Class – SetUp

- JUnit test runners automatically invoke the **setUp()** method before running each Test Class.
- This method typically initializes fields, turns on logging, resets environment variables, and so forth, i.e. it sets up a **context** for the test cases to be applied.

```
protected void setUp()
{
        System.out.println("Before testing");
}
```

- In JUnit 4, the initialization method no longer needs to be called setUp().
- It just needs to be denoted with the **@Before** annotation.
- We can have multiple methods noted @Before, each running before testing.

```
@Before protected void initialize()
{
        System.out.println("Before testing");
}
```

JUnit – the Java's Unit Testing Framework

:: TestCase Class – TearDown

- If we need at the end of each test to do a cleanup operation, we can use JUnit's ***tearDown()*** method. For example we can call the garbage collector there in case our tests consume large amount of memory.

```
protected void tearDown()
{
        System.out.println("After testing");
        System.gc();
}
```

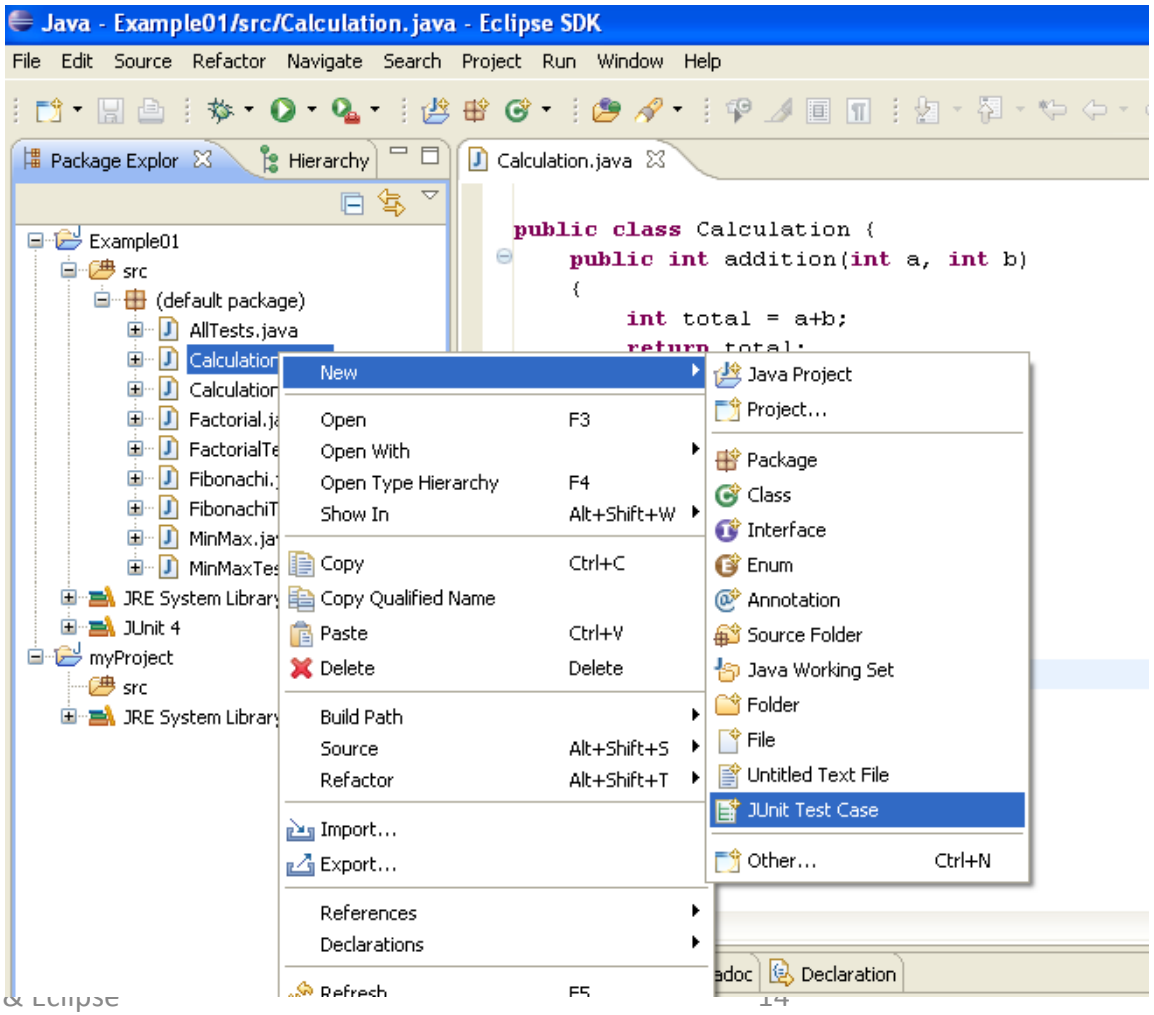
- In JUnit 4, we can give it a more natural name and annotate it with **@After**.

```
@After protected void disposeObjects ()
{
        System.out.println("After testing");
        System.gc();
}
```

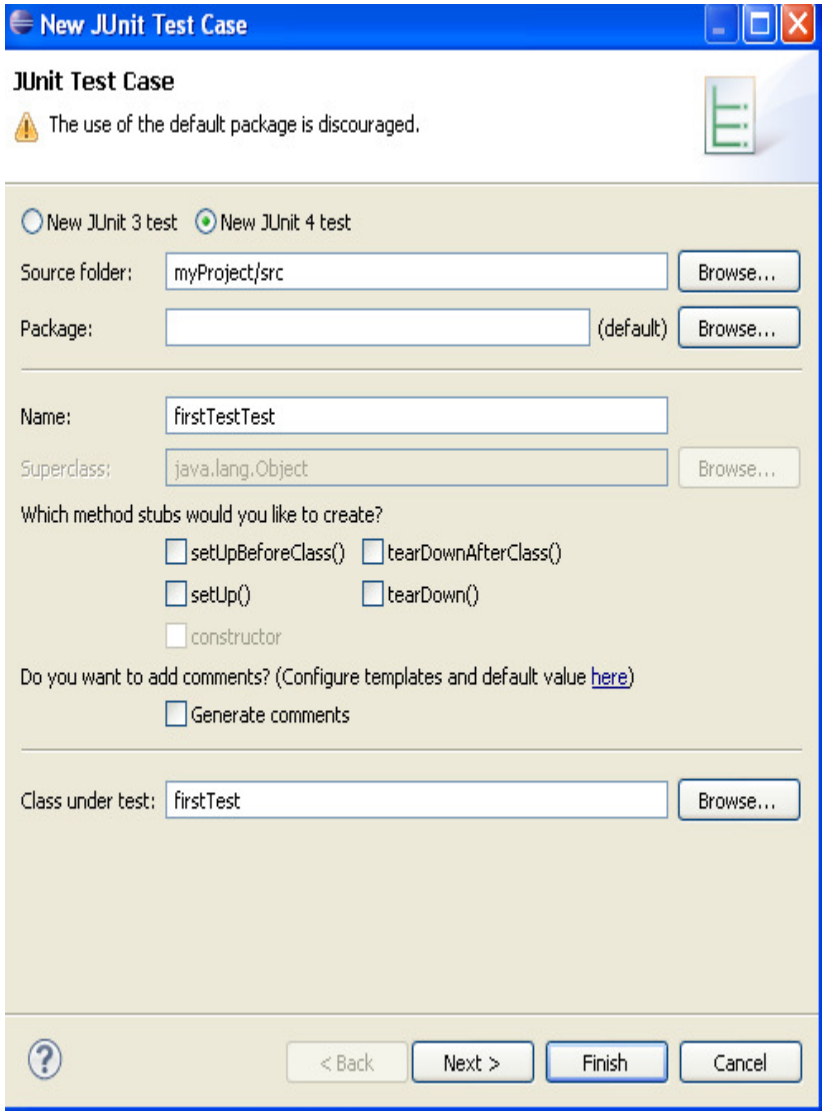
Unit Testing in Eclipse using JUnit

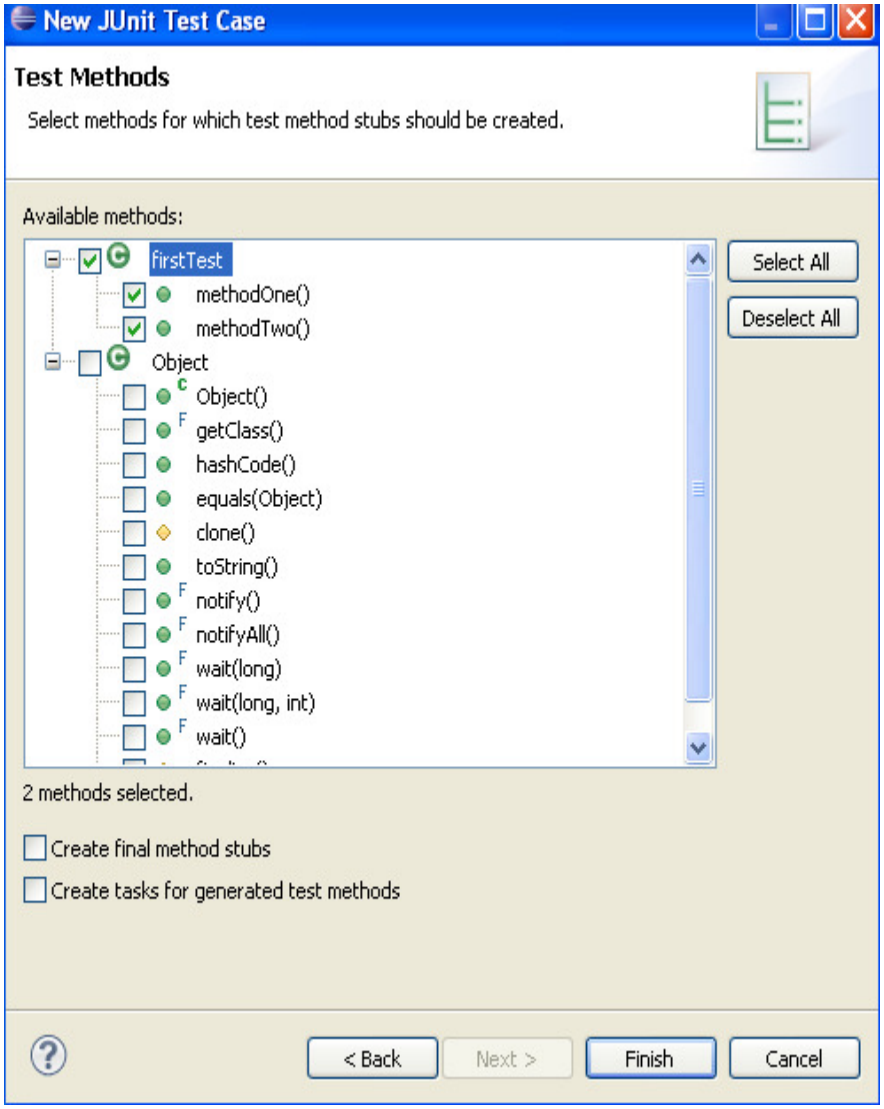
:: Adding a Test Case to the Project

- Once the class we want to test, is created we can start with building the test cases.
- To create a test case do



:: Create your Test Case



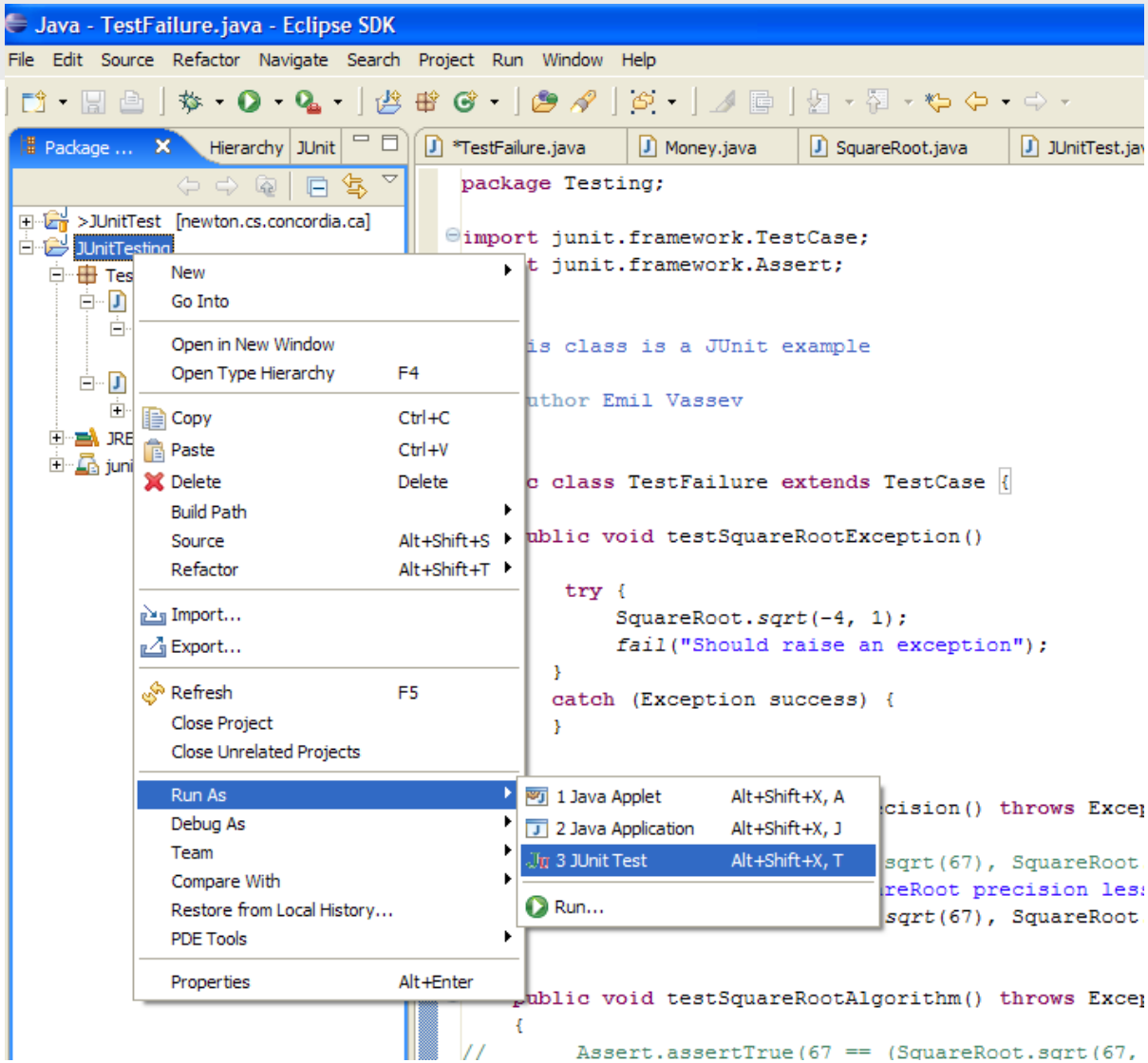


Unit Testing in Eclipse using JUnit

:: Running the Test

From Menu bar [Run → Run As → JUnit Test],

or:



Unit Testing in Eclipse using JUnit

:: Test Result Analysis

Java - TestFailure.java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Package ... | Hierarchy | JUnit

Finished after 0,06 seconds

Runs: 3/3 Errors: 0 Failures: 2

testSquareRootPrecision - Testing.TestFail
testSquareRootAlgorithm - Testing.TestFai

These two test case methods reported failures

Failure Trace

junit.framework.AssertionFailedError: SquareR
at Testing.TestFailure.testSquareRootPrecisio
at sun.reflect.NativeMethodAccessorImpl.invo
at sun.reflect.NativeMethodAccessorImpl.invo
at sun.reflect.DelegatingMethodAccessorImpl.

*TestFailure.java | Money.java | SquareRoot.java | JUnitTest.java | SquareRoot.java | TestFailure.java

```
package Testing;

import junit.framework.TestCase;
import junit.framework.Assert;

/**
 * This class is a JUnit example
 *
 * @author Emil Vassev
 *
 */
public class TestFailure extends TestCase {

    public void testSquareRootException()
    {
        try {
            SquareRoot.sqrt(-4, 1);
            fail("Should raise an exception");
        }
        catch (Exception success) {
        }
    }

    public void testSquareRootPrecision() throws Exception
    {
//      Assert.assertEquals(Math.sqrt(67), SquareRoot.sqrt(67, 5), 0.000001);
        Assert.assertEquals("SquareRoot precision less than 0.000001",
                            Math.sqrt(67), SquareRoot.sqrt(67, 5), 0.000001);
    }

    public void testSquareRootAlgorithm() throws Exception
    {
//      Assert.assertTrue(67 == (SquareRoot.sqrt(67, 10) *  SquareRoot.sqrt(67, 10)));
        Assert.assertTrue(67 == (SquareRoot.sqrt(67, 6) *  SquareRoot.sqrt(67, 6)));
    }

}
```

Unit Testing in Eclipse using JUnit

:: Rerunning the Test

Package ... | Hierarchy | JUnit

Finished after 0,02 seconds

Runs: 3/3 Errors: 0 Failures: 0

All the test case methods passed the test.

Failure Trace

*TestFailure.java | Money.java | SquareRoot.java | JUnitTest.java | SquareRoot.java | TestFailure.java

```java
package Testing;

import junit.framework.TestCase;
import junit.framework.Assert;

/**
 * This class is a JUnit example
 *
 * @author Emil Vassev
 *
 */
public class TestFailure extends TestCase {

    public void testSquareRootException()
    {
        try {
            SquareRoot.sqrt(-4, 1);
            fail("Should raise an exception");
        }
        catch (Exception success) {
        }
    }

    public void testSquareRootPrecision() throws Exception
    {
//      Assert.assertEquals(Math.sqrt(67), SquareRoot.sqrt(67, 5), 0.000001);
        Assert.assertEquals("SquareRoot precision less than 0.000001",
                            Math.sqrt(67), SquareRoot.sqrt(67, 6), 0.000001);
    }

    public void testSquareRootAlgorithm() throws Exception
    {
//       Assert.assertTrue(67 == (SquareRoot.sqrt(67, 6) *  SquareRoot.sqrt(67, 6)));
        Assert.assertTrue(67 == (SquareRoot.sqrt(67, 7) *  SquareRoot.sqrt(67, 7)));
    }

}
```

Exercise 01 – Familiar with Assertions & Annotations

1. Create a package named myTestPackage

a) Write class Calculations create test class

b) Write code for test addition and division without using Assertions.

c) Write code for test addition and division with using Assertions.

d) Write a code for class Factorial and for test Factorial with using Assertions.

2. Under the myTestPackage

a) Write code for class MinMax and test minmax with using Assertions.

b) Get error out put and evaluate

Home Work

a) Write code for class Fibbonachi and test with using Assertions.

b) Get error out put and evaluate

Test Suite

:: Test Suit - Introduction

- We have performed tests on only one class, i.e. we have tested methods under the consideration they belong to the same class.

- In large projects we have many classes with methods that should be tested.

- For testing multiple classes Eclipse and JUnit expose the concept of ***Test Suit***.

- A Test Suit is a collection of test cases that can be tested in a **single batch**.

- A Test Suite is a simple way of running one program that, in turn, runs all test cases.

Unit Testing in Eclipse using JUnit

:: Creating a Test Suit

There are four ways to create a JUnit Test Suite Class. First, select the directory (usually unittests/) that you wish to create the test suite class in.

- Select [File → New → Other... → Java → JUnit → JUnit Test Suite].
- Select the arrow of the button in the upper left of the toolbar. Select [Other... → Java → JUnit → JUnit Test Suite].

- Right click on a package in the Package Explorer view in the Java Perspective, and select [Other... → Java → JUnit → JUnit Test Suite].
- You can create a normal Java class, but import the package ***junit.framework*** and extend the TestSuite class.

Unit Testing in Eclipse using JUnit

:: Running All Tests

Right click on the test suite class and select [**Run As → JUnit Test**]

Java - AllTests.java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Package ... | Hierarchy | JUnit

Finished after 0,05 seconds

Runs: 8/8 Errors: 0 Failures: 0

Failure Trace

*TestFailure.java | Money.java | SquareRoot.java | TestFailure.java | TestMoney.java | Money.java | AllTests.java

```
package Testing;

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

public class AllTests {

    public static Test suite() {
        TestSuite suite = new TestSuite("Test for Testing");
        //$JUnit-BEGIN$
        suite.addTestSuite(TestMoney.class);
        suite.addTestSuite(TestFailure.class);
        //$JUnit-END$
        return suite;
    }

}
```

Exercise 02 – Familiar with Test Suite

1. Use the package myTestPackage

a) Create the test suite for test addition, division and Factorial.

b) Create the test suite for test Factorial and MinMax.

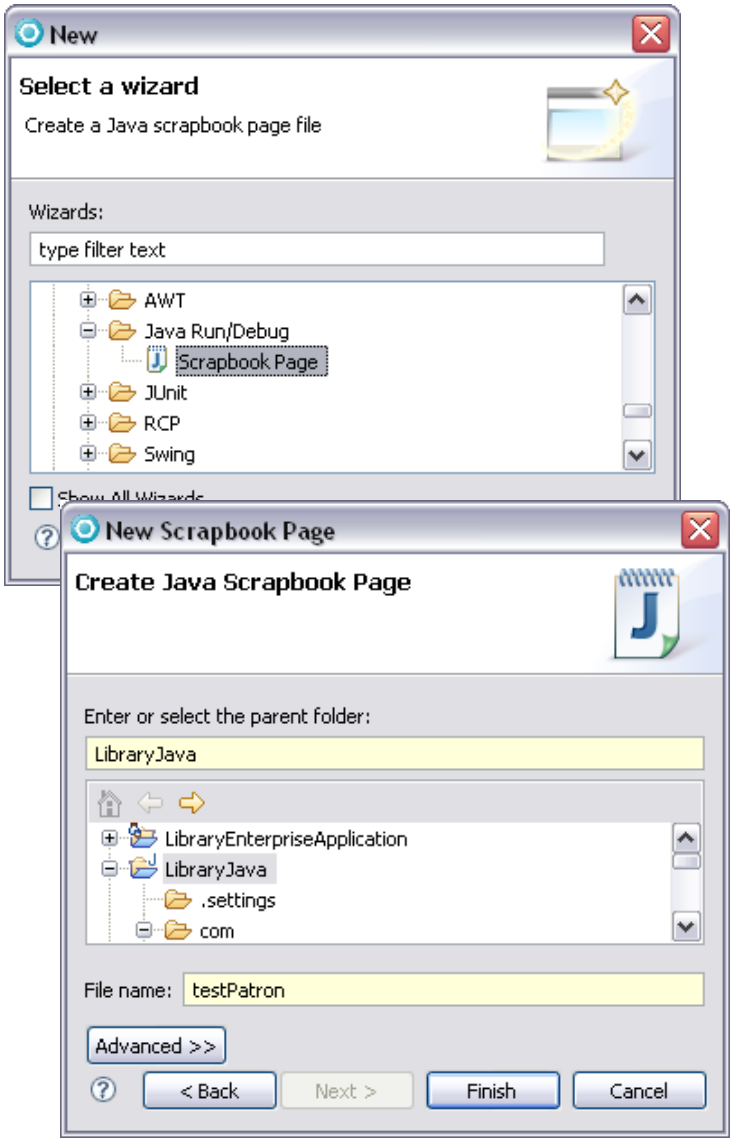
Home Work

1. Create Package named myAllTests
2. Create the test suite for test addition, division , Factorial , MinMax and Fibonacci.

Scrap Book

- Scrapbook pages allow you to evaluate and test a small section of code without creating an entire class.
  - To evaluate a code fragment in the scrapbook page, highlight an expression and select **Display** from the pop-up menu.
  - Add other classes or packages using **Set Imports**

Create a New Java Scrapbook Page



- Start the New Java Scrapbook wizard.
  - In the Java perspective, select **File** >**New** >**Other**.
  - Expand Java -> Java Run/Debug and select Scrapbook Page.
  - Click **Next**.
- Specify the name and location for the new Java scrapbook page.
  - Specify a location for the scrapbook page.
  - Choose a **name** for the scrapbook page.
  - Click **Finish**.
  - Select the statement to test.
  - Right click and execute.

Exercise 03 – Familiar with Scrap Book

1. Create a scrap book named myBook

a) write a test for print “Hello world” for get pass out put

b) write a code snippet for print “Hello world” for get error out put

c) write a code snippet for get the addition of two numbers and print the result.

Home Work

1. Write the code snippets for test addition, division , Factorial , MinMax and Fibonacci using scrap book.

Session expectations :

- Work with jUnit testing using:
  - Assertions
  - Annotations
  - Test case
  - Test Suite
  - Scrap Book

- Analyze the test results of jUnit testing

- Use Scrap Book to generate error free code snippets