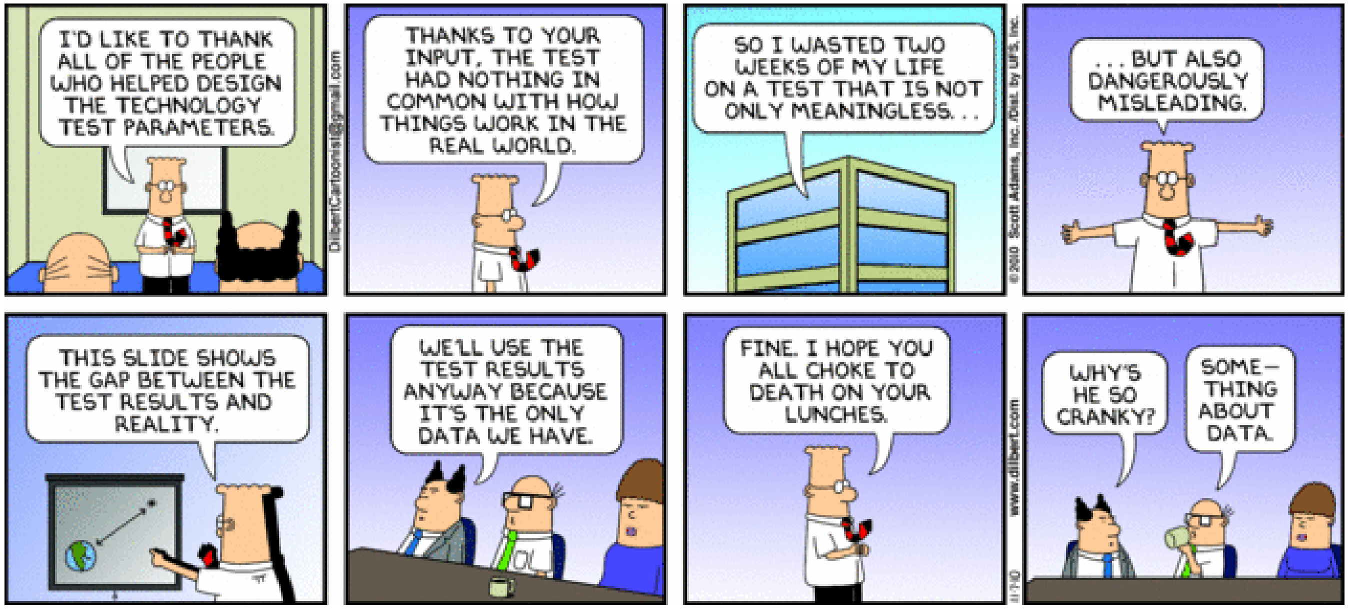

I'D LIKE TO THANK ALL OF THE PEOPLE WHO HELPED DESIGN THE TECHNOLOGY TEST PARAMETERS.
DilbertCartoonist@gmail.com
THANKS TO YOUR INPUT, THE TEST HAD NOTHING IN COMMON WITH HOW THINGS WORK IN THE REAL WORLD.
SO I WASTED TWO WEEKS OF MY LIFE ON A TEST THAT IS NOT ONLY MEANINGLESS...
© 2010 Scott Adams, Inc./Dist. by UFS, Inc.
...BUT ALSO DANGEROUSLY MISLEADING.
THIS SLIDE SHOWS THE GAP BETWEEN THE TEST RESULTS AND REALITY.
WE'LL USE THE TEST RESULTS ANYWAY BECAUSE IT'S THE ONLY DATA WE HAVE.
FINE. I HOPE YOU ALL CHOKE TO DEATH ON YOUR LUNCHES.
www.dilbert.com
WHY'S HE SO CRANKY?
SOME-THING ABOUT DATA.

+

Introduction to JUnit

IT323 – Software Engineering II

By: Mashael Al-Duwais

What is Unit Testing?

- A procedure to validate individual units of Source Code
- Example: A procedure, method or class
- Validating each individual piece reduces errors when integrating the pieces together later

Automated Unit Tests with JUnit

- Junit is a simple, open source unit testing framework for Java
- Allows you to write unit tests in Java using a simple interface
- Automated testing enables running and rerunning tests very easily and quickly
- Supported by www.junit.org

JUnit Example

```
public class Calc
{
   static public int add (int a, int b)
   {
      return a + b;
   }
}
```

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

public class CalcTest
{
   @Test
   public void testAdd()
        {
        int result = Calc.add(2,3);
        assertEquals( 5, result);
        }
}
```

Basic Information

- Test **Suit**
  - A collection of of test cases/classes executed together
- Test **Class**
  - Named [classname]Test.java, where classname is the name of the class that is tested.
- Test **Method**
  - A test method can contain one or more test cases.
  - Annotated with @Test to indicate them to Junit.
  - Has one or more assert statements or fail statements.
- Test **Case**
  - A test case is usually a single run of a specific functionality.

Steps to perform unit tests (Junit)

1. **Prepare** (or **setUp()**) environment conditions that must be met, according to the test plan. At this stage, define and set prefix values. E.g. instantiate objects, initialize fields, turn on logging, etc.

2. **Execute** the test case. This means, executing (exercising) the part of the code to be tested. For that we use some test inputs (test case values), according to the test plan.

3. **Evaluate** (or **assert*()**) the results, or side effects generated by the execution of the test case, against an expected value as defined in the test plan.

4. **Clean up** (or **tearDown()**) the test environment if needed so that further testing activities can be done, without being influenced by the previous test cases. We deal here with postfix values.

+

Step 1: Unit Testing with JUnit 4

1. **Prepare** (or **setUp()**) the test environment:

   - Annotate with @Before: Those methods are executed before each test case (test method).

```
@Before
public void setUp() {
   s = new Sample();
}
```

+ Step 2&3: Unit Testing with JUnit 4

2. Execute the test case.

3. Evaluate the results (using assertion).

```
@Test
public void testAddition( ) {
   int a=3 , b=6;
   int expectedOutput = (a+b);
   int res = s.Addition(a, b);
   assertEquals(expectedOutput, res);
}
```

+
Step 4: Unit Testing with JUnit 4

4. Clean up (or tearDown()) the test environment is done in one or several methods that are run after execution of each test method.
   - A method has to be annotated with @After.
   - If you allocate external resources in a @Before method, you need to release them after the test runs.

```
@After
public void tearDown() {
   s = null;
}
```

junit.framework.Assert

- Provide static methods which can help comparing the expected result and actual result.
- If any assert is violated, a failure will be recorded.

assertEquals (expected, actual)	assertEquals (message, expected, actual)
assertSame (expected, actual)	assertSame (message, expected, actual)
assertNotSame (unexpected, actual)	assertNotSame (message, unexpected, actual)
assertFalse (condition)	assertFalse (message, condition)
assertTrue (condition)	assertTrue (message, condition)
assertNotNull (object)	assertNotNull (message, object)
assertNull (object)	assertNull (message, object)
fail ()	fail (message)

Test Execution

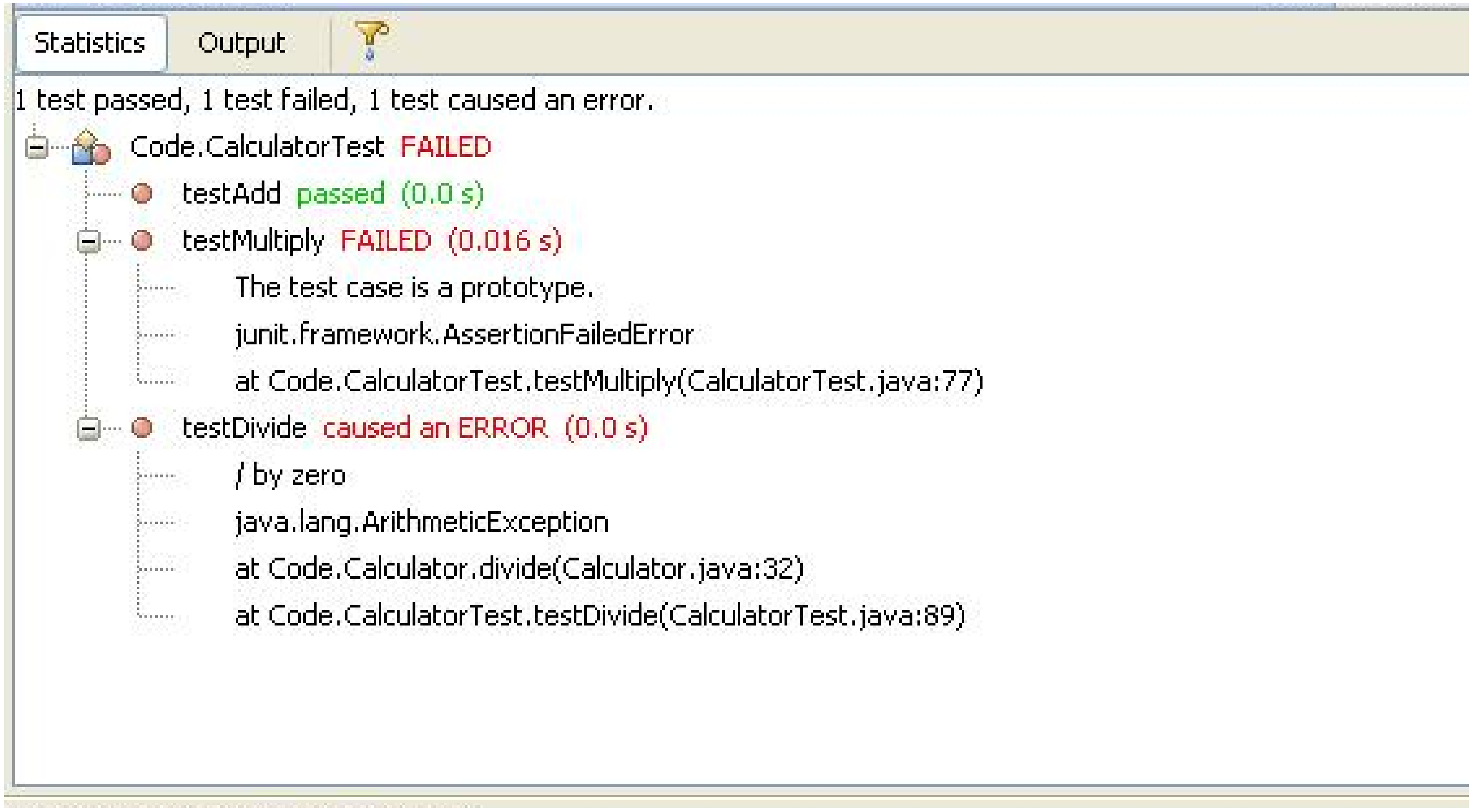
- Execute a test by using the *Run* function of the IDE.
  - NetBeans/Eclipse, can use a default test runner-- all the tests in the class run one by one.

Status of a Test

- A test is a single run of a test method.
- Success
  - A test succeeds in time when No assert is violated; No fail statement is reached; No unexpected exception is thrown.
- Failure
  - A test fails when an assert is violated or a fail statement is reached.
- Error
  - An unexpected exception is thrown or timeout happens.

Status of a Test

- On failure and error, the test results also show a stack trace of the execution.



```
1 test passed, 1 test failed, 1 test caused an error.
Code.CalculatorTest FAILED
    testAdd passed (0.0 s)
    testMultiply FAILED (0.016 s)
        The test case is a prototype.
        junit.framework.AssertionFailedError
        at Code.CalculatorTest.testMultiply(CalculatorTest.java:77)
    testDivide caused an ERROR (0.0 s)
        / by zero
        java.lang.ArithmeticException
        at Code.Calculator.divide(Calculator.java:32)
        at Code.CalculatorTest.testDivide(CalculatorTest.java:89)
```

Test Suit

- To run a subset of the tests or run tests in a specific order.
- A test suite is basically a class with a method that invokes the specified test cases, such as specific test classes, test methods in test classes and other test suites.
- You can create manually or the IDE can generate the suites for you.
- Example:

```
TestSuite suite= new TestSuite();
 suite.addTest(new MathTest("testAdd"));
 suite.addTest(new MathTest("testDivideByZero"));
```

+

Junit with Netbeans

1. Create the Java Project
2. Create the Java Class
3. Create a Test Class for Java Class
4. Write Test Methods for Test Class
5. Run the Test
6. Create Test Suit (optional)

Junit with NetBeans

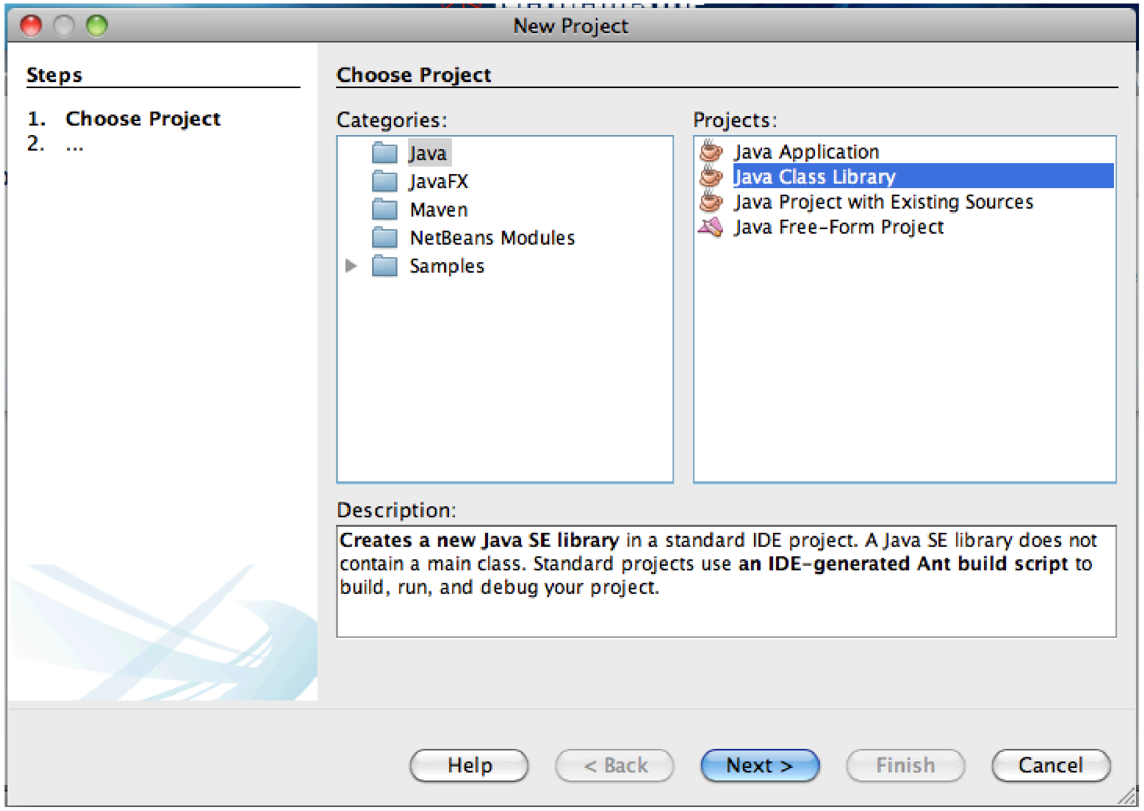
Lets Do The Code

- Make a simple class named (`SimpleMath.java`) that has the following methods:
  - Addition
  - Subtraction
  - Multiplication
- Create the test class for these method.

+

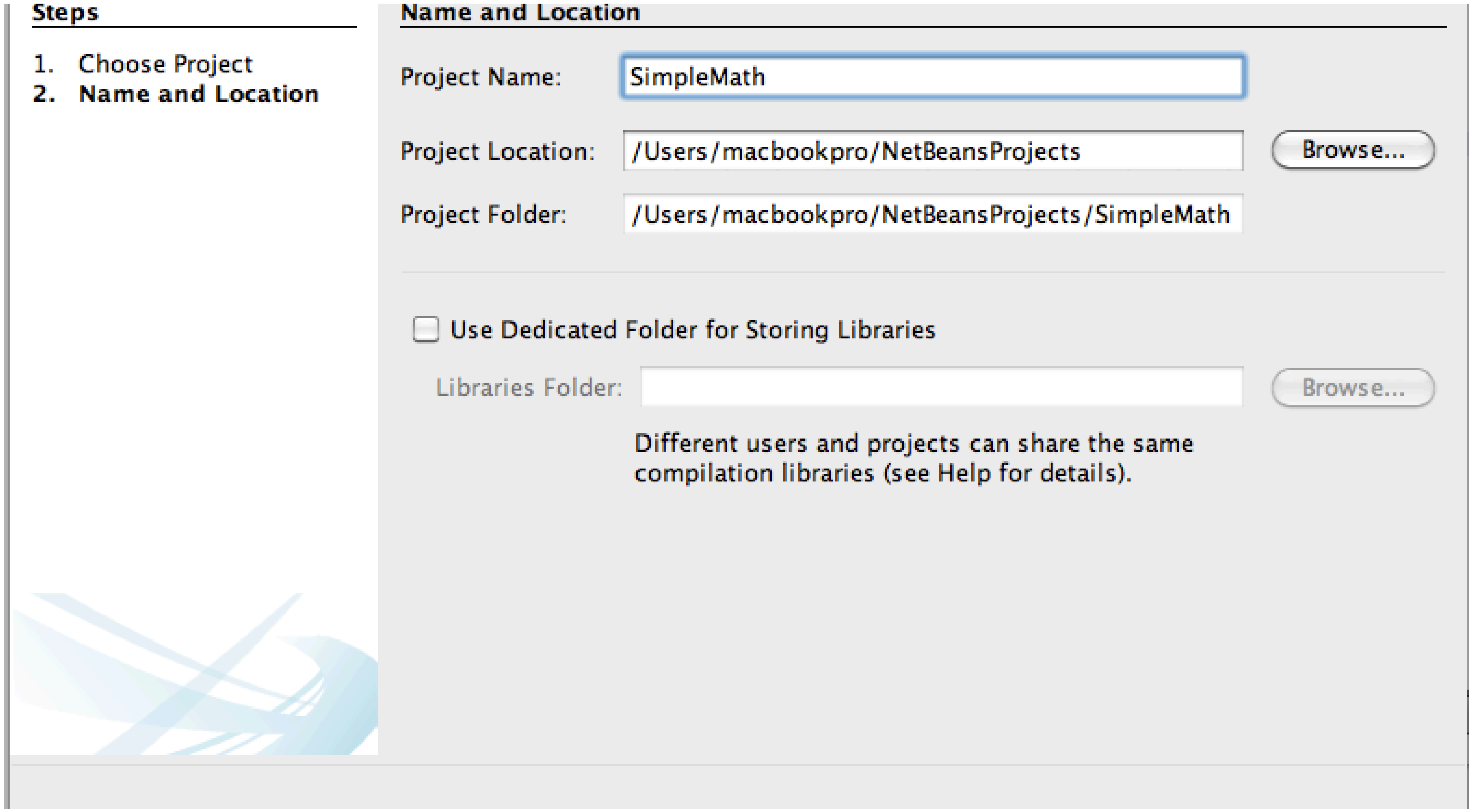
1. Create the Java Project

- Launch NetBeans
- File→ New Project



Description:
Creates a new Java SE library in a standard IDE project. A Java SE library does not contain a main class. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.

1. Create the Java Project



Steps

1. Choose Project
2. **Name and Location**

Name and Location

Project Name: SimpleMath

Project Location: /Users/macbookpro/NetBeansProjects

Project Folder: /Users/macbookpro/NetBeansProjects/SimpleMath

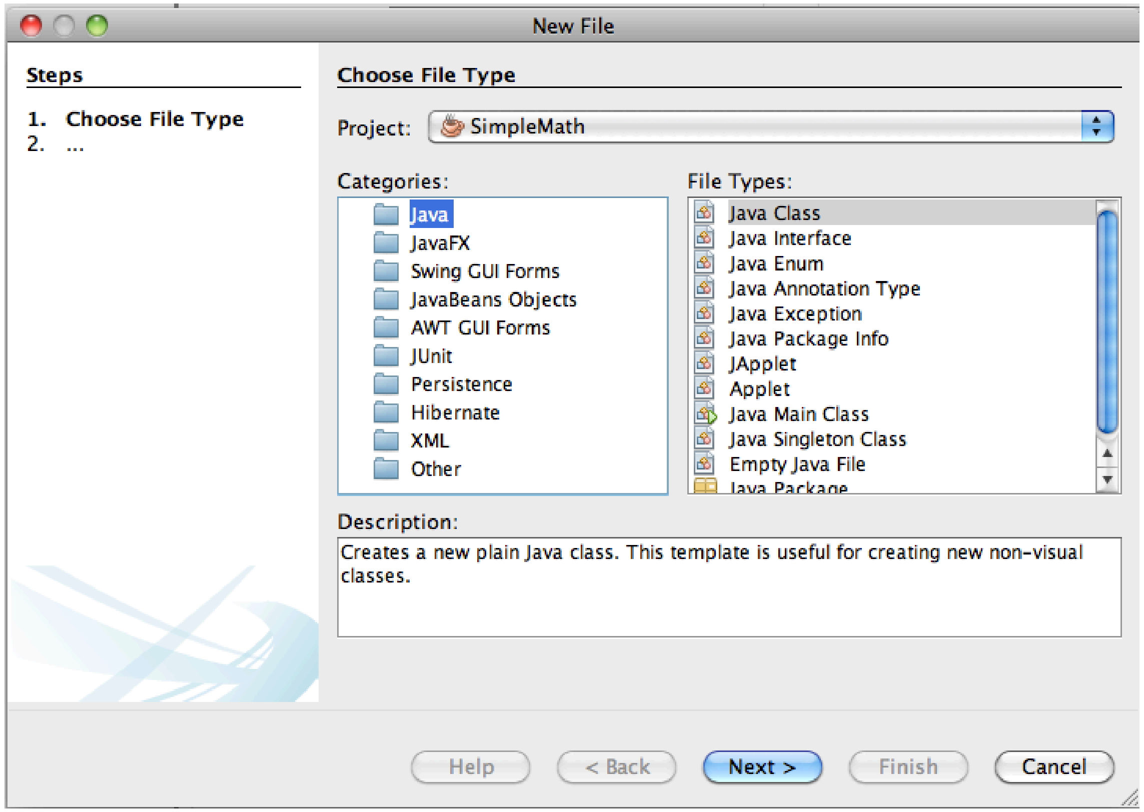
☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

2. Create the Java Class

- File→ New File



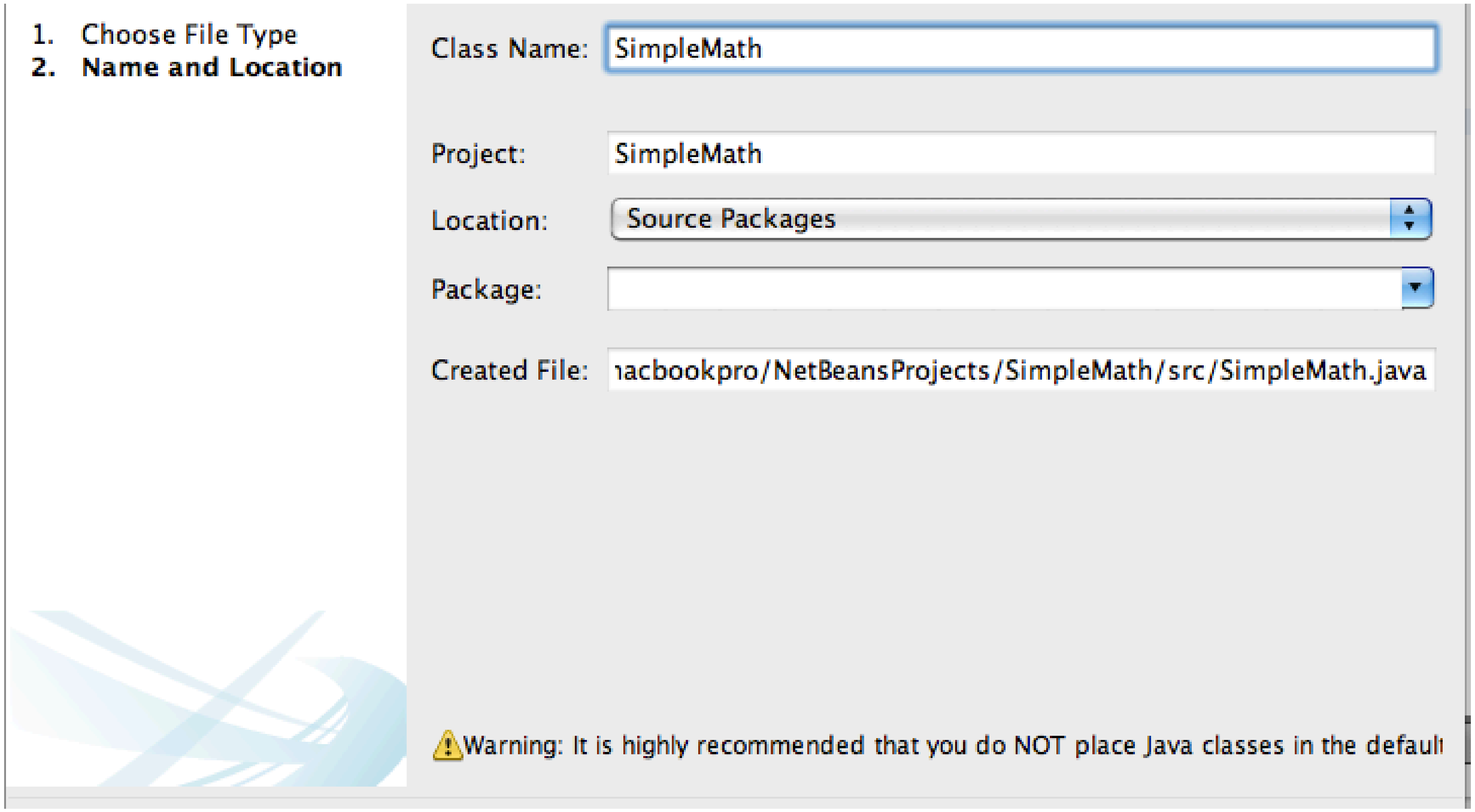
Project: SimpleMath

Categories:
- Java
- JavaFX
- Swing GUI Forms
- JavaBeans Objects
- AWT GUI Forms
- JUnit
- Persistence
- Hibernate
- XML
- Other

File Types:
- Java Class
- Java Interface
- Java Enum
- Java Annotation Type
- Java Exception
- Java Package Info
- JApplet
- Applet
- Java Main Class
- Java Singleton Class
- Empty Java File
- Java Package

Description:
Creates a new plain Java class. This template is useful for creating new non-visual classes.

2. Create the Java Class

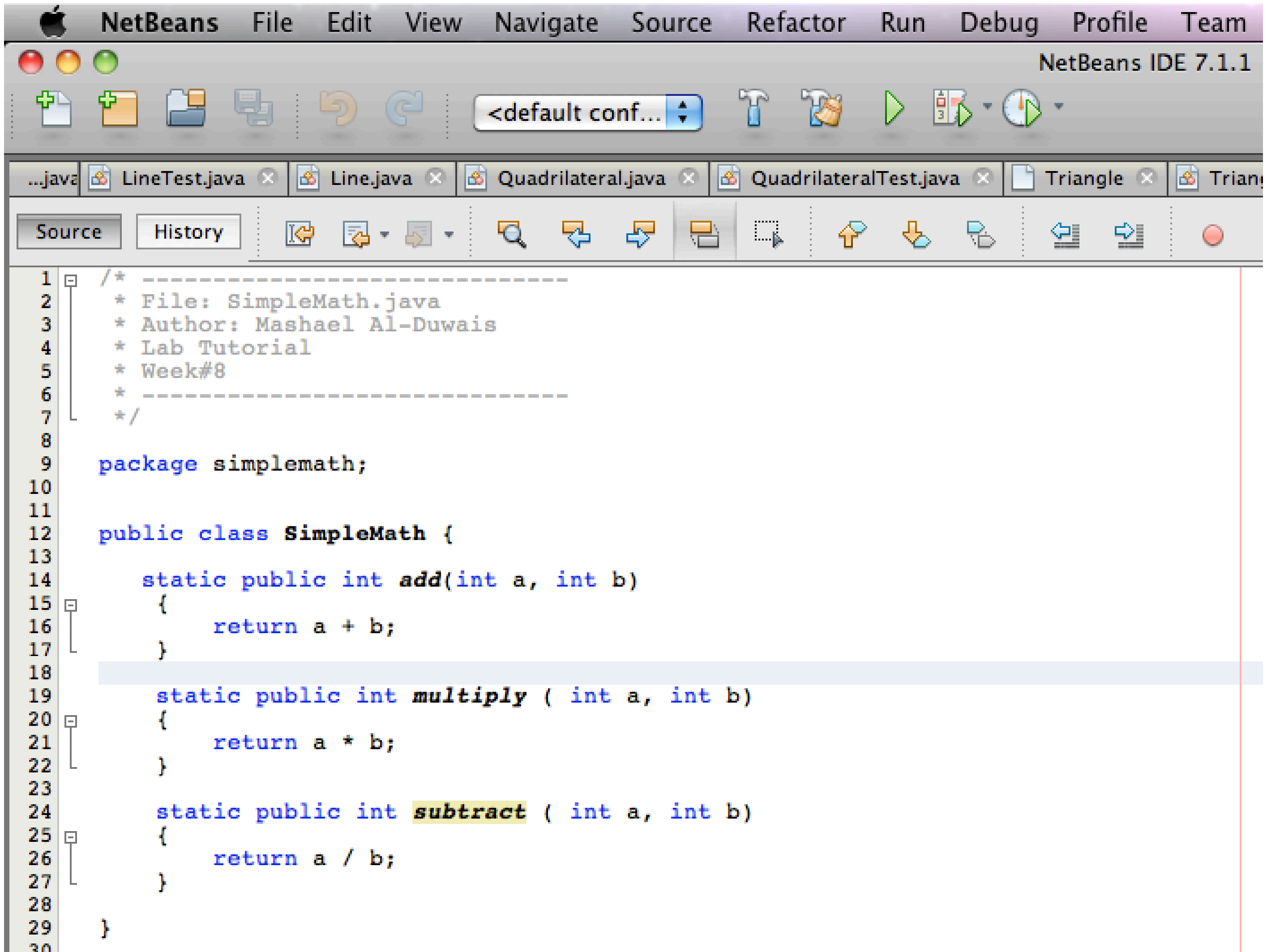

1. Choose File Type
2. **Name and Location**

Class Name:	SimpleMath
Project:	SimpleMath
Location:	Source Packages
Package:	
Created File:	ıacbookpro/NetBeansProjects/SimpleMath/src/SimpleMath.java

⚠Warning: It is highly recommended that you do NOT place Java classes in the defaul

2. Create the Java Class

SimpleMath.java

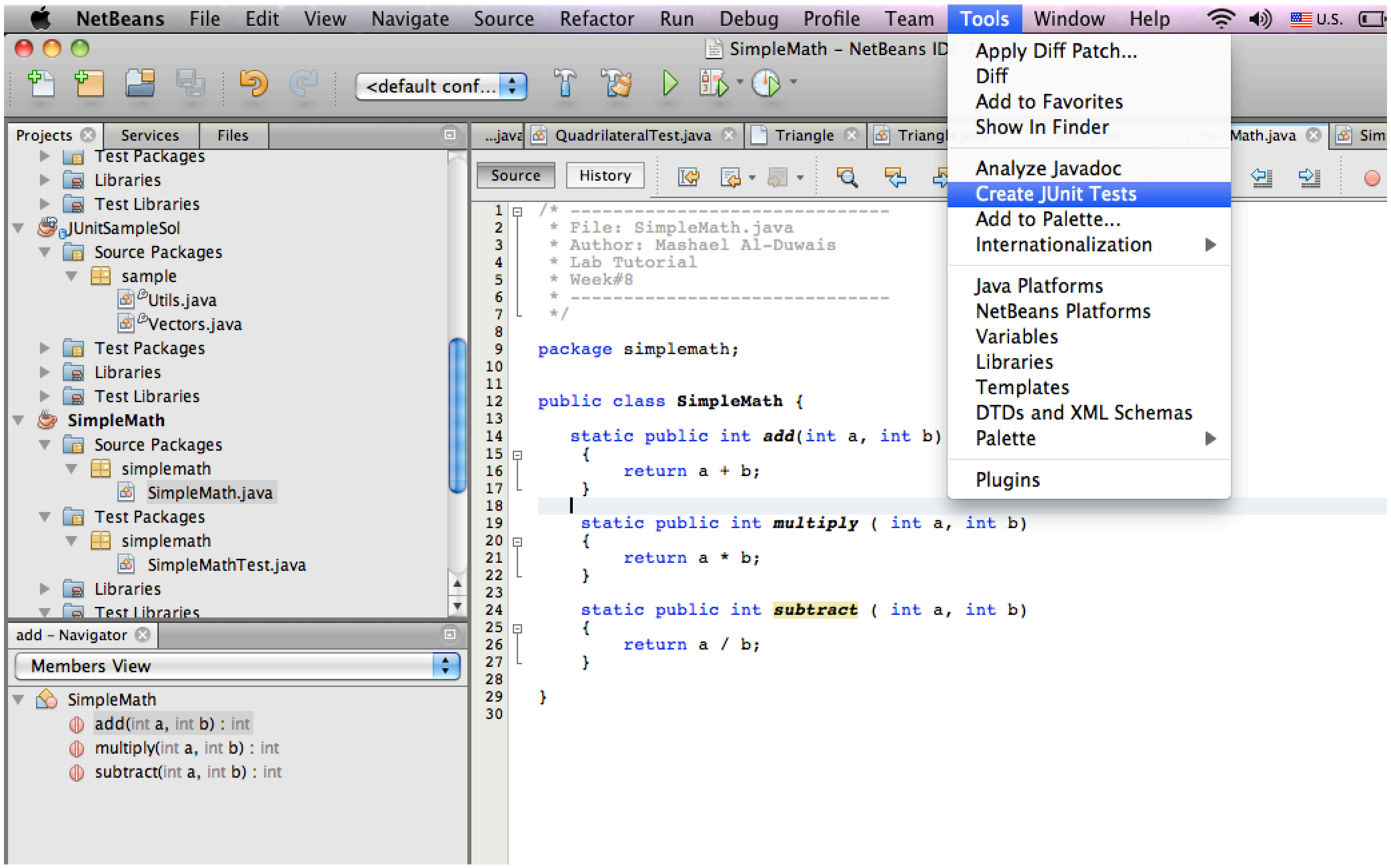

NetBeans File Edit View Navigate Source Refactor Run Debug Profile Team
NetBeans IDE 7.1.1
<default conf...
...java LineTest.java Line.java Quadrilateral.java QuadrilateralTest.java Triangle Trian
Source History

```
/* ------------------------------
 * File: SimpleMath.java
 * Author: Mashael Al-Duwais
 * Lab Tutorial
 * Week#8
 * ------------------------------
 */

package simplemath;


public class SimpleMath {

   static public int add(int a, int b)
    {
        return a + b;
    }

    static public int multiply ( int a, int b)
    {
        return a * b;
    }

    static public int subtract ( int a, int b)
    {
        return a / b;
    }

}
```

+
3. Create a Test Class for Java Class

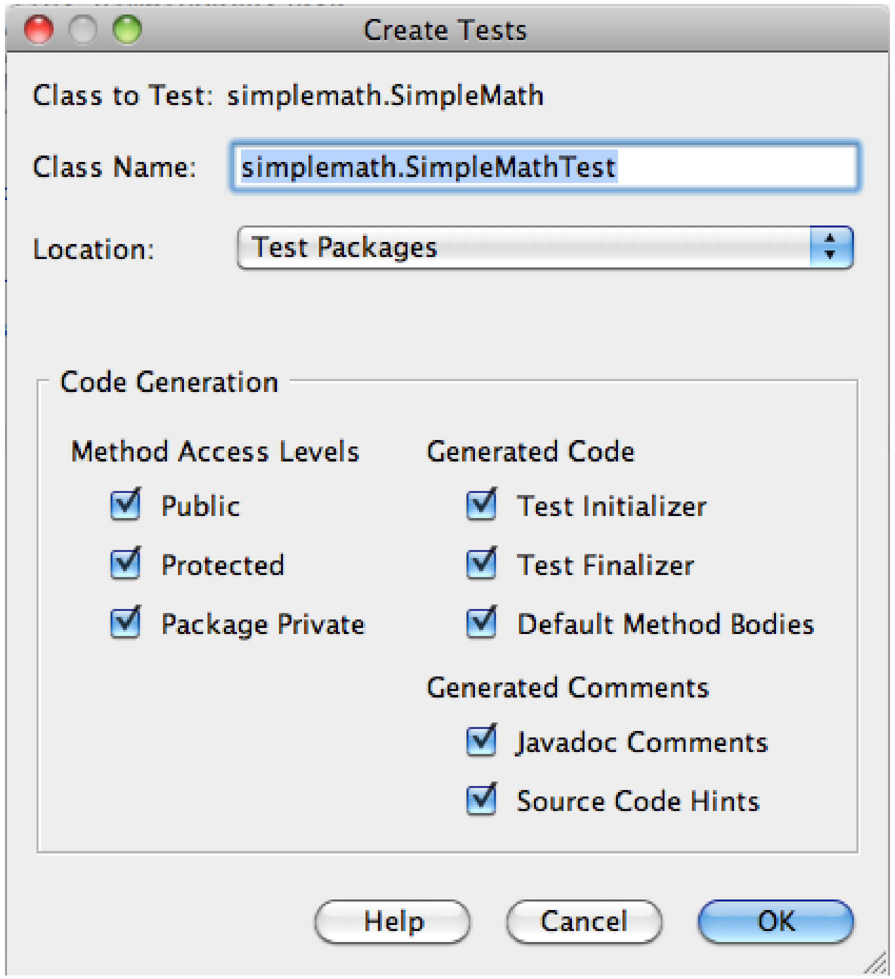
- Choose this menu in netbeans or from Right Click:
  - Tools > Create Junit Test
- Or just simply press Ctrl + Shift + U.
- A window dialogue will appear, choose suitable options.
- Test case will automatically build inside the test package folder.

3. Create a Test Class for Java Class



```
/* --------------------------------
 * File: SimpleMath.java
 * Author: Mashael Al-Duwais
 * Lab Tutorial
 * Week#8
 * --------------------------------
 */

package simplemath;


public class SimpleMath {

    static public int add(int a, int b)
    {
        return a + b;
    }

    static public int multiply ( int a, int b)
    {
        return a * b;
    }

    static public int subtract ( int a, int b)
    {
        return a / b;
    }

}
```

3. Create a Test Class for Java Class



Class to Test: simplemath.SimpleMath

Class Name: simplemath.SimpleMathTest

Location: Test Packages

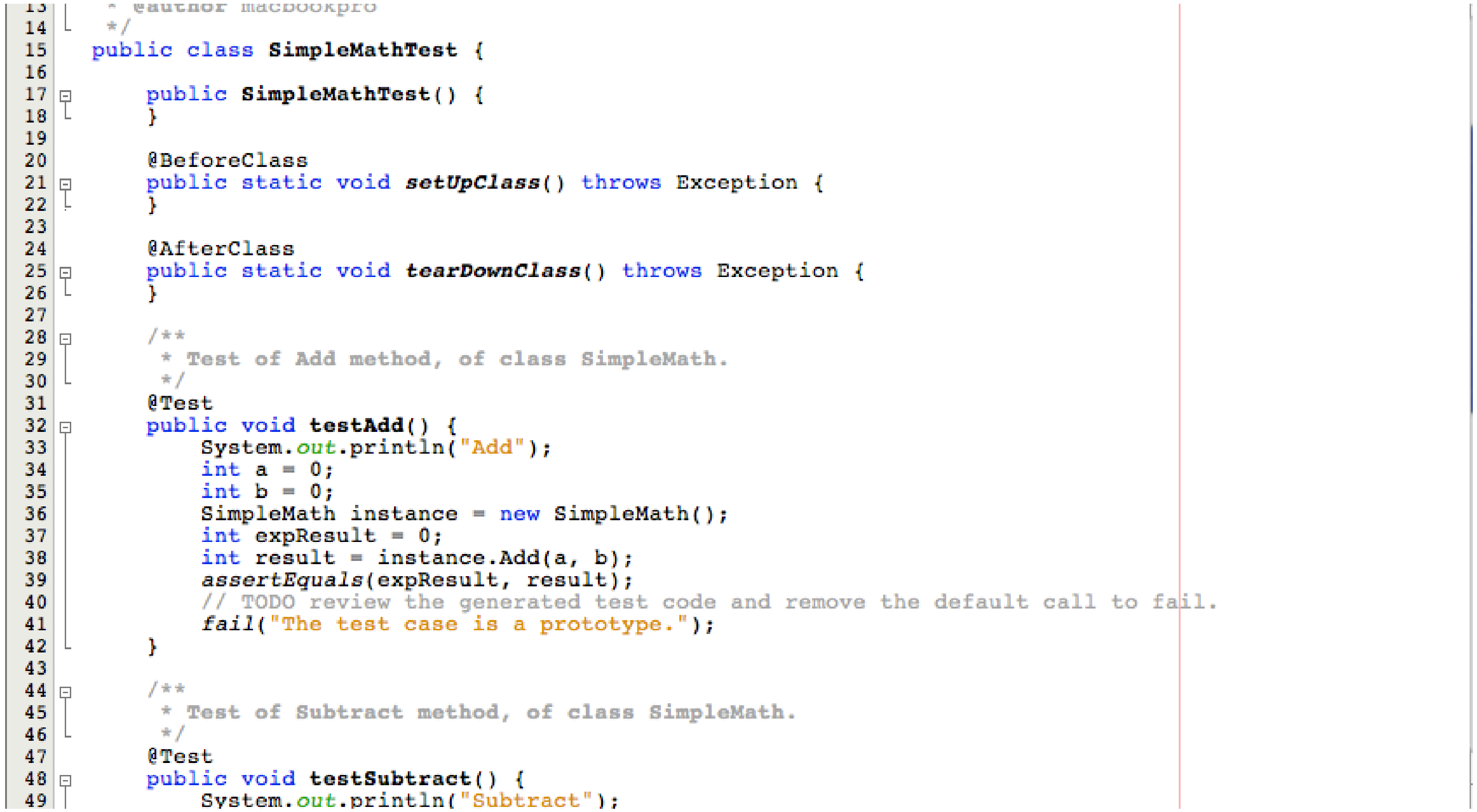
Code Generation

Method Access Levels
- [✓] Public
- [✓] Protected
- [✓] Package Private

Generated Code
- [✓] Test Initializer
- [✓] Test Finalizer
- [✓] Default Method Bodies

Generated Comments
- [✓] Javadoc Comments
- [✓] Source Code Hints

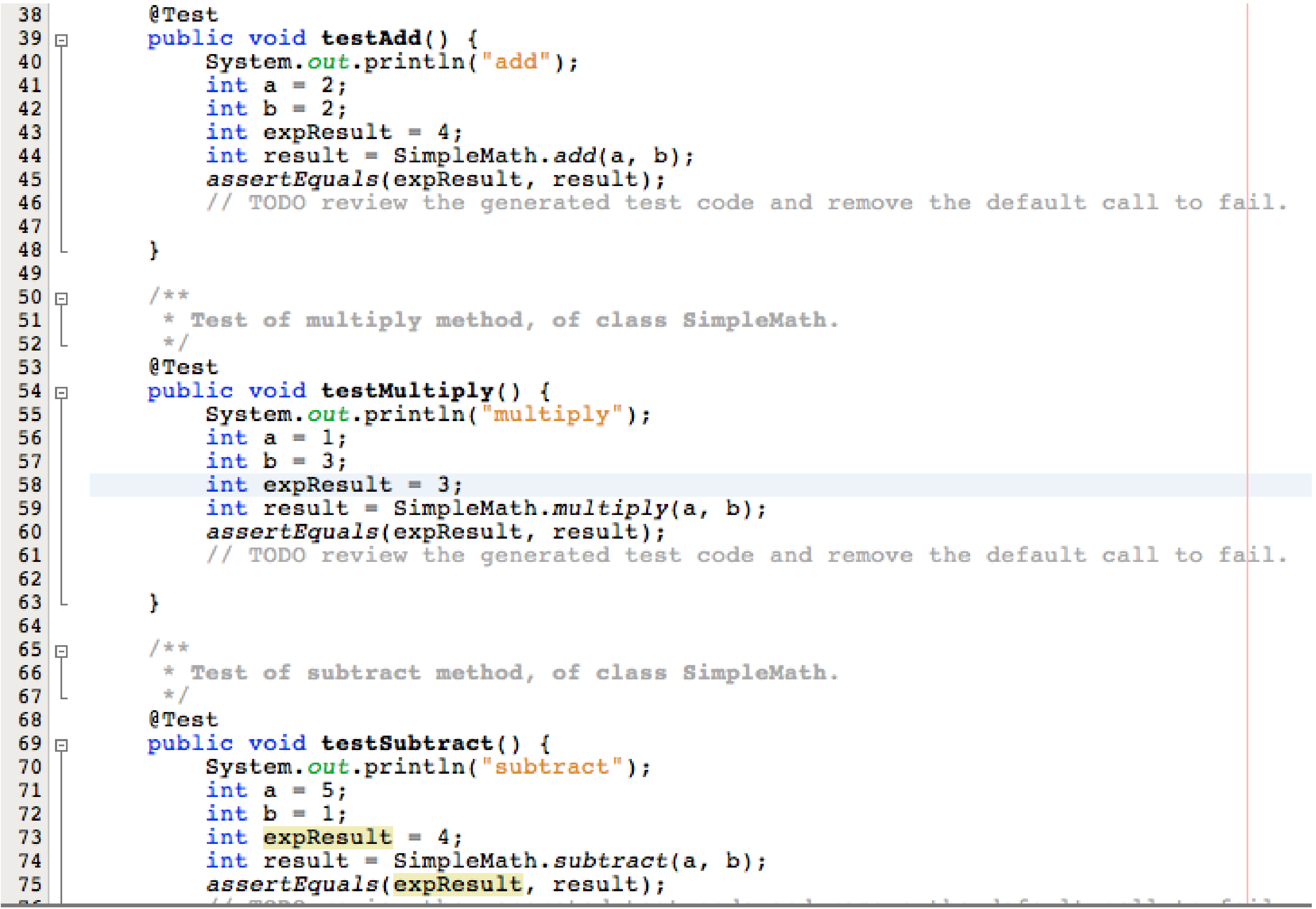
3. Create a Test Class for Java Class



```
 * @author macbookpro
 */
public class SimpleMathTest {

    public SimpleMathTest() {
    }

    @BeforeClass
    public static void setUpClass() throws Exception {
    }

    @AfterClass
    public static void tearDownClass() throws Exception {
    }

    /**
     * Test of Add method, of class SimpleMath.
     */
    @Test
    public void testAdd() {
        System.out.println("Add");
        int a = 0;
        int b = 0;
        SimpleMath instance = new SimpleMath();
        int expResult = 0;
        int result = instance.Add(a, b);
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        fail("The test case is a prototype.");
    }

    /**
     * Test of Subtract method, of class SimpleMath.
     */
    @Test
    public void testSubtract() {
        System.out.println("Subtract");
```

4. Write Test Methods for Test Class

SimpleMathTest.Java

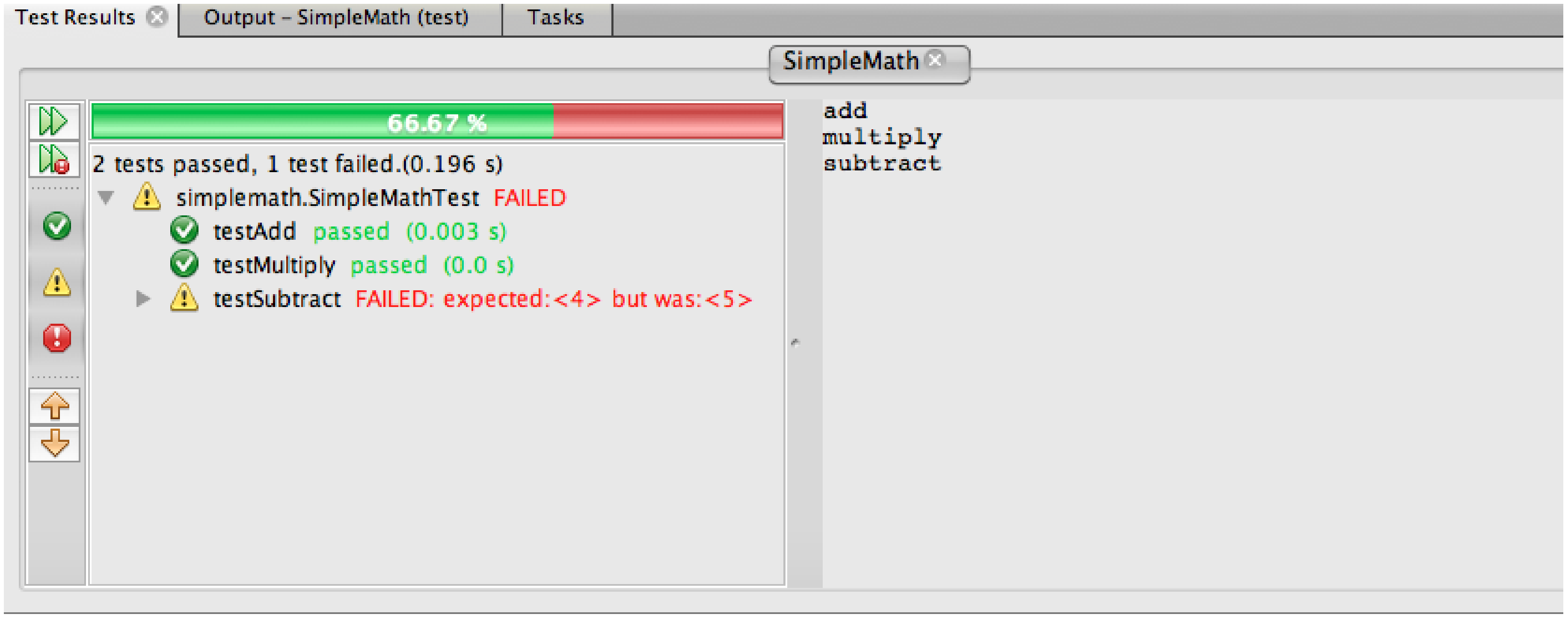


```
    @Test
    public void testAdd() {
        System.out.println("add");
        int a = 2;
        int b = 2;
        int expResult = 4;
        int result = SimpleMath.add(a, b);
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.

    }

    /**
     * Test of multiply method, of class SimpleMath.
     */
    @Test
    public void testMultiply() {
        System.out.println("multiply");
        int a = 1;
        int b = 3;
        int expResult = 3;
        int result = SimpleMath.multiply(a, b);
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.

    }

    /**
     * Test of subtract method, of class SimpleMath.
     */
    @Test
    public void testSubtract() {
        System.out.println("subtract");
        int a = 5;
        int b = 1;
        int expResult = 4;
        int result = SimpleMath.subtract(a, b);
        assertEquals(expResult, result);
```

4. Write Test Methods for Test Class

- Assign the variable value for the test case.
- Remove the fail() method in return valued method test.
- Run the test class using Shift + F6.
- See the test result

+
5. Run the Test



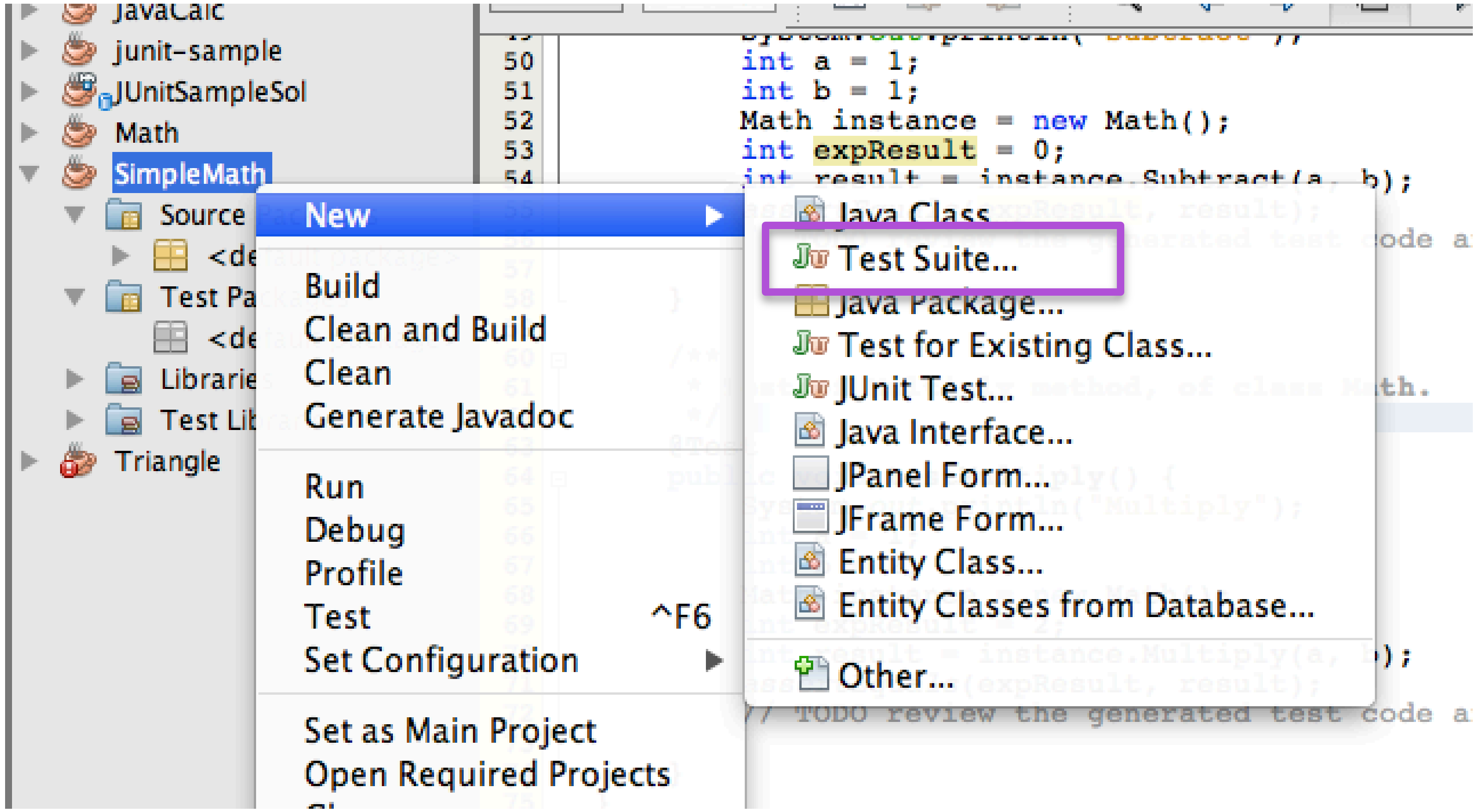
```
2 tests passed, 1 test failed.(0.196 s)
simplemath.SimpleMathTest FAILED
    testAdd passed (0.003 s)
    testMultiply passed (0.0 s)
    testSubtract FAILED: expected:<4> but was:<5>
```

```
add
multiply
subtract
```

6. Create Test Suit

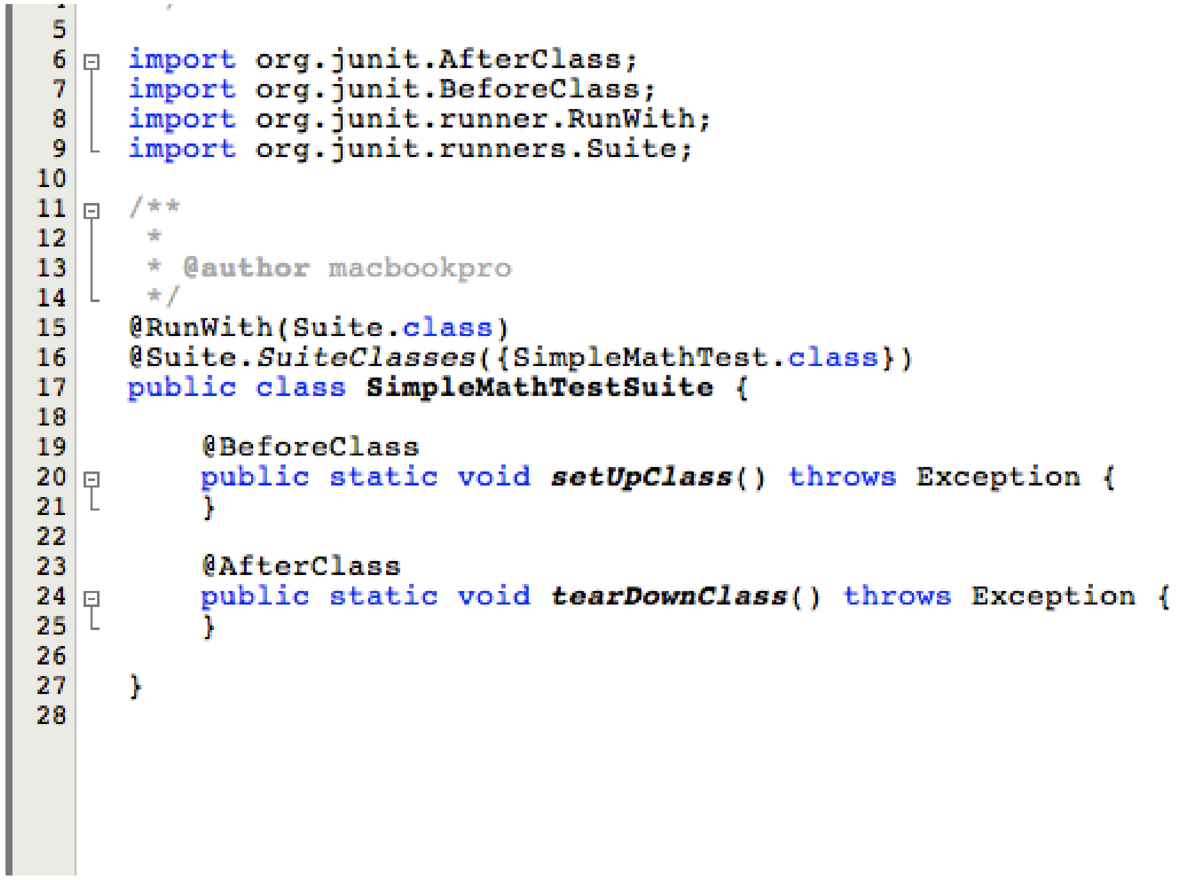
- Right-click the project node in the Projects window and choose New > Other to open the New File wizard.
- Select the JUnit category and Test Suite. Click Next.
- Type **SimpleMathTestSuit** for the file name.
- Deselect Test Initializer and Test Finalizer. Click Finish.

6. Create Test Suit



```
int a = 1;
int b = 1;
Math instance = new Math();
int expResult = 0;
int result = instance.Subtract(a, b);
```

6. Create Test Suit



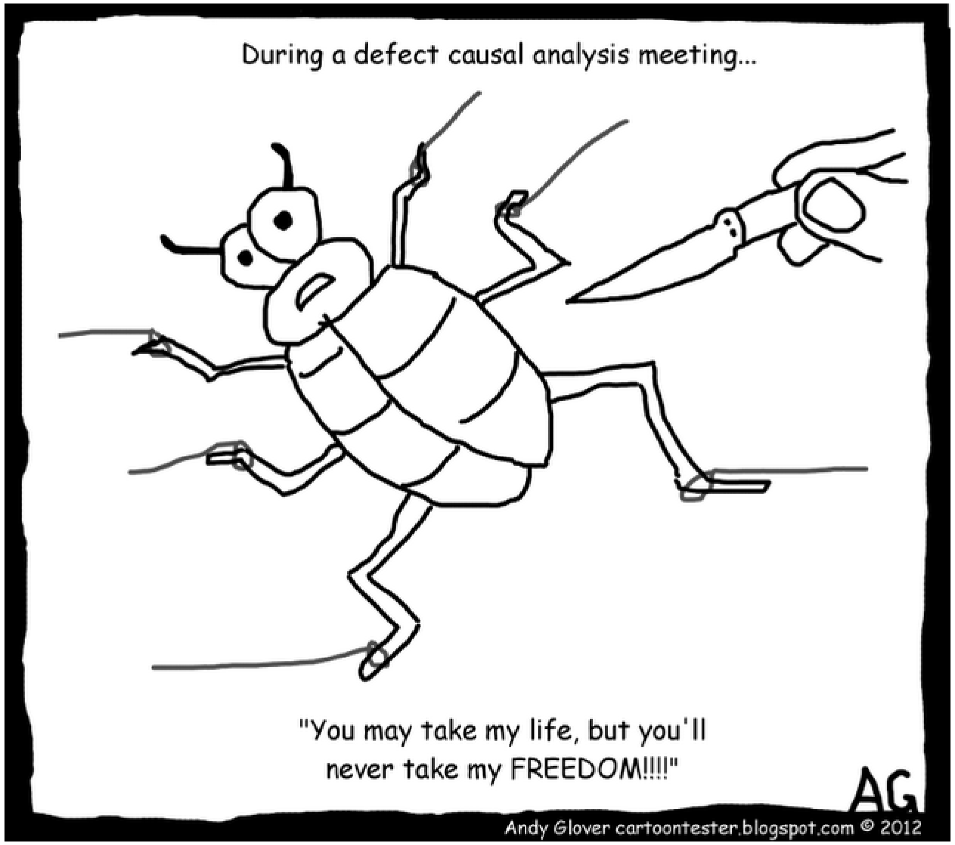
```
import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

/**
 *
 * @author macbookpro
 */
@RunWith(Suite.class)
@Suite.SuiteClasses({SimpleMathTest.class})
public class SimpleMathTestSuite {

    @BeforeClass
    public static void setUpClass() throws Exception {
    }

    @AfterClass
    public static void tearDownClass() throws Exception {
    }

}
```

Junit Resources

- http://junit.sourceforge.net/
- http://code.google.com/p/t2framework/wiki/JUnitQuickTutorial
- http://netbeans.org/kb/docs/java/junit-intro.html

Summary

- Unit tests can help test the details of your program
- Automated unit tests provide constant visibility and easy retesting


During a defect causal analysis meeting...
"You may take my life, but you'll never take my FREEDOM!!!!"
AG
Andy Glover cartoontester.blogspot.com © 2012

References

- LAB-5110 NetBeans™: JUnit (April 2005) (http://developers.sun.com/events/techdays/self_paced_labs.jsp)
- Unit Testing in Eclipse Using JUnit by Laurie Williams, Dright Ho, and Sarah Smith (http://open.ncsu.edu/se/tutorials/junit/#section1_0)
- JUnit Testing With Netbeans (http://www.fsl.cs.sunysb.edu/~dquigley/cse219/index.php?it=netbeans&tt=junit&pf=y)
- JUnit 4 Tutorial by Ji Chao Zhang, October 23, 2006 (CSI 5111 presentation) Based on "Get Acquainted with the New Advanced Features of JUnit 4" by Antonio Goncalves
- JUnit Test Infected: Programmers Love Writing Tests; Kent Beck, Erich Gamma.
- JUnit FAQ Edited by Mike Clark (http://junit.sourceforge.net/doc/faq/faq.htm#overview_1)