Supplement: Testing Java Programs Using JUnit

For Introduction to Java Programming
By Y. Daniel Liang

JUnit is a simple framework for testing Java programs. You can download the latest version of JUnit from http://sourceforge.net/projects/junit/files/. At present, the latest version is junit-4.10.jar. Download this file and add it to the classpath as follows:

```
set CLASSPATH=.;%CLASSPATH%;%JUNIT_HOME%\junit-4.10.jar
```

To use JUnit, create a test class. By convention, if the class to be tested is named A, the test class should be named ATest. A simple template of a test class may look like this:

```
package test;

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

public class ATest {
  @Test
  public void m1() {
    // Write a test method
  }

  @Test
  public void m2() {
    // Write another test method
  }

  @Before
  public void setUp() throws Exception {
    // Common objects used by test methods may be set up here
  }
}
```

To run the test from the console, use the following command:

```
java org.junit.runner.JUnitCore test.ATest
```

When this command is executed, JUnitCore controls the execution of ATest. It first executes the setUp() method to set up the common objects used for the test, and then executes test methods m1 and m2 in this order. You may define multiple test methods if desirable.

The following methods can be used to implement a test method:

assertTrue(booleanExpression)
The method reports success if the booleanExpression evaluates true.

assertEquals(Object, Object)
The method reports success if the two objects are the same using the equals method.

assertNull(Object)
The method reports success if the object reference passed is null.

fail(String)
Causes the test to fail and prints out the string.

Here is an example of a test class for testing java.util.ArrayList.

```
package test;

import org.junit.*;
import static org.junit.Assert.*;
import java.util.*;

public class ArrayListTest {
  private ArrayList<String> list = new ArrayList<String>();

  @Before
  public void setUp() throws Exception {
  }

  @Test
  public void testInsertion() {
    list.add("Beijing");
    assertEquals("Beijing", list.get(0));
    list.add("Shanghai");
    list.add("Hongkong");
    assertEquals("Hongkong", list.get(list.size() - 1));
  }

  @Test
  public void testDeletion() {
    list.clear();
    assertTrue(list.isEmpty());

    list.add("A");
    list.add("B");
    list.add("C");
    list.remove("B");
    assertEquals(2, list.size());
  }
}
```

<Output>

```
Testsuite: test.ArrayListTest
Setup
Setup
Tests run: 2, Failures: 0, Errors: 0, Time elapsed: 0.172 sec
```

<End Output>

You can define any number of test methods. In this example, two test methods testInsertion and testDeletion are defined. JUnit executes testInsertion and testDeletion in this order.

An IDE like NetBeans and Eclipse can greatly simplify the process for creating and running test classes. Consider using NetBeans 6.0. Let us create a class Loan.java for Listing 9.2 in Chapter 9.

Create a new project named JUnitDemo and class Loan in the project, as shown in Figure 1. To create a test class for Loan, right-click Loan in the project pane to display a context menu. Choose **Tools › Create JUnit Test** to display a dialog box, as shown in Figure 2. Check all boxes to generate LoanTest. LoanTest.java now appears under the *Test Packages* node in the test project, as shown in Figure 3.

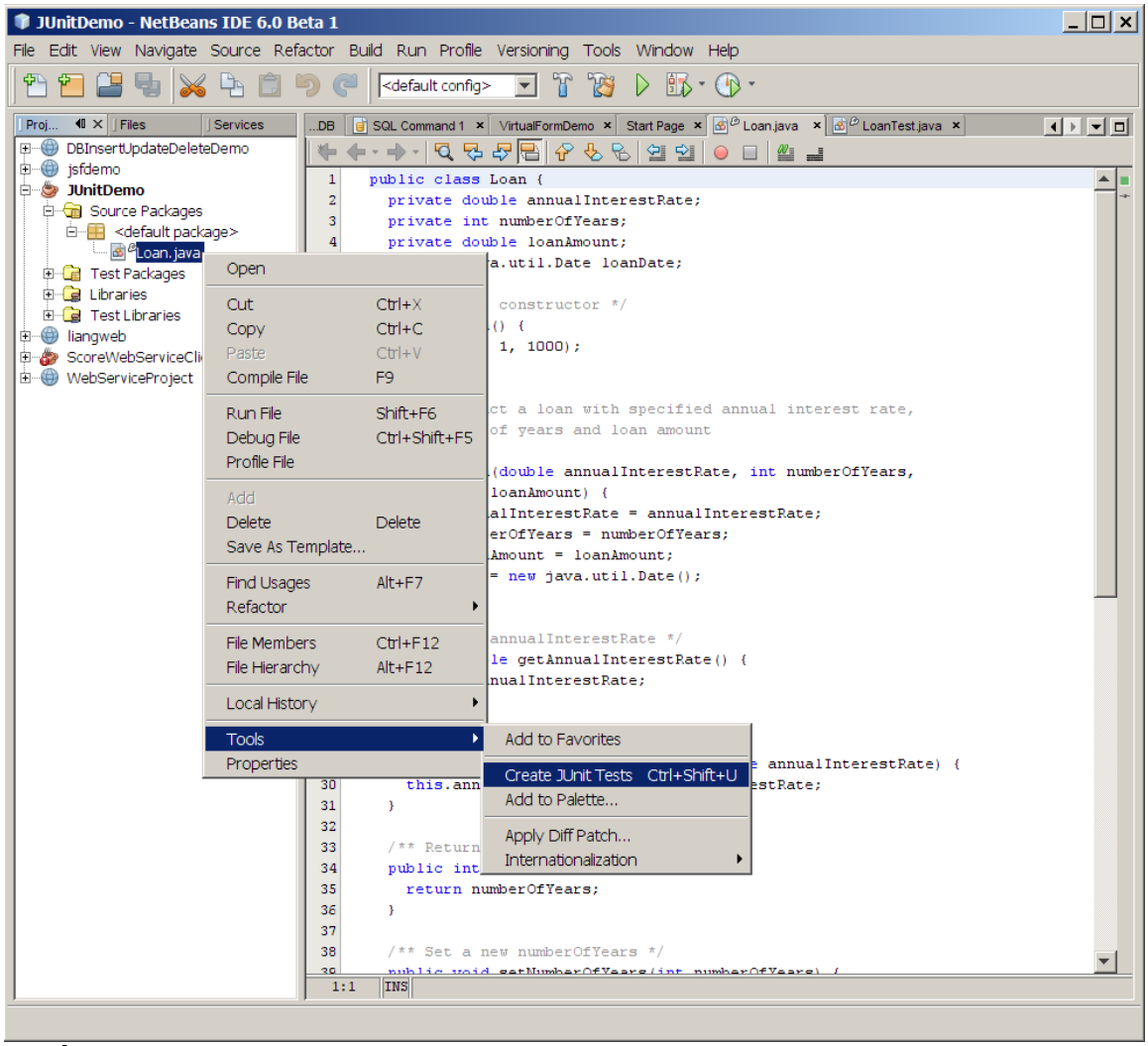

Figure 1

A test class can be created automatically in NetBeans.

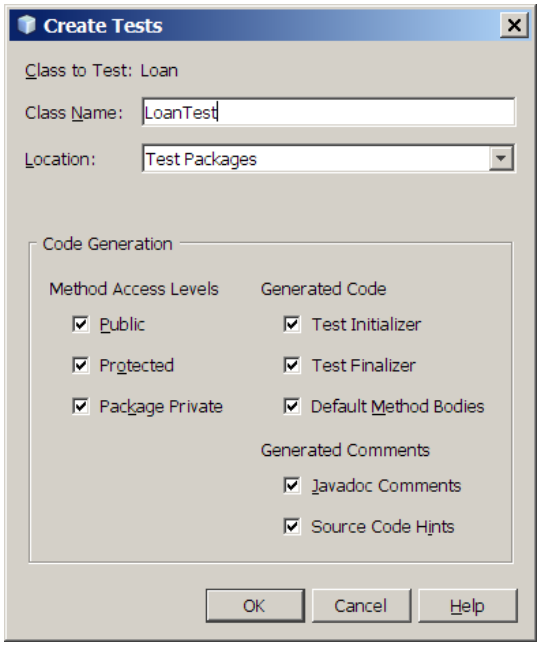

Figure 2
You can specify the options for code generation.

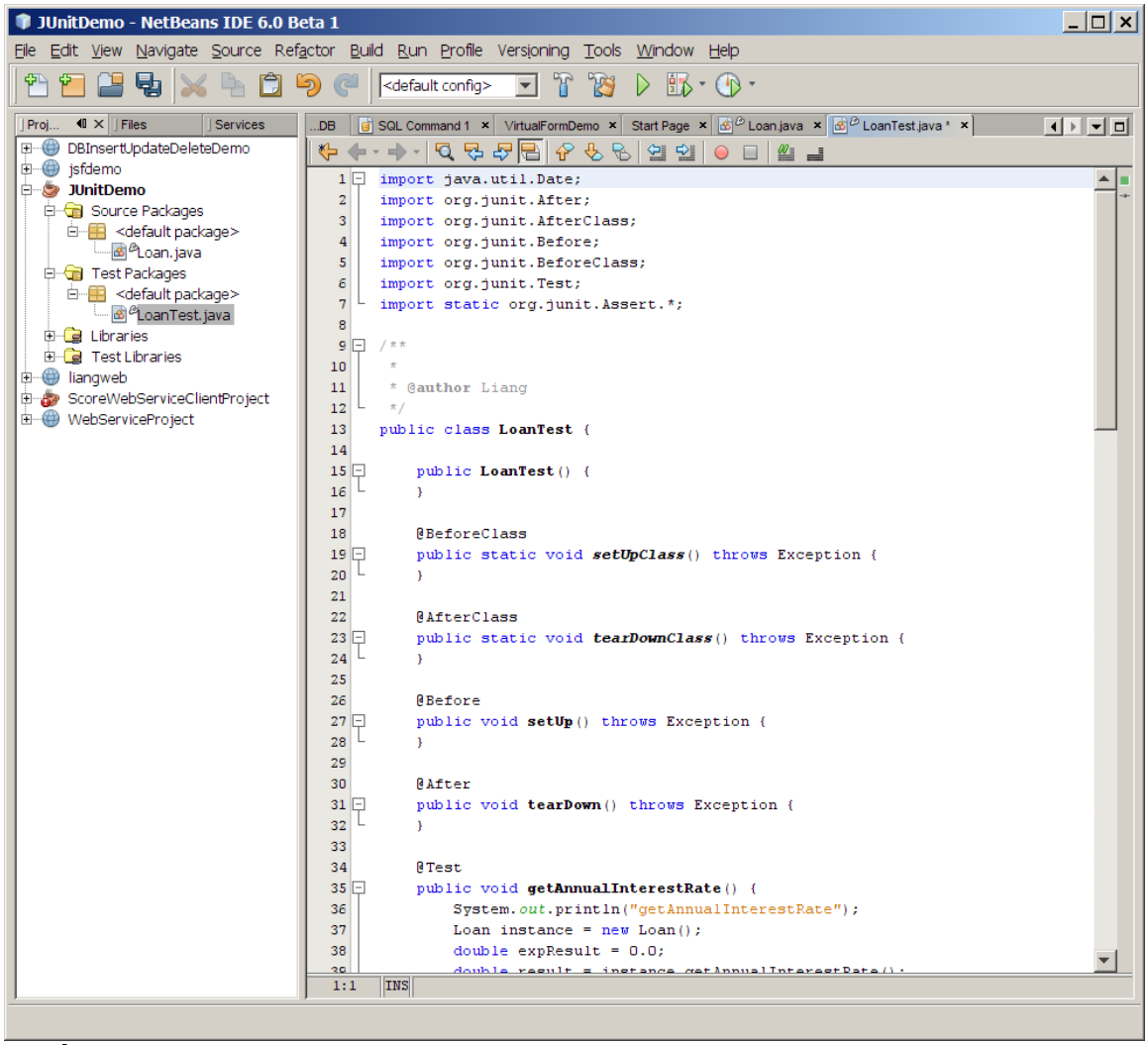

Figure 3
The test class LoanTest is generated.

The automatically generated test class contains the methods setUpClass(), tearDownClass(), setUp(), and tearDown(). These are the test class life-cycle methods. These methods are invoked in this order: setUpClass(), setUp(), test methods, tearDown(), and tearDownClass(). You can implement these methods to set up classes and objects before the test and destroy the objects and remove the classes after the test, if necessary.

NetBeans automatically generates test methods for each method. You need to modify the code to test the methods in the class.