Software testing

Recommended Reading

We recommend the following sources of information:

- General information about JUnit and how to write test cases: http://junit.org/
- The JUnit FAQ: http://junit.sourceforge.net/doc/faq/faq.htm
- Testing an expected exception: http://junit.sourceforge.net/doc/faq/faq.htm#tests_6
- *Myers' 14 questions: http://tinyurl.com/n3acs7*

Part A – *Unit Testing and Junit*

Unit Testing refers to testing every module/unit separately in order to find failures in its code. Experience has showed that testing each part of the project individually saves much time in the software life cycle. Moreover, combining these tested parts performs better together more properly.

JUnit makes a good connection between development and testing. While programmers develop an application they can use the functionality of JUnit and to run tests to find out whether the unit works as it is expected. This kind of test driven approach is commonly used in practice, especially when using approaches such as *extreme programming*

Purpose

The purposes of this lab are:
1) To learn how to create a test case and run it in the JUnit framework.
2) To get more familiar with the software testing in small scale.
3) To evaluate the test result and improve the test case with better heuristics

Tasks

In this section, we first learn how we can create a test case based on a program in Eclipse, and how we can run this test case via JUnit and evaluate the result. Secondly, you will write the assigned program, its test case, and run the test case against the program to verify whether it performs correctly.

Task One: Introduction

This task consist of an example that will introduce you to JUnit step-by-step.

A. First of all, we need a program (implementation of requirements) to test. The below program converts an ASCII string to the sum of its characters represented as a Binary string i.e. adds up the values (ASCII) of strings , character by character, and show the sum as a binary digit. Copy the following code into the eclipse and save as ChekcStr.java[1]

Note: All source methods in the class under test must be public or protected, not private, in order to be tested by JUnit. If the method in the class under test is private, the test class must be in the same package.

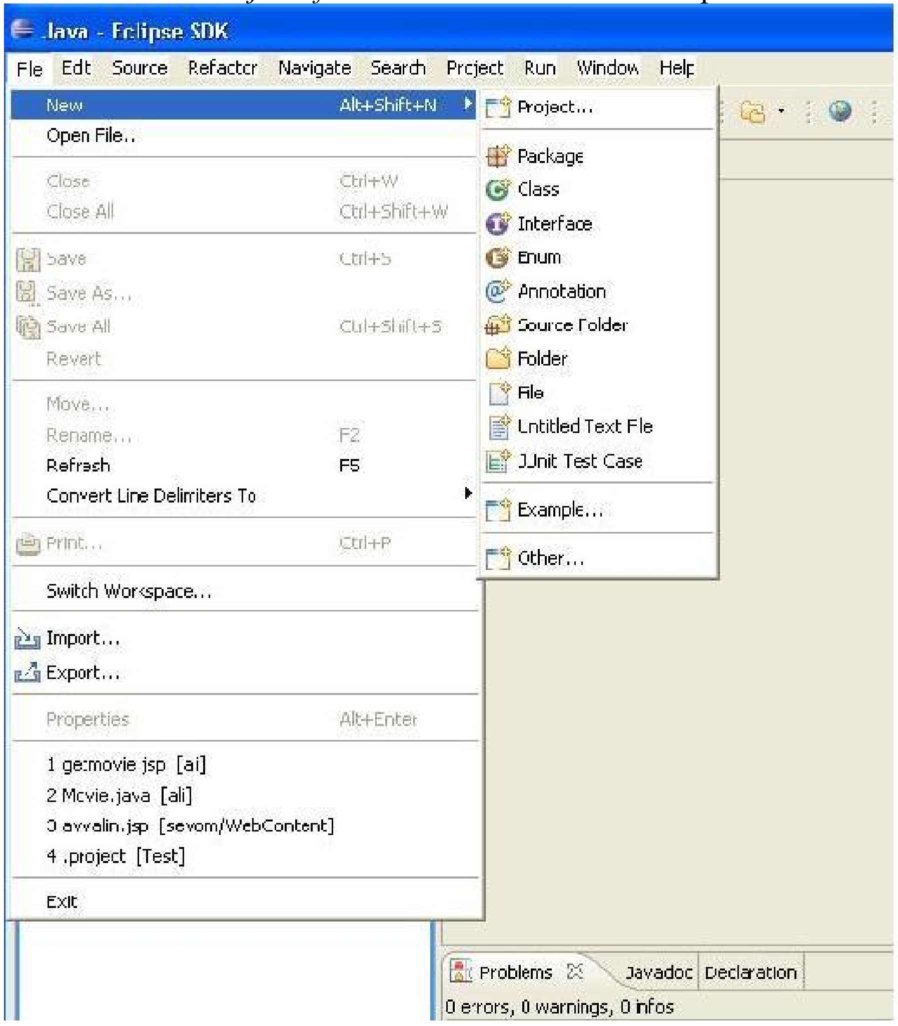
```
public class ChekcStr {
public ChekcStr() {}
public String convert(String str) {
return binarise(total(str));
}
public int total(String str) {
if(str=="") return 0;
if(str.length()==1) return ((int)(str.charAt(0)));
return ((int)(str.charAt(0)))+total(str.substring(1));
}
public String binarise(int givenstrvalue) {
if(givenstrvalue==0) return "";
if(givenstrvalue %2==1) return "1"+binarise(givenstrvalue/2);
return "0"+binarise(givenstrvalue/2);
}
}
```

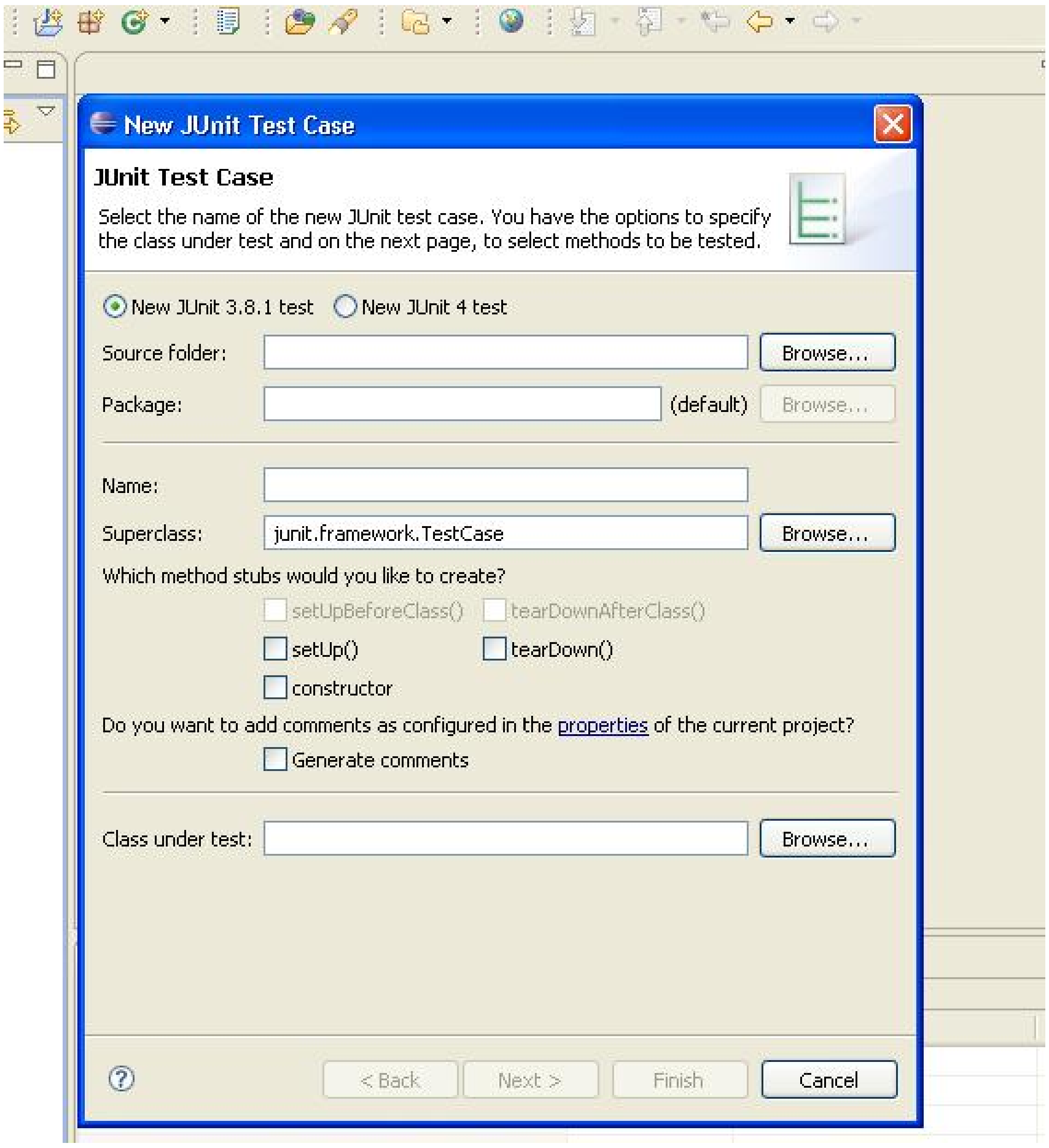
B. The next phase is to pass through JUnit wizards to create a test case. JUnit convention is that there is a test class created for every application class that does not contain GUI code. Usually all paths through each method in a class are tested in a unit test; however, you do not need to test trivial getter and setter methods. Now, select the directory (usually *unittests/*) that you wish to create the test case class in. There are five ways to create a JUnit Test Case Class:

1. Select **File > New > JUnit Test Case**
2. Select the arrow of the button in the upper left of the toolbar. Select **JUnit Test Case**
3. Right click on a package in the Package Explorer view in the Java Perspective, and select **JUnitTestCase**, or
4. Click on the arrow of the icon in the toolbar. Select **JUnit Test Case.**

1 ChekcStr.java can also be copied from /home/TDDC88/www-pub/labs/lab5/

5. You can create a normal Java class as shown in the Eclipse basic tutorial in Help menu, but include *junit.framework.TestCase* as the super class of the test class you are creating.




New JUnit Test Case

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 3.8.1 test) ○ New JUnit 4 test

Source folder: [ ] Browse...

Package: [ ] (default) Browse...

Name: [ ]

Superclass: junit.framework.TestCase Browse...

Which method stubs would you like to create?

☐ setUpBeforeClass() ☐ tearDownAfterClass()

☐ setUp() ☐ tearDown()

☐ constructor

Do you want to add comments as configured in the properties of the current project?

☐ Generate comments

Class under test: [ ] Browse...

< Back Next > Finish Cancel

- Check to make sure that you are creating the TestCase in the proper package.
- Give the test case a name.
  - Use the **Browse** button to search for a different super class. The default super class is *junit.framework.TestCase*.
  - Check which method stubs you would like to create. You can create a main method, *setUp()*, *tearDown()*, or a *constructor()*, but all of these are optional. A constructor is only run when the test case class is first instantiated, but the *setUp()* and *tearDown()* methods are run before and after, respectively, each test case is run. (**Also remember to click *Add Junit 3.8.1 to classpath* , if this hasn't been done**).
  - You can browse the application that you are creating for a class that you wish to test, or this could be left blank if you will generate the class while creating the test.

If you selected a "Class Under Test" you can click the **Next** button, otherwise click **Finish**. You will be able to select which methods in the class under test that you want to write test cases for. The method signatures will be created for you. Click **Finish**. The new test case class will be open in the editor.
C. The third step is creating a test case to check if the code performs its intended goal. Below, is a test case example. Copy the test code below into the eclipse and save as ChekcStrTest.java[2]

```
import junit.framework.*;
public class ChekcStrTest extends TestCase {
private ChekcStr ChekcStr;
public ChekcStrTest(String name) {
super(name);
}
protected void setUp() {
ChekcStr = new ChekcStr();
}
public void FuncInTest() {
int expected = 0;
assertEquals(expected, ChekcStr.total(""));
expected = 100;
assertEquals(expected, ChekcStr.total("d"));
expected = 265;
assertEquals(expected, ChekcStr.total("Add"));
}
public void testBinariseFunction() {
String expected = "101";
assertEquals(expected, ChekcStr.binarise(5));
expected = "11111100";
assertEquals(expected, ChekcStr.binarise(252));
}
public void testTotalConversion() {
String expected = "1000001";
assertEquals(expected, ChekcStr.convert("A"));
}
}
```

D. Every thing is ready to run the test case against the program. There are three ways to run JUnit Test Cases or Test Suites.

- You can right click on the test case class or test suite class and select **Run As > JUnit Test.**
- You can select a test case or suite and click the arrow on the icon or select Run from the toolbar, and select **Run As > JUnit Test.**
- You can select a test case or suite and click the arrow on the icon or select Run from the toolbar, and select **Run...** From here you will create a new JUnit test configuration, and name it. You can choose to run a single test case, or run all test cases in a project or folder. You can see:

2 ChekcStrTest.java can also be copied from /home/TDDC88/www-pub/labs/lab5/

```
FAILURES!!!
Tests run: 2, Failures: 1, Errors: 0
```
The test results indicate that an error happened in the function.

Task Two: Lab Assignment

Personal Number

Here you are supposed to write a class that represents the Swedish personal number and then a test case for it. A Swedish personal number consist of 10 digit number: YYMMDD-XYZC or YYMMDD+XYZC.

- The "+" shows that the person is hundred years old, or more. The "-" is changed to "+" the year the person turns 100 years old.
- The first six digits represents the date of birth with year, month and day, for example 640823 which is 23rd of August 1964.
- The following three digits, XYZ, is a serial number, where Z represents the persons sex. If the person is a female Z is an even number and odd if the person is male.
- The last digit, C, is a checksum and is calculated in the following manner:
  - Multiply the the digits in the date and serial number with `2, 1, 2, 1,...`
    For example:
    ```
    6  4  0  8  2  3 –  3 2 3
    2  1  2  1  2  1    2 1 2
    12,4, 0, 8, 4, 3,   6,2,6
    ```
  - Add the resulting digits: `1+2+4+0+8+4+3+6+2+6=36`
    - If a resulting number is larger than 10, it becomes the sum of the digits, for example `12 → 1+2`
  - Take the last digit from the sum (the 6 in the above example) and subtract it from 10.
    In the example it will be `10-6 = 4`, the resulting number is the check sum C.
    - If the result is 10 then the checksum C is 0.

The definition from the Swedish tax agency can be found here: www.skatteverket.se/privat/folkbokforing/omfolkbokforing/personnumretsuppbyggnad.4.18e1b10334ebe8bc80001502.html

The Personal number class shall have the following methods:

- getDate() should return the date of birth, YYMMDD in some form.
- getYear() only return the year

- getMonth() only return the month
- getSex() return if the person is male of female (returning just Z is not allowed).
- getCheckSum() return the checksum digit
- When instantiating a personal number class, give the personal number to the constructor as a parameter.
- It shall not be possible to instantiate an invalid personal number. If the instantiation fails an exception shall be thrown.
- Divide the problem of checking if a personal number is valid into sub-problems.

You are free to choose the types (integers, strings etc.) for your class by your self, still you shall be able to motivate your choices. If needed, you can add more member functions, it is then a good idea to make unit tests for them!

Hints:

- Do not forget to check the number of days in a month and if it is a leap year.
- To be able to call a member method belonging to an uninstantiated class the method has to be static.