M. Roggenbach, CS-135 – Lab Class 4 – 5/3/19

- To be solved in groups of two.
- To be ticked off in one of the labs of your house on Monday, 24.2., or Monday, 2.3.
- For being ticked off on Monday, 2.3., you need to have your solution ready at the begin of the labclass.
- You can obtain two marks by solving this sheet.
- Each completed task gives you one mark.
- All group participants need to be present to be ticked off.

This lab is about Boundary Value Analysis.

□ Task 4.1

Programming in "JUnit"

The purpose of this task is to understand JUnit more thoroughly. JUnit simply automates tests. Test cases are nothing but data: the inputs and expected results in a test case are simply pieces of data. JUnit allows you to write test cases down and execute them. The language that you use to write your tests in JUnit is Java; thus tests in JUnit are simply pieces of Java code. The tests can be simple pieces of Java code; or more complex pieces of Java code.

So far in the course, a test suite is nothing else but a table, where each row encodes a test case. We have encoded these in JUnit as separate tests. However, as tests are simply Java code, we could program them so that they look more like a table. We can capture a test case (i.e., data) in a datatype (i.e., a Class) within Java.

1. Make a new Java Project in Eclipse and add JUnit in version 4 as a library.
2. Download the files `TriangleClassifier.java`, `SingleTest.java`, and `TestingWithArrays.java` from `http://www.cs.swan.ac.uk/~csmarkus/Tools/`.
3. Import these files into your Eclipse Project. Note `TestingWithArrays.java` is a JUnit test suite.
4. Consider the class `SingleTestCase`: It collects three values of type `int` and one value of type `Triangle.TriangleType` – i.e., it represents a test case for the triangle problem.
5. Consider the class `TestingWithArrays.java`: It encodes a test suite within the method `myTestSuite`. To this end it declares an array `tmp`, whose entries are test cases. Each test case is created using the constructor `SingleTest`.

   ```
   SingleTest[] tmp = new SingleTest [2];
   tmp[0] = new SingleTest(1, 2, 3, Triangle.TriangleType.NOT_A_TRIANGLE);
   tmp[1] = new SingleTest(1, 2, 3, Triangle.TriangleType.NOT_A_TRIANGLE);
   ```

   The method `ExecuteTestSuite` executes these test cases within a `for` loop. Note: This is what JUnit considers as the actual test as it is marked with the `@test` tag.
6. Run `TestingWithArrays.java` as a JUnit test suite.
7. Encode the four test cases from weak equivalence class testing from the lectures in the style of `TestingWithArrays.java` and run them.

Test Case	a	b	c	expected output
WN1	5	5	5	equilateral
WN2	2	2	3	isosceles
WN3	3	4	5	scalene
WN4	4	1	53	not a triangle

Material to show when getting ticked off: Your 4 test cases run; explain pros and cons of writing test cases in JUnit in this style with arrays (as opposed to writing a method tagged with `@test` for each one).

☐ Task 4.2

Equivalence Class Testing

This task involves using JUnit on a more realistic program. The specification and implementation use different notations, something that is common in industrial specifications. For example, the specification of the test cases is written in the form d/m/yyyy (where d and m may be 1 or 2 digits). However, the implementation has a much more wacky output format. You will have to write your test cases to deal with this.

1. Make a new Java Project in Eclipse and add JUnit in version 4 as a library.
2. Download the file `nextDateFunction.java` from `http://www.cs.swan.ac.uk/~csmarkus/Tools/`.
3. Import this file into your Eclipse Project.
4. Run the file. Read the code, understand the main program; particularly the inputs and outputs of the method `tomorrow` of the `NextDateFunction` class.

   Note: There are many bugs in the `nextDateFunction` class.
5. Encode the test cases in the table below (which have been created using equivalence class testing) in JUnit and test the method `tomorrow` with them.

   Note 1: Test the method named `tomorrow` (note: you should test only this method).

   Note 2: The program is so buggy that some of these tests when executed will actually produce no output. The first test case `T1` is an example of this.

   Note 3: You will need your tests to use the correct "language". The data in the table below is not the actual language that the method `tomorrow` uses. Try run the main method again and look at the output. You will need to make sure that the TEV (Testing Environment), in this case JUnit, and the SUT (System under test), in this case the method `tomorrow`, can effectively communicate. There is no point in providing expected outputs to JUnit which do not match the format that the method `tomorrow` produces. Be aware of the "error" messages that `tomorrow` provides, they are more specific than the test cases require.

   Final note: You cannot just blindly copy the triangle tests and expect them to work.

Test Case Name	Month	Day	Year	Expected Output
T1	6	14	2000	6/15/2000
T2	6	14	1996	6/15/1996
T3	6	14	2002	6/15/2002
T4	6	29	2000	6/30/2000
T5	6	29	1996	6/30/1996
T6	6	29	2002	6/30/2002
T7	6	30	2000	7/1/2000
T8	6	30	1996	7/1/1996
T9	6	30	2002	7/1/2002
T10	6	31	2000	Invalid Input Date
T11	6	31	1996	Invalid Input Date
T12	6	31	2002	Invalid Input Date
T13	7	14	2000	7/15/2000
T14	7	14	1996	7/15/1996
T15	7	14	2002	7/15/2002
T16	7	29	2000	7/30/2000
T17	7	29	1996	7/30/1996
T18	7	29	2002	7/30/2002
T19	7	30	2000	7/31/2000
T20	7	30	1996	7/31/1996
T21	7	30	2002	7/31/2002
T22	7	31	2000	8/1/2000
T23	7	31	1996	8/1/1996
T24	7	31	2002	8/1/2002
T25	2	14	2000	2/15/2000
T26	2	14	1996	2/15/1996
T27	2	14	2002	2/15/2002
T28	2	29	2000	3/1/2000
T29	2	29	1996	3/1/1996
T30	2	29	2002	Invalid Input Date
T31	2	30	2000	Invalid Input Date
T32	2	30	1996	Invalid Input Date
T33	2	30	2002	Invalid Input Date
T34	2	31	2000	Invalid Input Date
T35	2	31	1996	Invalid Input Date
T36	2	31	2002	Invalid Input Date

6. Correct the code such that all tests pass.

Material to show when getting ticked off: all 36 tests pass; the corrected method.

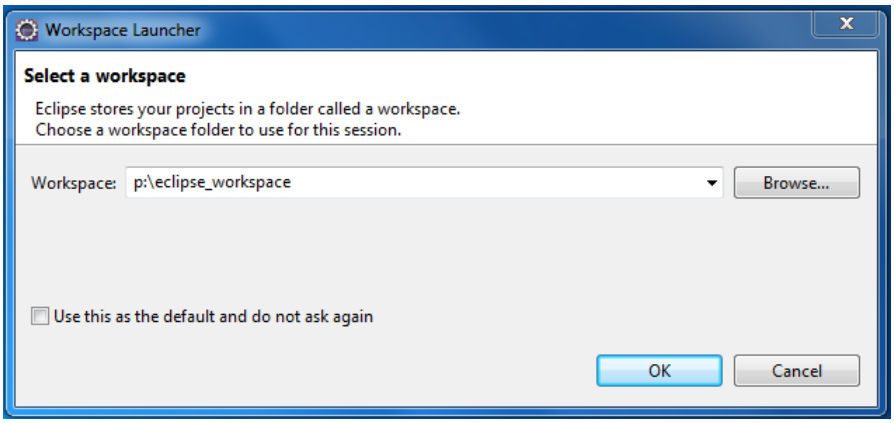
Computer Instructions

1 Making a screen-shot

Click on 'Start', type 'Snipping Tool' in the search field, press 'enter'. Use the tool.

2 Eclipse

Under the "Specialist Apps", open the folder "College of Science". Within this folder, open the folder "Computer Science". There, you find the program "Eclipse". When you start Eclipse you might be asked for the workspace path. This path should be set as follows:



2.1 Making a new project

1. Click `File` → `New` → `Project` → `Java Project`.
2. Typing a good project name i.e. `Sphinx`.
3. Click `Finish`.

2.2 Importing a file into a project

1. Expand your project, say `Sphinx` in the left hand panel (Package Explorer),
2. Right click the `src` folder, click import.
3. Select `File System` under `General`, click `Next`.
4. Locate the directory containing the `Sphinx.java` file, click `OK`.
5. Check the file, e.g. `Sphinx.java`, in the right hand list, Click `Finish`.

2.3 Running a program

You run a program, e.g., `Sphinx.java`, by clicking the play icon. This may bring up a wizard where you need to select to run a `Java Application`. You may need to show the `Console` view by clicking `Window` → `Show View` → `Console`.

2.4 Activating JUnit4 for a project

1. Right-click on your project and select `Properties`.
2. Click on `Java Build Path`.
3. Select `Libraries`
4. Select `Add Library`.
5. Select `Junit`.
6. Click on next, select the Junit Version `JUnit 4`.
7. Click `Finish.`
8. Click `OK.`