M. Roggenbach, CS-135 – Lab Class 3 – 26.2

- **To be solved in groups of two.**
- **To be submitted to blackboard by Monday, 4.3., 11am.**
- **Submission: Every student individually, one pdf file.**
- **Please mark clearly with whom you are working together: there should be 2 student numbers on top of your submission.**
- **No other formats than .pdf will be accepted!!!**

The purpose of this lab is to get some experience with the JUnit tool.

Note that there are computer instruction at the end of this lab sheet.

For submission you best create *one* file in Word. In this Word file, you included all screenshots required under a clearly visible headline of the task at hand. You then save this word file as a .pdf and submit this .pdf file.

☐ Task 3.1

Getting started with Eclipse (Indicative duration: about 10 mins)

Download, compile, and run the Java `Hugo.java` program from
`http://www.cs.swan.ac.uk/~csmarkus/Tools/`.

To this end, use the Eclipse IDE, i.e., you create a new project in Eclipse and import the program `Hugo.java` to it and run it in Eclipse – see the Computer Instructions at the end of this lab sheet.

Submission: Screen-shot showing the running program.

☐ Task 3.2

Install and Run JUnit (Indicative duration: about 15 mins)

1. Make a new Java Project in Eclipse and activate JUnit for it – see Computer Instructions.
2. Download the files `Hugo.java` and `TestSuite.java` from
   `http://www.cs.swan.ac.uk/~csmarkus/Tools/`
3. Import these files into your Eclipse Project.
4. Run the program `Hugo`. This main program simply prints the call to the method `produceHugo` (in Hugo.java) several times with various inputs.

   The `produceHugo` method attempts to implement the computational problem:

   produceHugo:
   Input: integer i
   Output: string "Hugo" if i is 1, "Erna" otherwise.
5. Run the JUnit tests in the file `TestSuite.java`.

   `TestSuite.java` encodes two tests for the method `produceHugo`. To this end it imports two packages:

   - `org.junit.Assert.*;` (as static) in order to write down "assert" statements, and

- `org.junit.Test;` in order to allow for the `@Test` tag.

`JUnit` encodes tests as methods that are tagged with `@Test`. We want to run the tests

Test Case Name	Input `i`	Expected Output
Test1	1	"Hugo"
Test2	2	"Erna"

on the method `produceHugo` (in the class `Hugo`). `Test1` is is encoded as

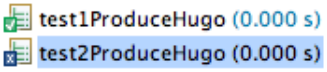
```
assertEquals("Hugo", Hugo.produceHugo(1));
```

`assertEquals` takes two parameters: the first is the expected output, in our case the string "`Hugo`"; the second parameter is the call to the method under test with the input value(s) as actual parameters, in our case the integer `1`. Note that in this case, you have to qualify the method with the name of the class it is defined within (i.e., the class `Hugo`).

`Test2` is is encoded as

```
assertEquals("Erna", Hugo.produceHugo(2));
```

When you run these tests, JUnit automatically produces the test verdicts. The first test passes, the second test fails. You obtain an output like



6. Fix the method `produceHugo` so that it passes both tests.

Submission: Screen-shot showing that both tests are passed.

☐ Task 3.3

Triangle Classification (Indicative duration: about 25 mins)

Consider the **Triangle Problem:**

Input: three integers a, b and c

Output: out of range, if c1, c2 or c3 fails
otherwise:
equilateral, if a=b=c
isosceles, if exactly two of the inputs are equal
scalene, if the inputs are pairwise different
not a triangle, if c4, c5, or c6 fails

c1	$1 \leqslant a \leqslant 200$	c4	$a < b + c$
c2	$1 \leqslant b \leqslant 200$	c5	$b < a + c$
c3	$1 \leqslant c \leqslant 200$	c6	$c < a + b$

It decides if – given the lengths of the three sides of a triangle – the inputs are valid, and if so, if these inputs belong to a triangle. If these inputs belong to a triangle, the triangle is classified to be equilateral, isosceles or scalene.

1. Make a new Java Project in Eclipse.
2. Download the file `TriangleClassifier.java` from `http://www.cs.swan.ac.uk/~csmarkus/Tools/`.
3. Import this files into your Eclipse Project.
4. Run the file `TriangleClassifier.java` to check the program is able to be run in Eclipse.

5. Consider the following 15 test cases (produced using Boundary Value Analysis):

Table 5.1 Boundary Value Analysis Test Cases

Case	*a*	*b*	*c*	*Expected Output*
1	100	100	1	Isosceles
2	100	100	2	Isosceles
3	100	100	100	Equilateral
4	100	100	199	Isosceles
5	100	100	200	Not a Triangle
6	100	1	100	Isosceles
7	100	2	100	Isosceles
8	100	100	100	Equilateral
9	100	199	100	Isosceles
10	100	200	100	Not a Triangle
11	1	100	100	Isosceles
12	2	100	100	Isosceles
13	100	100	100	Equilateral
14	199	100	100	Isosceles
15	200	100	100	Not a Triangle

6. Encode the 15 test cases in JUnit using the statement `assertEquals` for the method `classify` and run them. All the test cases should pass (i.e., there is no bug in the SUT).

   Hint: Note that the method `classify` takes in three parameters of type `int` and produces an output of type `TriangleType`. `TriangleType` has been declared as a Java enumeration (within the file `TriangleClassifier.java`). It consists of the constants `EQUILITERAL, ISOSCELES, SCALENE, NOT_A_TRIANGLE, OUT_OF_RANGE`. In order to work with these constants, you will have to qualify them, e.g., `TriangleClassifier.TriangleType.EQUILITERAL`.

Submission: Screen-shot showing that all 15 tests are passed and your JUnit code for these 15 test cases.

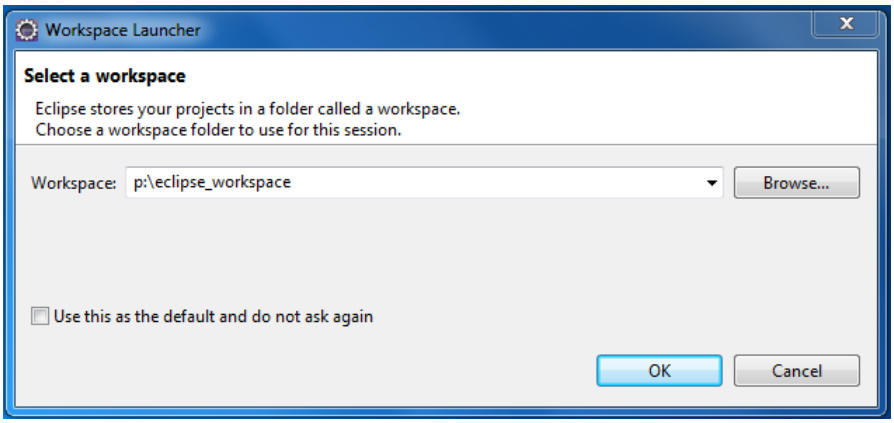
Computer Instructions

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.`