COMP3111: Software Engineering

Unit Testing

Learning Outcomes

- Be able to write unit tests
- Be able to generate and understand coverage reports

Setup: Create a local branch (Lab7Test) on your project folder. You will delete this branch after the lab is done. By now, you should already have a few classes in your project other than the one given in the skeleton. If not, you need to create one at this lab.

Since we are already in a late stage of the course, we will not provide all step-wise instructions with screenshot. Instead we will only give you a big picture and some key instruction. You really need to try or even guess what is going on to complete the lab. Discussion within your team is encouraged.

Lab Activity and Assessment

Lab Activity

1) Write two unit-tests on your own.

Assessment

Show your TA the following:

1) 100% pass of the tests.
2) JacocoReport that says branch coverage > 10%.

Note: Create a local branch (no need to push) called Lab7Test to avoid messing up your work!

Exercise 1: Writing unit tests

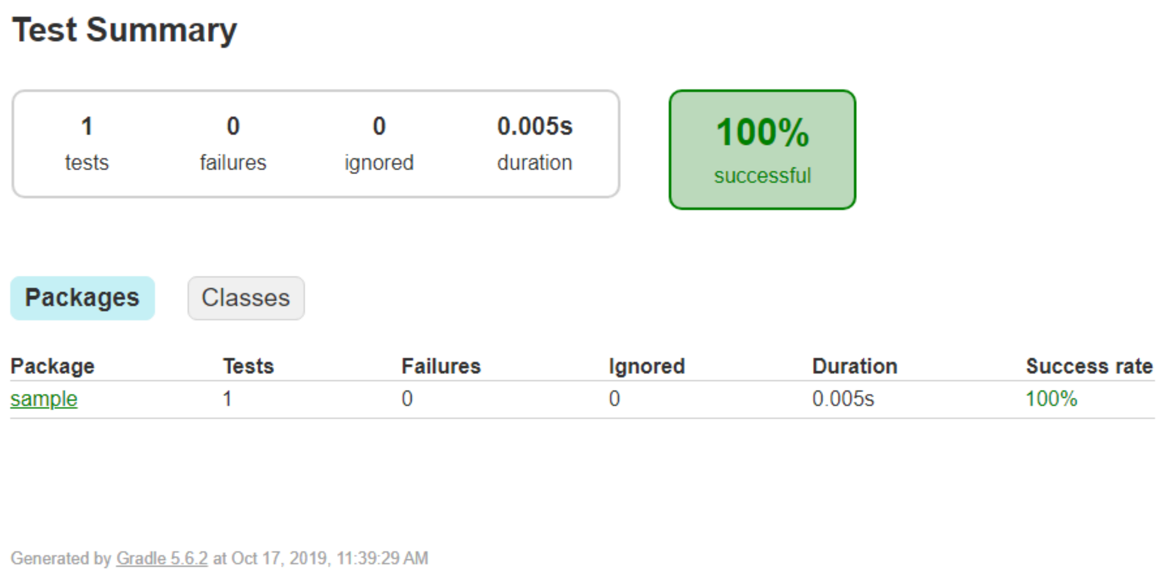
In this exercise you will need to write some JUnit tests, some very simple tests, to test one of your class. It is recommended that you test a getter function of a class you have already written. If you don't have such class, write one.

- You would need to use JUnit 4 to test this.
- All testing files should be placed under the directory 'src/test/java', which you need to create manually.
- You are advised to use **assertEquals** to check whether or not the class/function behaves as expected.
- After finished coding, you can try run the gradle task "test" under the category "verification".
- If your configuration is correct, the test will be either pass or fail. It shows error in Eclipse if your test fail.
- A report can be seen at: 'build/reports/report/tests/test/index.html'.
- Remember to annotate your testing function with @Test

A very simple unit test may looks like the following:

```
package sample;

import org.junit.Assert;
import org.junit.Test;

public class Test1 {

	@Test
	public void testGetName() {
		LabMonster l = new LabMonster("Kevin");
		Assert.assertEquals(l.getName(), "Kevin");
	}

}
```

A report that passes may look like:


Test Summary

1 tests	0 failures	0 ignored	0.005s duration

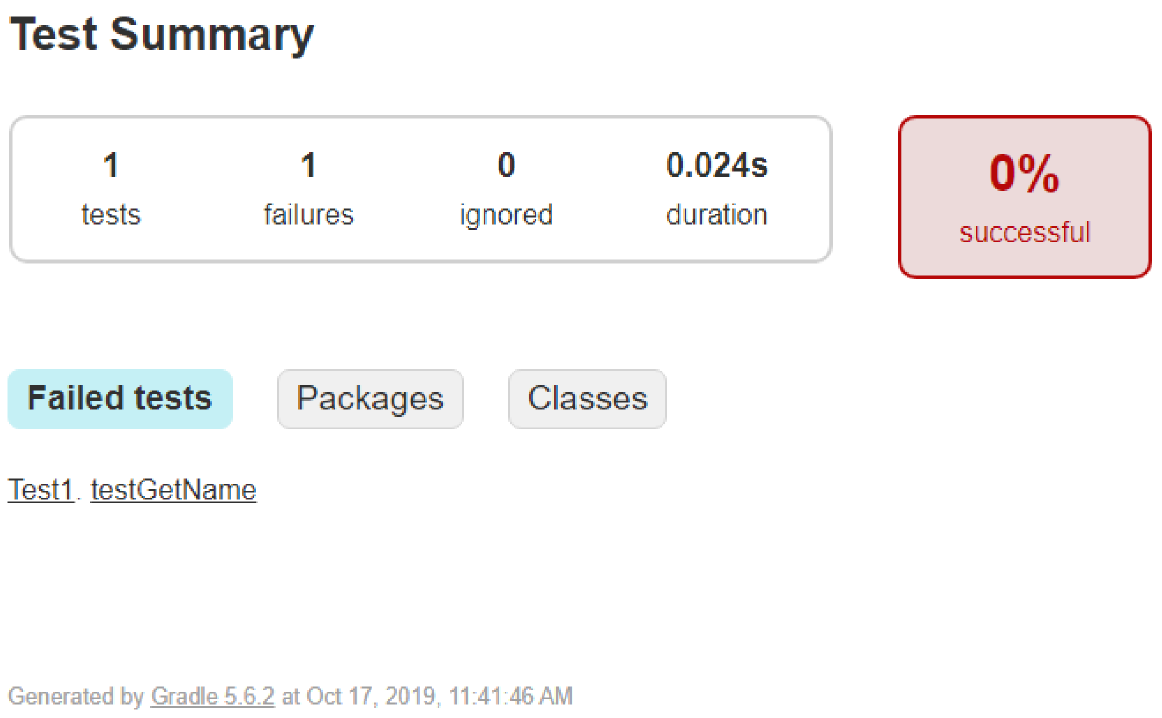
100% successful

Packages | Classes

Package	Tests	Failures	Ignored	Duration	Success rate
sample	1	0	0	0.005s	100%

Generated by Gradle 5.6.2 at Oct 17, 2019, 11:39:29 AM

A report that fails may look like:


Test Summary

1 tests	1 failures	0 ignored	0.024s duration

0% successful

Failed tests | Packages | Classes

Test1. testGetName

Generated by Gradle 5.6.2 at Oct 17, 2019, 11:41:46 AM

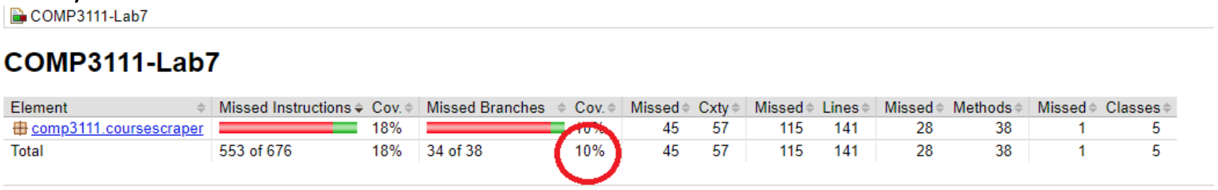
Usually there are much more tests (e.g. 20 – 40 tests in a COMP3111 project)
Apart from the command assertEquals, there are some there commands that you can use.

Please refers to: http://junit.sourceforge.net/javadoc/org/junit/Assert.html

Exercise 2: Generating coverage reports

- Run the Gradle task “jacocoTestReport” to generate the coverage report, after you have passed the test.
- The result of the report can be found at: build/jacocoHTML/index.html

It may look like



COMP3111-Lab7

COMP3111-Lab7

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
comp3111.coursescraper	🟥🟩	18%	🟥🟩	10%	45	57	115	141	28	38	1	5
Total	553 of 676	18%	34 of 38	10%	45	57	115	141	28	38	1	5

The red circled is branch coverage – measured in our project.

So what does this "coverage" mean? It measures how good are your test cases have been written or how many different branches of code that your tests have covered.

Further explanation:
There are two types of coverage: statement coverage and branch coverage.

- *A statement is covered if there is a test case that executes that statement*
- *A branch is covered if there are test cases that evaluate the condition as true and the condition as false.*

Note that 100% coverage does not mean that your code is bug free! For example, your test cases may only cover a small range of values. To make sure your code is bug free, you should always consider testing a wide range of values even if your coverage no longer changes. But, if the coverage is low (<50%), it implies there are not enough effort in testing.

Easter Egg: We provide you a TestFX example in the project skeleton code. It is not a mandatory task to use TestFX in your project (or this lab). It might however help you to pull up the coverage, enjoy.