

# CSCB07 – Software Design

## Lab 4

---

### Objectives

- Learn how to perform unit testing using JUnit
- Learn how to generate code coverage information using Eclemma

### Logistics

- This lab is worth 2% of the course grade, and it will be supervised by your TA during the tutorial session in the week of Oct 7<sup>th</sup>.
- If you encounter any problem while doing the steps listed in the following sections, ask the TA for help.
- Attendance will be taken during the tutorial. If you are unable to attend, please make a private Piazza post explaining why and submit the deliverables by the due date. Failing to do so might result in a 10% penalty.
- The lab should be done individually.
- The due date is Oct 13, 2024.

### Background

#### JUnit

Unit testing is a software testing approach where individual components of a software (such as methods) are tested. It is widely used in industry where numerous supporting tools exist. In this regard, JUnit was developed as a unit testing framework for Java. It is an instance of the xUnit architecture for unit testing frameworks (e.g. CppUnit for C++ and PyUnit for Python).

Testing with JUnit involves writing test methods (annotated with `@Test`) which check the code being tested using assertions. For example, the following test method checks whether method **distance** in class **Point** works properly when applied to points (0,0) and (1,0).

**@Test**

```
void testDistance(){
    Point p1 = new Point(0,0);
    Point p2 = new Point(1,0);
    assertEquals(p1.distance(p2), 1);
}
```

JUnit provides several assertions including:

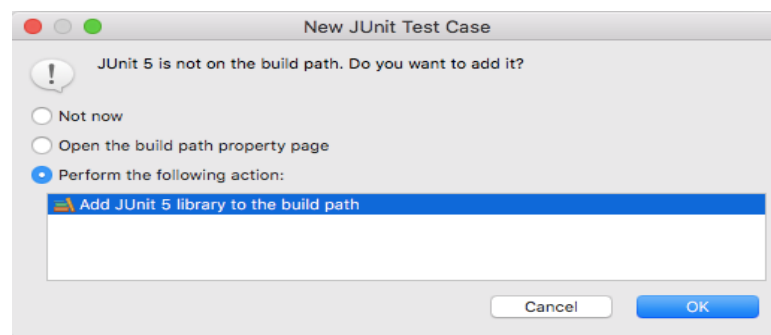
- assertEquals
- assertNotEquals
- assertTrue
- assertFalse
- assertNull
- assertNotNull

More information regarding JUnit assertions could be found at the following link:

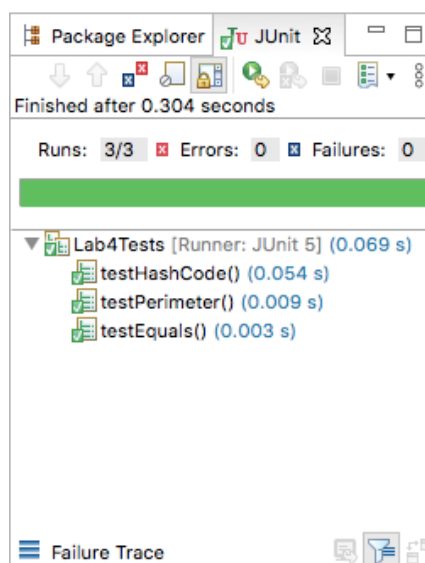
<https://junit.org/junit5/docs/5.0.1/api/org/junit/jupiter/api/Assertions.html>

To test the code of some package in Eclipse using JUnit, you can follow the steps below:

1. Right-click the package in Package Explorer -> New -> JUnit Test Case -> Provide a name and click "Finish" (a test class would be created where you could add your test methods). You might be prompted to add JUnit 5 to the build path. Press OK and continue.



2. Add the test methods
3. Run the test methods as follows: Right-click the test class in Package Explorer -> Run As -> JUnit Test
4. The results of the JUnit tests would be shown in the JUnit view as follows:

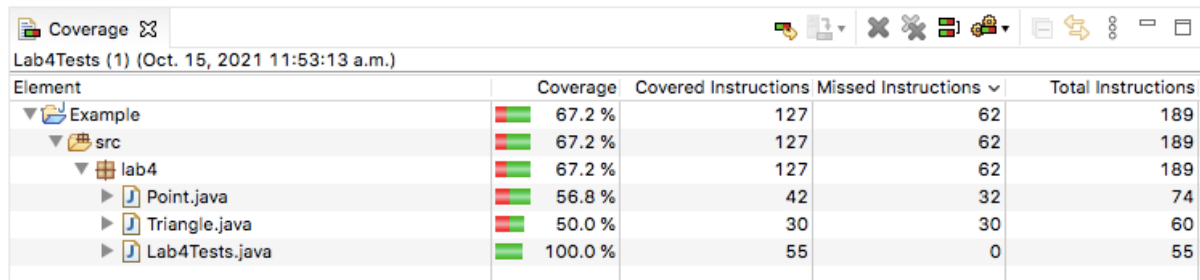


## EclEmma

EclEmma is a tool that generates coverage information for program elements such as instructions. To use EclEmma, the target application or JUnit test should be launched in Coverage mode. For example, to generate information regarding the code that gets covered as a result of running a test class, you can do the following:

Right-click the test class in Package Explorer -> Coverage As -> JUnit Test

The coverage information would be shown in the Coverage view as follows:



The screenshot shows the Eclipse Coverage view for a project named 'Example'. The view displays a tree structure of the project's source code, with coverage statistics for each element. The statistics include the coverage percentage, the number of covered instructions, the number of missed instructions, and the total number of instructions.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Example	67.2 %	127	62	189
src	67.2 %	127	62	189
lab4	67.2 %	127	62	189
Point.java	56.8 %	42	32	74
Triangle.java	50.0 %	30	30	60
Lab4Tests.java	100.0 %	55	0	55

In addition to these statistics, EclEmma highlights coverage results in the Java source editor using three colors: green, yellow, and red. More information regarding source highlighting could be found at:

<https://www.eclEmma.org/userdoc/annotations.html>

## Lab Description

You are required to provide as many JUnit tests as needed to achieve 100% instruction coverage for classes **Point** and **Triangle** that are provided with this lab. To test method **isEquilateral**, you can use the following points: (0,0), (Math.sqrt(5), 0), and (Math.sqrt(5)/2, Math.sqrt(15)/2).

Each test method should contain one and only one assertion. In addition, test methods that aim at increasing coverage by adding code that is not relevant to the assertion are not allowed. The following is an example of such test methods:

**@Test**

```
void badTestMethod(){
    Point p1 = new Point(1,1);
    Point p2 = new Point(2,2);
    int x = p1.hashCode(); //added to increase coverage, not relevant to the assertion
    assertFalse(p1.equals(p2));
}
```

## Submission

Upload the java file containing the test methods to "Lab 4" on Quercus.