COSC222 LAB 1 - Code Testing

In this first lab you will learn to use Eclipse IDE add-ons for Unit Testing (with JUnit) and Coverage Testing (with EclEmma).

Unit Testing with JUnit

Unit testing is an important part of quality assurance. The purpose of unit tests is to confirm that a class/method does what you claim it does and does not break under certain circumstances (i.e. wrong input, corner cases etc.).

You are given a Java project containing an incomplete implementation of Student, Teacher, Class and Administration Classes. Your task will be to complete the implementation and create additional unit test cases to test these classes.

The given Eclipse project can be loaded into Eclipse by extracting the project folder and following these menu choices in Eclipse:

```
Import > General > Existing Projects into Workspace > Select archive file
```

Then browse and find your .zip file where ever you downloaded it on your local machine.

This Eclipse project is a small example of four inter-related Java Classes: Student, Teacher, Class, and Administration. The project is also set up with the `Junit4` test suite. The starter code contains some JUnit tests of varying completion. There are comments within the classes with some description of the methods.

All the tasks you need to complete for this lab are marked by **`//TODO`** comments in the code.

1) Begin by looking at the class **`UniClassTest.java`**. This gives a simple example of how most tests are structured

> **`@Before`** methods are done before tests initializing useful variables, and objects that will be used in the test cases.
>
> **`@Test`** methods are where you test the functionality of your classes/methods. This example has one test per method but this does not always have to be the case.

A good rule of thumb to follow is that you should gain a better understanding of the method from each test, whether it passes or fails. This usually involves finding corner cases and testing what you expect to happen against the returned result.

`Assert` statements are what is used to validate the correctness of methods. In general, they are used to test what a method call is expected to return versus what is returned. There are many different type of assert statements. See the documentation here:

> http://junit.sourceforge.net/javadoc/org/junit/Assert.html

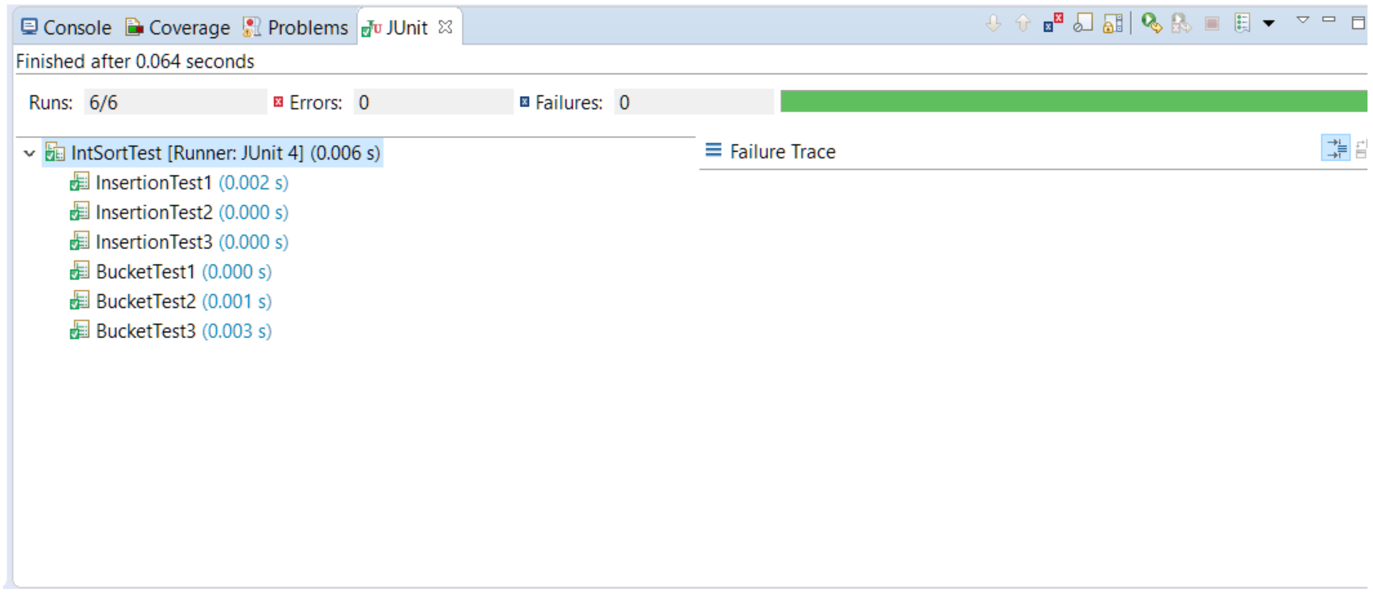
Here are some additional reference material for creating tests:

> http://www.tutorialspoint.com/junit/
> https://dzone.com/articles/junit-tutorial-beginners

2) The 2 classes **`Administration.java`** and **`AdministrationTest.java`** are two classes you need to complete. The tasks required in each are marked with a //TODO comment. You will also need to complete the **`StudentTest.java`** class and test both the getAge and getClasses method.

Coverage Testing with EclEmma

Unit testing ensures code quality while coverage testing ensure the quality of the unit tests. Coverage testing shows you how much of your code is being executed in a test class, a useful thing to know when trying to test your code.



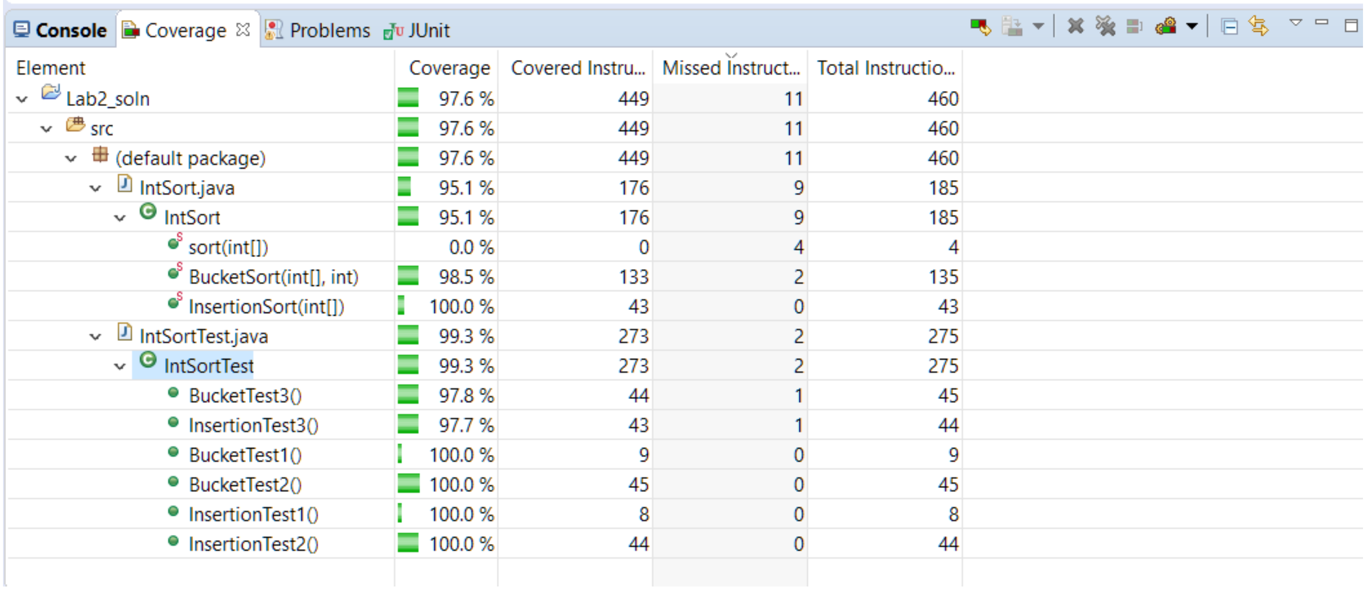
```
Finished after 0.064 seconds
Runs: 6/6    Errors: 0    Failures: 0

IntSortTest [Runner: JUnit 4] (0.006 s)
    InsertionTest1 (0.002 s)
    InsertionTest2 (0.000 s)
    InsertionTest3 (0.000 s)
    BucketTest1 (0.000 s)
    BucketTest2 (0.001 s)
    BucketTest3 (0.003 s)
```

Coverage testing works in conjunction with a regular test class. To have Eclipse measure your coverage, you need to add the EclEmma add-on from the eclipse market. To get the EclEmma add on go to the toolbar click help>eclipse marketplace from there search EclEmma and install. See:

http://www.eclemma.org/installation.html

Once you have the add-on you should be able to run coverage testing by right-clicking a test class and selecting run as coverage test. This will result in a summary of how much of your code was cove red by your test!



Element	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
Lab2_soln	97.6 %	449	11	460
src	97.6 %	449	11	460
(default package)	97.6 %	449	11	460
IntSort.java	95.1 %	176	9	185
IntSort	95.1 %	176	9	185
sort(int[])	0.0 %	0	4	4
BucketSort(int[], int)	98.5 %	133	2	135
InsertionSort(int[])	100.0 %	43	0	43
IntSortTest.java	99.3 %	273	2	275
IntSortTest	99.3 %	273	2	275
BucketTest3()	97.8 %	44	1	45
InsertionTest3()	97.7 %	43	1	44
BucketTest1()	100.0 %	9	0	9
BucketTest2()	100.0 %	45	0	45
InsertionTest1()	100.0 %	8	0	8
InsertionTest2()	100.0 %	44	0	44