Lab. Report #2 – Automated Requirements-Based API Unit Testing using JUnit

Group #: 17

Student Names: Matteo Morrone, Adeshpal Virk, Sam Farzamfar, Taimoor Abrar

Table of Contents

Introduction

In this lab we were tasked with performing automated unit testing. Specifically, we were to design unit tests based on the requirements for each unit. To develop and run the automated test code we used Junit. Often, the units being tested are not totally isolated, they depend on getting a value using a different method. For our unit tests we used Mocking to allow us to isolate each unit. Mock objects allow us to return any value we want when an external method is called. Automated unit testing allows us to ensure that all individual units are working and bug free.

Detailed description of unit test strategy

The method in which we developed our unit test cases was a very broad testing method. We looked at the boundary cases of each function. Furthermore we also tested negative entries to make sure that negative values also work in the function as an example of a partition of an equivalence class. Additionally, our tests calculating both the row and column total with multiple rows were designed to represent the entirety of that equivalence class, assuming that any number of rows would have the same effect on the results.

Mocking has the benefits of not requiring any external dependencies because a true unit test does not require any external dependencies. This is really helpful for databases and web services. The drawback to mocking is that it requires a lot more manual coding and can be difficult to use for some users.

Test cases developed

public void testCalculateColumnTotal_EmptyTable() Boundary value analysis

public void testCalculateColumnTotal_oneRow() ECT
public void testCalculateColumnTotal_multipleRows() ECT
public void testCalculateRowTotal_EmptyTable() Boundary value analysis
public void testCalculateRowTotal_oneRow() ECT
public void testCalculateRowTotal_multipleRows() ECT
public void testCreateNumberArray_oneElement() ECT
public void testCreateNumberArray_negativeElement() ECT
public void testCreateNumberArray_multipleElements() ECT
public void testCreateNumberArray2D_oneElement() boundary value analysis
public void testCreateNumberArray2D_negativeElements() Boundary
public void testCreateNumberArray2D_multipleElements() ECT
public void testGetCumulativePercentages_oneElement() Boundary value analysis
public void testGetCumulativePercentages_multipleElements() ECT

public void testContains_lowerBound() Boundary value analysis
public void testContains_upperBound() Boundary value analysis
public void testContains_inBetween() ECT
public void testContains_belowLowerBound() ECT
public void testContains_aboveUpperBound() ECT
public void testGetLowerBound_lowerbound() Boundary value analysis
public void testGetLowerBound_lowerboundNeg() Boundary value analysis
public void testGetUpperBound_upperbound() Boundary value analysis
public void testGetUpperBound_upperboundNeg() Boundary value analysis
public void testGetLength_SameUpperLower() Boundary value analysis
public void testGetLength_NegUpperLower() ECT
public void testGetLength_posUpperLower() ECT

How the team work/effort was divided and managed

We split the 10 methods into 4 parts, each student was responsible for 1 part and held one other person accountable for doing their part. Within the 4 parts we decided that each person had to do at least one test that required mocking. This assured that everyone would get an equal workload. Then we shared our tests with each other and tested whether each test was working as intended. If a test was not working as intended the person responsible for the test was made aware and resolved the issue.

Difficulties encountered, challenges overcome, and lessons learned

The first difficulty we encountered was our eclipse software missing the necessary components to run JUnit. After we remedied this issue Mocking proved to be extremely

challenging. For us to create mock objects we first had to research and understand what external methods our units were using and then understand what the desired output for these external methods looks like.

Comments/feedback on the lab and lab document itself

One thing our group noticed within the lab document itself was that we were told to expect 5 methods and 15 methods for org.jfree.data.DataUtilities and org.jfree.data.Range respectively. However, the files themselves contained 9 and 19 methods, and we were uncertain whether it mattered which 5 we chose from each class. The lab itself was very interesting as it gave us a glimpse into new kinds of testing that we could potentially be using in our careers in the future, as well as an introduction to new tools for said testing such as jMock. The instructions were relatively clear and concise, and there was a helpful introduction preceding the document explaining useful terms and concepts.