Learning Outcome 3

Dan Lindsay

Range of Techniques

A range of tests were implemented as planned, to cover the three requirements listed in `LearningOutcome2.pdf`. These tests assure that almost all basic functionality required by the PizzaDronz system has been implemented. The implemented system level test also puts stress on the PizzaDronz system, by feeding many orders into the system. This ensures that the system can still operate in under 60 seconds, which is a key requirement for the system to be functional.

Evaluation Criteria

The unit tests designed in the test plan are all equivalence partition tests, and so the following equation was used to calculate the coverage:

$$C = (\frac{N}{T} * 100)\% \tag{1}$$

Where C is the coverage, N is the number of partitions covered by the executed test cases, and T is the total number of partitions identified. Secondary values of Class, Method, and Line coverage have been computed by Intellij and are included in the results section of this supporting document. The expected coverage for this section is 100% as I plan to implement tests for all identified partitions in the test plan. Failing this, a coverage of between 75-99% would be acceptable as any lower would result in over a quarter of identified partitions being unfulfilled. The partitions identified in `LearningOutcome2.pdf` were all derived from requirements which were considered crucial to the correct functionality of the PizzaDronz system. Over a quarter of these partitions being unfulfilled could therefore result in requirements unknowingly not being met.

For the identified system level test, being that the system can operate in under 60 seconds, there were no partitions. The coverage for this system test is simply the line coverage provided by Intellij, as it gives a good estimate for how much of the entire system is being tested. As tests for the JSON data handling system were not written, there are elements of the RestConnection class (where input validation occurs) which are never reached and therefore the line coverage will not be 100%. Nevertheless, a coverage of over 85% is expected, as the rest of the system should be reached by both the unit tests and the timer system test.

Results of Testing

The following results are derived from the *InputHandlingTests.java*, *LngLatTests.java*, *OrderValidationTests.java*, and *SystemTest.java* files in the provided Github repo. The coverage results from running these tests has been formatted into the tables featured below.

System	Partitions Identified	Partitions Tested	Coverage (%)
Order Validation	25	25	100
Coordinate Management	7	7	100
Input Validation	3	3	100

Table 1: Primary Values for Unit Test Evaluation

System	Class Coverage	Method Coverage	Line Coverage
Order Validation	100%, (1/1)	100%, (6/6)	100%, (92/92)
Coordinate Management	100%, (1/1)	100%, (5/5)	93%, (27/29)
Input Validation	100%, (3/3)	100%, (11/11)	67%, (64/95)

Table 2: Secondary Values for Unit Test Evaluation

System	Class Coverage	Method Coverage	Line Coverage
PizzaDronz	100%, (11/11)	100%, (62/62)	88%, (386/435)

Table 3: Primary Values for System Test Evaluation

Additionally, the following figure provides proof of the test partitions for the Coordinate Management and Input Validation systems passing, as well as proof of the PizzaDronz system completing in under 60 seconds.

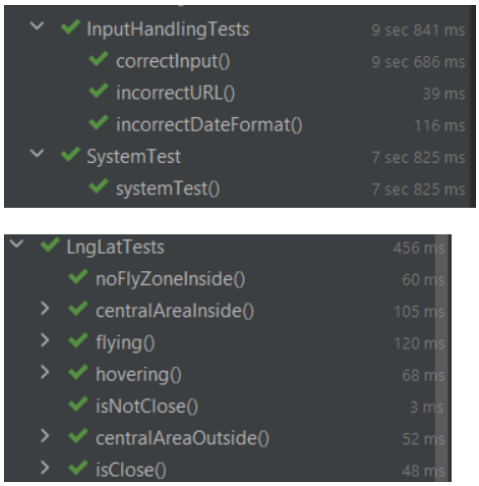

Figure 1: Input Validation, Coordinate Management, and System Tests

The following figure shows proof of the Order Validation unit tests passing. Note there are more than 25 tests, this is due to some partitions being split into multiple tests, but still being essentially the same partition (such as "TestTotalPriceIsNegative" and "TestTotalPriceOnePizzaNoOrderFee" being variants of the total cost being wrong).

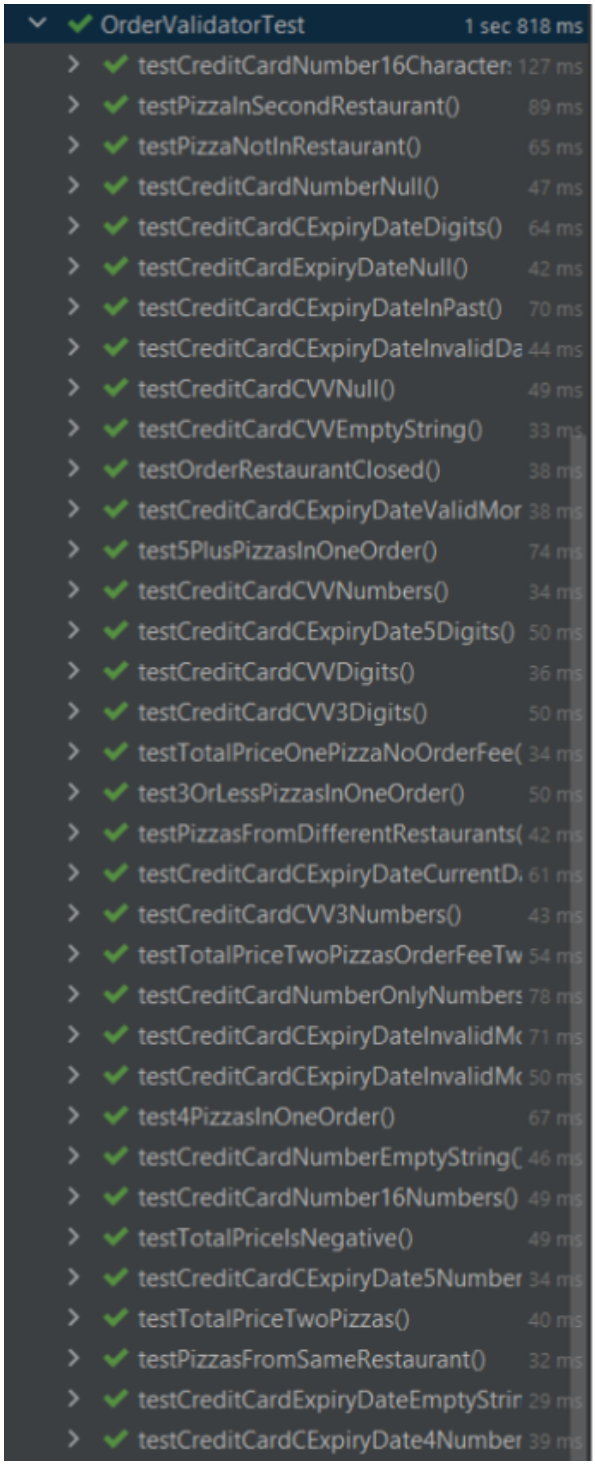

Figure 2: Order Validation Tests

Evaluation of the Results

From Table 3, one can see that the System Test provides 100% coverage in both Class and Method coverage, and achieves 88% Line coverage, which is well within the accepted coverage set out in the Evaluation Criteria section even with no tests implemented for the JSON handling parts of the system. From Table 1, one can see that all identified partitions were implemented, and from Table 2, one can see that the actual line coverage for the first two systems were either 100% or close to it. In the case of the Input Validation Line Coverage metric, the reason for a much lower line coverage rests in the location of Input Validation in the PizzaDronz system. Input Validation is done in 3 classes (shown by the class coverage metric), the *App.java* class, the *Main.java* class, and the *RestConnection.java* class. Breaking this down, in the *App.java* class the missing lines are related to the Alive status of the RESTful API. Specifically, when the REST service is down (i.e. Not Alive), the following lines are accessed:

```
if (!isAlive){
    System.err.println("Provided REST Server is not alive");
    System.exit( status: 0);
}
```

Figure 3: Example of code not reached by the implemented tests

This is a symptom of the lack of integration testing in the PizzaDronz system, as any tests which could have been written to test the effects of accessing a RESTful API which does not meet the specification were not implemented due to a lack of time. The ripple effect of the lack of tests can also be seen in the *RestConnection.java* file's Line Coverage, which sits at 60%, as a large portion of this file is dedicated to throwing errors when the RESTful API does not contain the correct information.

Despite the gaps presented in the Input Validation Line Coverage, the partitions identified in the Test Plan (`LearningOutcome2.pdf`) are all met, and as such the crucial requirements for the correct functionality of the PizzaDronz system are implemented and rigorously tested. Additionally the System level requirement, that it must operate in under 60 seconds no matter what, is also implemented and tested as according to the test plan. These test suites also provide an excellent jumping off point for any further in-depth testing of the PizzaDronz system, which another developer or testing team would benefit from.