**Junit Tests Journal**

Norvensky Francois

Module 7-2 Project Two Submission

CS-320 Software Test Automation & QA

Tanisha Jacks

Jun 18, 2023

**Junit Tests Journal**

**Evidence of Passed Unit Tests**

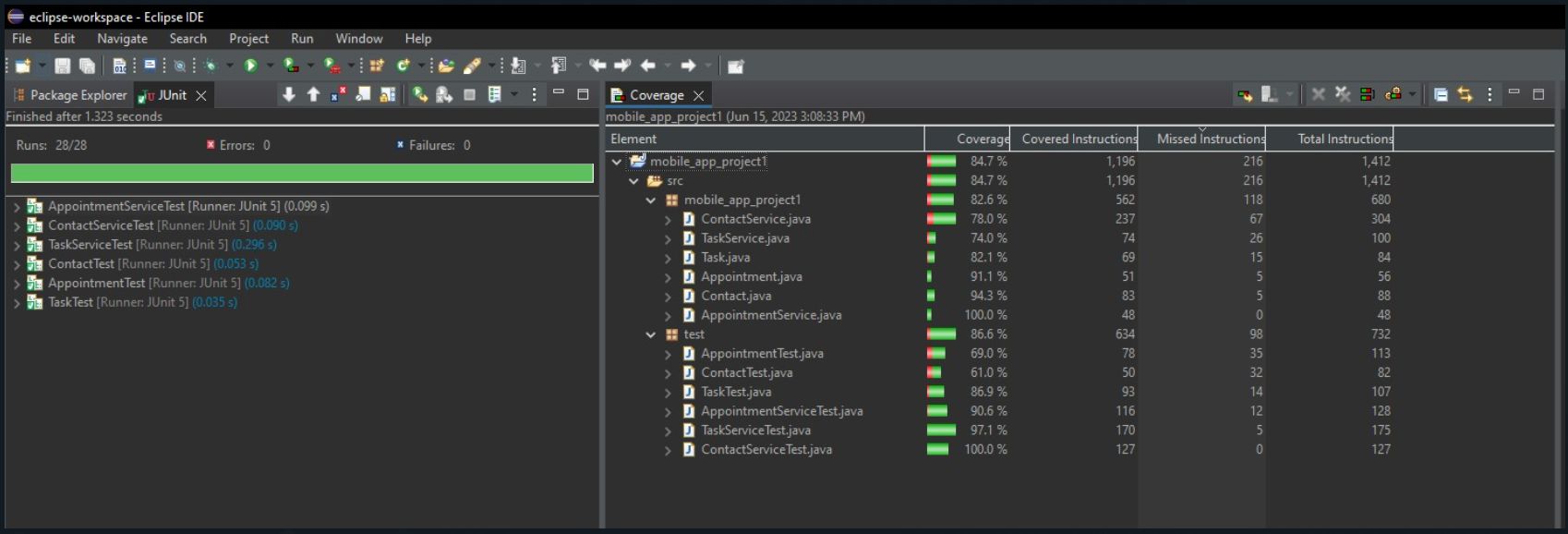
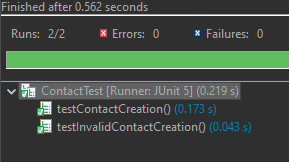
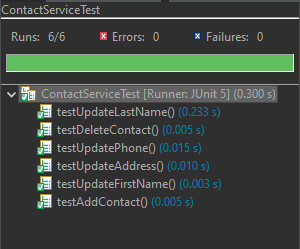


Figure : All tests were successful

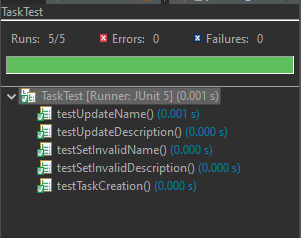
I used the rules in the requirements to write my tests. For instance, in the Contact class, I tested the correct way to create functions and whether the object will accept invalid arguments.

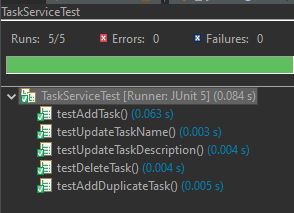


I decided to test whether each method works correctly for the Contact service class. I did not test invalid arguments.



I used the same strategy that I used on Contacts to test the Task class. I tested whether the methods work correctly and whether the object will accept invalid arguments. However, unlike ContactService, I tested invalid arguments on the TaskService. I tested whether the class can accept duplicates.



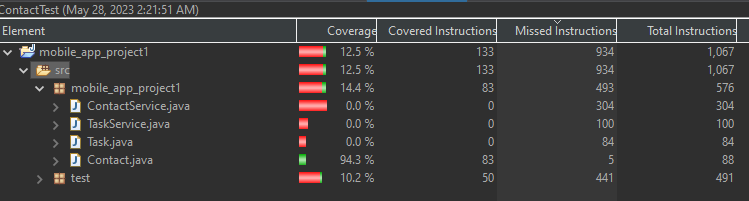


Therefore, I believe my tests were done according to the software requirements.

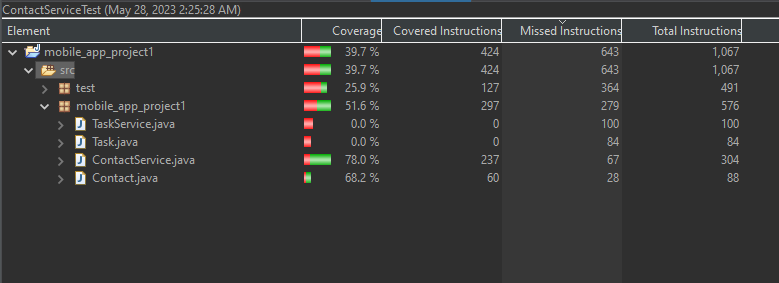
**Test Coverage**

I used the coverage tool to determine the level of coverage in my tests. The following screen captures show the test coverage for each class. I believe my tests are good. Although both tests for Service classes failed to reach the recommended 80% coverage, I was able to achieve the minimum 80% total coverage guideline.

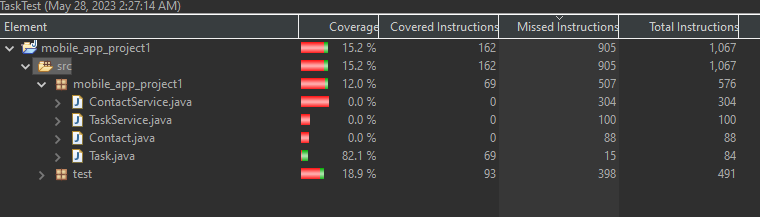
**Contact 94%**



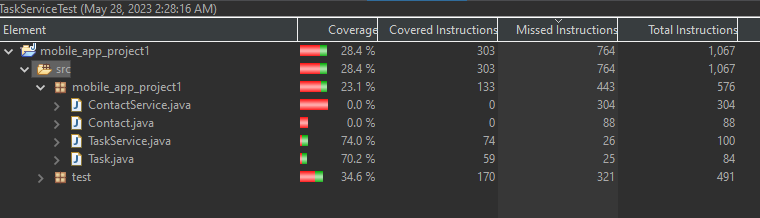
**Contact service 78%**



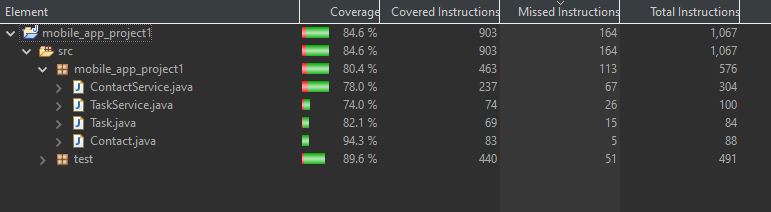
**Tasks 84%**



**TaskService 74%**



**Total Coverage**



**How was my code technically sound?**

I ensured my code is technically sound by doing input validation. I validated the contact names and numbers to ensure they did not violate the requirements. I also validated tasks inputs to ensure they followed the requirements.

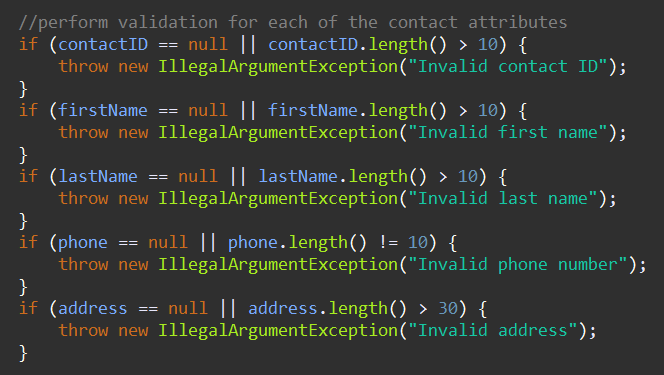


Figure : Input Validation

Access modifiers are another way to ensure the code is technically sound. I chose to use private modifiers on all classes to ensure the attributes are hidden from other classes. I then used setters and getters to control how these attributes are accessed. Finally, I used the final keyword on the **taskId** attributeto ensure it was not updateable.

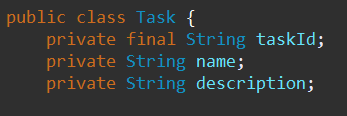


Figure : Access Modifiers

**How did I ensure my code was efficient?**

I ensured the efficiency of my code by using a fast data structure. I used a hash table data structure to store both Contacts and Tasks. Hash tables have an average case constant running time. This means all entries and retrievals from the data structures will be blazingly fast. The following screen captures show where I used hash tables.

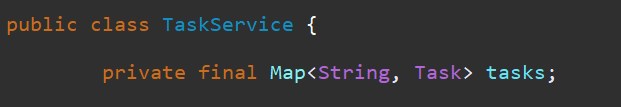


Figure Taks Hashmap

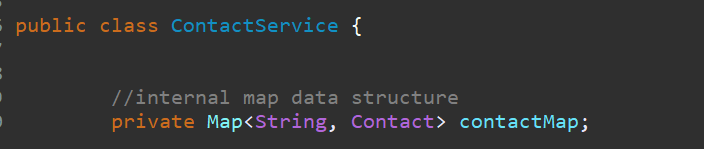


Figure : contacts data structure

**Reflection**

**Testing Techniques**

I used a similar testing method for all three module submissions. I used unit tests to ensure the software met requirements. These unit tests were automated. I used a testing tool called Junit Test, which performs unit tests in Java. My testing was also White box testing. This means I had access to the source code and could thus write tailor-made tests. Finally, I used static testing. This means I tested individual code components without running the program.

**Practical use and implications of used techniques**

Unit testing happens as the individual units of code are constructed. The testing technique can thus be used to find bugs early in the development process (Tilley, 2019). Additionally, one may wish to perform code refactors. It is important to perform unit tests to ensure the changes are implemented correctly. There are several implications of unit testing. They increase code maintainability by acting as living documentation. Unit tests also help to increase code quality as they enforce good coding practices.

**Tests I did not perform**

There are three important tests to assess the software. These are Unit, integration, and system tests. I only performed Unit testing for this project. However, it is important to discuss the other two tests. Integration testing is used to test whether individual software components work together as required. There are two types of integration testing. Component integration testing, which is conducted after unit testing, verifies the effectiveness of interactions and interfaces of components (Garcia, 2017). On the other hand, system integration tests focus on interactions between different systems. For instance, integrating a web application with a payment platform.

System testing is performed in the last stages of a software project. This testing technique aims to verify whether the whole system works according to the requirements. System testing also uses black box techniques. That is, the testers have no access to the source code. There are several types of testing at the system level. These include smoke testing, functional testing, performance testing, security testing, and compliance testing.

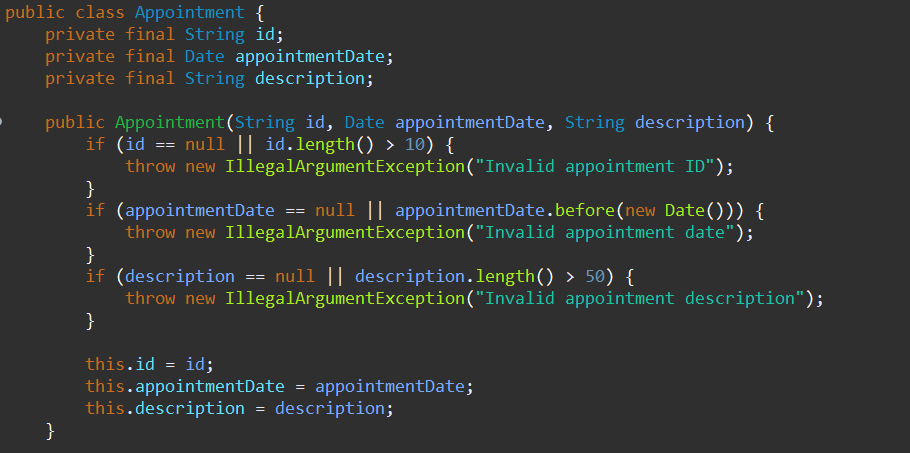
**Practical Use and Implications of unused techniques**

Integration testing is important when individual components start working together to achieve a goal. The software may also need to interact with other services, such as APIs. Integration testing ensures these integrations are seamless. This form of testing is especially important for large systems as it contributes to improved system reliability. The testing will ensure the system can endure different conditions. Another implication is reduced dependency risk. The engineers can see the degree of dependency between components and how changes in some components will affect others.

System testing is important for ensuring system-wide functionality. Users also get to test the system during the later stages of the software projects. System testing techniques can thus be used to conduct user acceptance testing (Garcia, 2017). One would also be interested in investigating how the system would behave in different production environment conditions. A system testing technique called stress testing can conduct these tests. Finally, system tests can be used by auditors when determining compliance with regulations. Examples of regulation include protecting data in transit and at rest and software quality assurance. Therefore, system testing ensures the system works properly, is secure, resilient, and can scale efficiently.

**Mindset**

I excersiced a lot of caution when writing the tests for the application. I was careful to ensure every requirement was addressed. For instance, when creating the appointment class, I ensured that the constructor checks each argument for validity.



I also ensured the code had some safety by using access modifiers. A user or attacker cannot change values in my program without using the setters and getters. I can then use the setters and getters to validate inputs.

I reduced bias by testing two scenarios for every function. Many developers will be biased in testing the best-case scenario. However, I tested the valid arguments and invalid arguments to see how the application would perform. For instance, I tested valid appointments and invalid appointments.



Figure : Testing Both Scenarios

There are many reasons why software engineers should exercise discipline. First and foremost, most applications are critical in nature. They are the lifeline for many businesses and individuals (Rombach et al., 2008). Therefore, errors in these applications could have massive economic and public safety implications.

Most developers will work within a team. The quality of software the team produces depends on team members being disciplined. Cutting corners in development could lead to inferior products that will destroy the reputations of multiple developers or the whole company. Finally, consistency and hard work ensure that software development projects are always on time and of the right quality.

**References**

Garcia, B. 2017. *Mastering Software Testing with JUnit 5*: Pakt Publisher.

Rombach, D., Münch, J., Ocampo, A., Humphrey, W. S., & Burton, D. (2008). Teaching disciplined software development. *Journal of Systems and Software*, *81*(5), 747-763.

Tilley, S. 2019. *Systems analysis and design*: Cengage Learning.