Summary

I tested every variable and method to ensure the code met the software requirements, such as unique IDs, variable lengths, and proper format. Unique IDs were a requirement for tasks, contacts, and appointments. This was tested by ensuring duplicate IDs could not be used for the objects:

Text

Description automatically generated Text

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

The required format for phone numbers was tested, ensuring they were 9 integers:

Graphical user interface, text, application, email

Description automatically generated

The appointment date was tested per the requirement that the date could not be in the past:

Graphical user interface, text

Description automatically generated

The requirements had maximum lengths for every variable, which were tested for being over their maximum length, for example:

A picture containing company name

Description automatically generated Logo, company name

Description automatically generated

A picture containing text

Description automatically generated

When it comes to testing coverage, 80% is considered reasonable, my tests had an overall coverage of 93.3%.

I made sure the code was technically sound by ensuring illegal arguments were thrown to avoid any undesired behavior or crashing. Example in Lines 29-33 in ContactTest:

A picture containing text

Description automatically generated

I ensured the code was efficient by using and testing the hash maps that contain the objects, testing their expected functionality (add, get, update, delete). Example in Lines 57-80 in ContactServiceTest:

Graphical user interface, text, application, email

Description automatically generated

Reflection

For each milestone, JUnit5 was used as the automated testing tool as well as a coverage tool. The techniques used with JUnit were input partition testing, decision testing, state transition testing, and use case testing. Input partition techniques will test for valid and invalid inputs to a program, for example, only accepting integers and not chars, or accepting dates but not strings. Boundary testing was sort of implemented, testing methods did not accept string variables that were over the maximum number of characters. Decision testing was used in cases of adding objects when they already existed as well as deleting objects that did not exist. Decision testing tests the program will make the correct decisions from the workflow in the program, if an object exists, it can be updated or deleted. I think use case testing could describe the updating methods of objects, getting the object – making sure it exists, then only being able to update the allowed fields. State transition testing was implemented to see if objects were being properly added and removed, state transition testing tests to see if correct changes in the system occur which can be implemented by inputs or outputs.

Techniques I did not use are performance and integration testing as well as decision table testing. Performance testing is best when using an automated tool which tests the performance of the system under test under various loads and usage patterns, stress tests, and integration tests. “Performance testing tools can find defects such as general performance problems, performance bottlenecks, memory leakage, record-locking problems, concurrency problems, excess usage of system resources, and exhaustion of disk space” (Hambling et al. 2015). Decision table testing is listing all input conditions that can occur and all of the possible outcomes and then incorporating them into a test. All techniques are important, of course, however, more coverage and complicated testing with all scenarios are going to be necessary for critical systems but not necessarily for a notepad or game application.

Exhaustive testing is not always practical, possible, or feasible for software products. Important things to determine when planning testing techniques are whether or not the system is a critical system, where every testing all scenarios is going to be necessary; if the system is expected to have high traffic, then dynamic performance testing would be necessary; and if it is a system with confidential or sensitive information where security needs to be exhaustively tested.

References

Hambling, Brian Morgan, Peter Samaroo, Angelina Thompson, Geoff Williams, Peter. (2015). *Software Testing - An ISTQB-BCS Certified Tester Foundation Guide (3rd Edition) - 6.3.4.5 Tool Support for Performance and Monitoring.* BCS The Chartered Institute for IT. Retrieved from  
https://app.knovel.com/hotlink/pdf/id:kt00UC2L91/software-testing-an-istqb/tool-support-performance