Peer review of Sebastian Airisniemi's (sa223gd) Workshop 2

Reviewer: Leif Karlsson (lk222rc)

Source: https://github.com/Saabiaan/Workshop-2/tree/master/src/workshop2

Running the application

Sebastian has provided a Java implementation of the registry system for the yacht club. No instructions were provided on how to run the application, but simply opening the project folder in Eclipse proved to be a working solution.

Upon running the application, a file not found exception is thrown as the member database file couldn't be found. This was remedied by creating an empty text file and updating the MemberRegistry class and its file property so it pointed to the newly created file.

Adding, removing and updating members work during the initial execution of the application. However, if two or more members have been added, the application fails to start again.

Viewing a single member works.

I'm not able to register a boat. The application presents different boat types to choose from, but no matter the choice, I am met with a prompt proclaiming "Invalid input. Please try again." This makes it difficult to test the rest of the boat related functionality.

Viewing members as a compact or verbose list works as expected.

Design and architecture

The application has a clear model-view separation. The Program class instantiates the view, and the model is not coupled to the view. The view contains no application logic, which is in line with the model-view separation principle (Larman 2005).

The unique id generation is well implemented, incrementing the id by one depending on existing ids.

The application design is undoubtedly object oriented, and objects are connected using associations instead of ids or keys.

Sebastian has had the GRASP principles in mind when designing his application, with high cohesion and low coupling seen throughout the code – an important part of object oriented design (Larman 2005).

Class diagram

The class diagram is a correct representation of the implementation, with valid UML notation. Associations and dependencies correspond with the implementation.

Sequence diagrams

Sebastian has provided sequence diagrams for all requirements, and here I will focus on two – adding members and listing members.

The sequence diagram for adding members is easy to understand and gives a clear idea of the operation. Correct notation is used. One thing that caught my eye is the returned value from MemberRegistry to Member. The author might take a look at how this part of the diagram corresponds to the implementation.

The diagram for printing a compact list of members follows the same patterns as the previous – easy to understand and gives a good insight of the implementation. However, the life time of the MemberRegistry as depicted in the diagram does not seem to correspond to the implementation. After the memberList is returned, the object is done.

The diagrams would definitely help a developer to understand the design and implementation of the application.

Conclusion

Sebastian has designed and implemented an application that follows essential object oriented design principles and adheres to a strict view-model separation. His diagrams are easy to understand and corresponds to the actual implementation.

No instructions were provided on how to compile and run the application, which would maybe have been preferable. The file handling could be more user friendly. Not being able to register boats needs to be addressed as well.

That said, I believe Sebastian's design and implementation passes the grade 2 criteria. His design is sound and the aforementioned issues are more concerned with minor implementation details than overall object oriented architecture.

References

Larman C., 2005. *Applying UML and Patterns*. 3rd ed. Prentice Hall.