prssbsl

Beppe Karlsson edited this page 20 hours ago · 2 revisions

Link

Test of Application

Before I go any further I should say that I have 0 experience of working with Java applications and I can only really understand an overview of Java code, though I am sure I have run a few in the past.

Sadly I was not able to get this application to work. I downloaded the runnable .jar version according to the instructions and it starts indeed. However I am unable to add any new members to the boat club. I click the button and input the information then hit save. The dialog box closes but I get no feedback on the input (no problem since user friendly wasn't part of the reqs) but the problem is that my newly added user doesn't show up in the list. I tried adding another user but can't seem to make that work either. From the instructions I am unable to figure out the correct path to go and as I do not have a Java IDE installed and do not wish to install one just to run a single app I am unable to test the application further.

Edit: After handing in the review I was contaced by Sarpreet who wanted to know what was wrong with the runnable .jar file. After going through everything with him I got it up and running without any issues. I tested the application and it had no bugs that made it crash. Simple error handling was done well as far as I can tell.

Implementation and Diagram Conformity

Since I am not a Java programmer I will not go into details about the code and what each method does. It is also not necessary for me to review the conformity between diagrams and implementation. The class diagram and the implementation does indeed seem to conform. The classes retain the same names as in the diagram and the seperation of Model/View is the same as in the class diagram.

As for the class diagram itself there are a couple of things of note. The relation between classes does not need to be annotated in the same way as when making a Domain Model class diagram. Meaning you do not need to use multiplicity in the same way. For example multiplicity at the *source* end is not necessary. [1, Chapter 16.4 *Ways to Show UML Attributes: Attribute Text and*]

Architecture

The Architecture does have a model view seperation, but without the use of a controller or other form of middleman to further seperate the two from each other. In this case it has meant a coupling that might be a little too close between the model and the view where the view requires the models to get information. With the use of a controller you can let the controller handle all the calls to the model and only return information the view needs rather than model specific information and objects. [2, L04 Model View Controller]

Otherwise the seperation works well and the model stays in the model domain.

The unique ID requirement is fulfilled by reading the registry database and getting the highest member id then incrementing that every time a new member is added. This ensures the member is always assigned a fresh id.

Quality of Implementation and Source Code

Overall the quality of the code is good. The methods are easy to follow with good naming conventions used throughout.

I did find an unhandled exception in a try catch block where the catch block is empty. The block can be found in the ReadWriteFile class at the very bottom on line 106-107.

Other than that I did not find any dead code segments or unused variables but then again I could not open the code in an IDE which would facilitate the finding of unused code.

Quality of Design

Overall the quality of the design is good. The classes are well thought out with seperate responsibilities keeping them short. The only class that is rather long is the one that handles user input but there isn't much to be done about that problem.

I do find that the view is a little too connected to the model like I have previously stated.

Encapsulation is used, although the program assumes that input to the Member model is correct without checking it when assigning the personal number for instance on line 44. The control has been done in the view already which might be in the wrong domain. Regardless if the view checks input or not the model should not assume that the input is correct as that reveals a hidden dependency.

Would the Diagrams help me as a developer?

Yes. They are clear and describe the associations between classes in the case of the class diagram.

The sequence diagrams do well to describe the flow of the application making it easy to use as a reference point.

Strong points

Relations between classes are very clear in the class diagram. The code in the implementation is easy to read and the object oriented design makes it easy to follow.

Weak points

The application could have used a mediator between view and model. Also the assumption that input into the model is correct is a weak point. Since the coupling is a bit too tight between model and view with the lack of a middleman, a changed view and a forgotten check would result in a faulty model.

Summary

To round things up I think this design and implementation has deserved a pass grade. There are a few minor things to fix to make the application stronger and more versatile. Once those fixes are done there should be nothing stopping them. The design itself is strong and reflects the implementation well.

References

1. Larman, Craig. 2005. *Applying UML and Patterns (Third Edition)*
2. Lectures, Object Oriented Analysis and Design at Linnaeus University 2016