Peer review for jh222qr by jr222er, lm222ix and aa222zh

Runnable version
After installing Visual Studio we can run your program in the IDE. The console is very well done and covers all requirements. However it was a requirement in the workshop to provide a runnable file, so keep that in mind next time.

Implementation and diagram correlation
We cannot open the Sequence diagrams or the Class diagram, please include png or other picture formats next time. We realise you are supposed to view them with Visual Studio but the version we downloaded (newest one) does not support it.

Architecture
The application follows a clear Model-View Separation according to the principles noted by Larman [13.7 p. 392]. The models are independant and do not rely on the View in any way.

Code quality
Even though the structure of the code is very good and readable we would really like to see some commenting to explain and separate methods.

You start a lot of variable names with “_” or “m_” which is good practice but you don’t seem to be consistent with it, at least when it comes to the “_” prefix. Maybe there is some reason you use it that is helpful to you that we are not aware of.

The source code follows a good practice of encapsulation.

Design quality
The Member class contains several methods regarding boats. One of which is used again in the controller class, this usage is unnecessary and both methods do the same thing. Instead you could have just instantiated the member and added it to the list in the controllers getMember method. The method name createBoat in the Member class can give a bad impression of what i actually does, since in reality all it does is makes an instance of a Boat and puts it inside a list. Other than that we feel like all the other classes have a strong cohesion and the elements inside the classes are strongly related to each other.

One of the strong points of using the method inside a method approach as explained above is that you can acquire a lower amount of coupling. However considering that most of the other methods inside the usercontroller are using dependencies linked to the Boat and Member objects, we do not see why this should be an exception.

Other than that the source code seems to use coupling where it is needed.

Looking at the old domain model from workshop 1 it is clear that the application is inspired from it.

Conclusion

Strongpoints

+ The application makes use of high cohesion in all classes except for the Member class. However, the design of the Member class does correspond very well to the domain model.
+ The naming and encapsulation of the source code is done very well.
+ The source code represents a clear Model-View separation.
+ Although it was not a highly prioritized requirement we were very impressed with the UI.

Weak Points

+ Needs instructions for how to run the program and how to view the diagrams.
+ The getMember method might be redundant.
+ The source code is in need of comments

We believe that the work done has passed grade 2. The application fulfills all of the requirements and has a strict Model-View Separation.

References

1. Larman C., Applying UML and Patterns 3rd Ed, 2005, ISBN: 0131489062