Workshop 2 Peer review 2014-10-17
By: Robert Roos and Mattias Karlsson
For: Jesper Holmström jh222xd@student.lnu.se https://github.com/Eldbollarna/UML.git

Is the requirement of a unique member id correctly done?

Yes, the solution to a unique member id is simple and elegant.

What is the quality of the implementation/source code?

The source code is well written and commented. There are some bugs however that we feel need to be looked to, like no input validation. This is not a requirement, but error handling is, for example entering a string into an int causes the app to crash now.

Also we found a very serious bug where you can't add two members with the same name.

The code compiles and "runs as it should".

What is the quality of the design? Is it Object Oriented?

The design is good and it is Object Oriented. The main view could be moved to a class of it´s own though and leave the Console class with the main method to just start the program and import the view(s).

As a developer would the diagrams help you and why/why not?

First off the class diagram looks good. We can clearly identify the relations between the different classes/layers and all the methods are clearly declared which makes it easy to understand. Good job.

Secondly the sequence diagrams. We really don't have that much to say about these other than they look really good and as far as we can see (and understand) looks correct. We like how you have added the "class" "Client", to clearly show where/when the scenario starts.

What are the strong points of the design/implementation, what do you think is really good and why?

First off, very good that you guys added some "readme" instructions, makes life easier for someone not familiar with the code. Moving on to the code itself you really have adopted the MV(C) well and you clearly master working with objects and the "interaction" between different layers. Meaning, you don't mix stuff up, making the model layer do something the view layer should do and such.

Small things such as using Enums for the boat types makes the code that much easier to understand.

Workshop 2 Peer review 2014-10-17
By: Robert Roos and Mattias Karlsson
For: Jesper Holmström jh222xd@student.lnu.se https://github.com/Eldbollarna/UML.git

What are the weaknesses of the design/implementation, what do you think should be changed and why?

To further improve the code we feel that the view layer could have been split into smaller classes. This would improve the readability (but that's just our opinion) and it would also make it easier to further build upon for the grades 4 and 5 for example.

Another thing is, even though it's not a requirement, it's hard to understand sometimes, like when entering PersonalNr how many nums are needed. We had to check the source code before figuring it out.

Also it's very hard to understand at first what's happening after you've "done something" like added a member. At first we thought the program had stopped, but it turned out we had returned to the main menu without any output telling us this.

When updating a boat, if the info is wrong the old boat is removed as a whole from the user-textfile.

Do you think the design/implementation has passed the grade 3 criteria?

Unfortunately no, there are some things that needs to be fixed before all the requirements for grade 3 are fulfilled.

- No members with the same name can be added
- No input validation at all, resulting in crash of app, error handling is a requirement
- Main menu is not looping, shows one time, then you have to guess if you are done with a task or not
- These are not requirements, but we really miss units (for input) for boat-length and such, and error messages
- When updating a boat, if any info is wrong, the old boat is completely removed.
- You can update a member that does not exist