PEER REVIEW RIKARD ÖSTERLUND

Is the Architecture ok?

Is there a model view separation?

Overall, there seems to be a clear model-view separation, as the controller accepts input from the view and sends input to it, independent of the implementation of the view. In theory, this means that the view should be easy to replace with a GUI, for example. However, the text to display is created in the controller, rather than the view. Perhaps it would be better to let the view handle exactly what the viewer sees and just let the controller decide when to display these messages.

The model is almost entirely separate from the view; however, in our review we noticed one point where this is not the case: the class MemberList.java, line 52:

view.TextInterface.displayLine(info.getMemberInfo(m, longFormat));

This line references not only the view but a specific, console implementation of the view, clearly a breach of model view separation. This line line (and any others like it) should be changed to ensure a good model/view separation.

Is the model coupled to the user interface?
No, its not coupled and the MVC separation is nicely followed with only the Controller accessing the model. However, see above for one place in the code where this is not the case.

Is the model specialized for a certain kind of IU (for example returning formated strings to be printed)
Yes, unfortunately the model returns unformatted but joined ready-built strings which in itself favours a pure text-based application such as a console app. Had the data returned from the model been in the form of objects, different types of user interfaces would have been much more easy to implement. The current model locks the app to be text based.

Considering the below comment found in the view source code:
/ Extremely basic user interface. I have tried to keep everything*
** reusable, every message that is printed is sent from the ctrl-package. */*
The correct thought is clearly there based on what is written in the above comment. This is as correctly stated in the comment indeed one of the strong points of the MVC Pattern [2].

One way to make the various MVC parts more flexible would be to remove the strings set Model / MemberInfo:

memberInfo.append("\nNumber of boats: " + m.getNumberOfBoats());

- If you in the future wanted to also add a webinterface, and would like to present this html “<h1>Number of boats<h1>” 5 - with the current code you’d have to perform a string split from what you get from the Model or in some other way reformat it so you can get the number and the presentation text separately.

[2.]

- **Are there domain rules in the UI?**

The view is not enforcing any domain rules onto the model so there is good clear separation here and domain rules are instead correctly implemented in the Model.

Is the requirement of a unique member id correctly done?

Yes it is. A note though:
It might be fragile as the uniqueness is dependant on a member counter stored in a separate file. As the member count grows and shrinks duplicate id’s could be generated in case of loss of the member counter storage. A combination with the members personal number (they should be unique) and/or by using the timestamp when the user was created would feel more solid as new members would never get the same id even if counter starts over from 0.

What is the quality of the implementation/source code?

- **Code Standards**

The code is very well-formatted. Indentation and formatting of the code is nicely consistent. The code follows Java standards very well. Including @Override notations on overridden methods, for example, is an easily forgotten standard that Rikard has taken care to observe. Overall, the code is very easy to read and we would like to commend the high quality of the code.

- **Naming**

Naming is crystal clear. Couldn’t be better!

- **Duplication**

No code duplication seems to occur.

- **Dead Code**

No dead code could be found.

- **Comments**

Comments are written for “the future programmer”, as a reviewer I appreciate that the comments are talking to me as a reader and not written for the coder henself. Comments are

also put in the right places, skipping parts that aren't necessary to comment (not "commenting for commentings sake", they serve more as a narrator to explain the overall logic and structure of the program which is very helpful.

One comment that could be reduced in size:

/ This section creates an object of class MemberDB and passes*
** the member list to it. MemberDB containts methods to read*
** the xml-file that stores members from previous sessions. */*

The first paragraph is unnecessary as it can be read directly from the nicely named and written code right below this comment.

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

- **Objects are connected using associations and not with keys/ids.**

Yes

- **Is GRASP used correctly?**

The responsibilities required to be implemented to fulfill the requirements are in place. The implementation could be more General in terms of the hardcoded strings discussed earlier, but the division of responsibilities into various functions is well made.

- **Classes have high cohesion and are not too large or have too much responsibility.**

Not much to complain about here either. Only thing is that the boats perhaps should not be handled in the UpdateMember class.

- **Classes have low coupling and are not too connected to other entities.**

Low coupling indeed. We couldn't find anything to criticize here at all.

- **Avoid the use of static variables or operations as well as global variables.**

Statics and globals are pretty much avoided except for the already mentioned unique id handling.

- **Avoid hidden dependencies.**

Hidden dependencies are hidden. If there are any we couldn't find them.

- **Information should be encapsulated.**

Not a single public primitive property. Only access is by public methods. Getters and Setters are implemented along with standard good practice. Variables and methods are private where they should be. Excellent encapsulation.

- **Inspired from the Domain Model.**

(should the question be “Inspired from the Domain Model.”? That is what we answer)
Class names are clearly inherited from the domain model which is great as it immediately gives an idea of what class does what and relates to what in the real domain.

- **Primitive data types that should really be classes (painted types)**

No there are not such a thing. For example, the memberlist is gracefully wrapped in a class instead of treating it as an array.

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

The class diagrams are have a good level of detail and makes relationships clear. Had the project been bigger the class diagram could have potentially gotten difficult to read if more is added, however at this delimitation the level of detail is helpful.

The sequence diagrams gave me an idea of how the application works before i had a chance to run it, and i felt they represented the final real world implementation well.

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

The implementation is easy to read and understand. The methods follows the single responsibility principle. Where the method names doesn’t fully explain their intention there are short, straight to the point comments. The logic is clearly separated and each module/namespace deals effectively only with their own business.

Properly setting to private fields and functions and keeping encapsulation tight.

The quality of the code is very high.

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

We believe that the pre-formatted strings that are being sent to the view would be better of passed on as raw data to be put together in the view. The reason is that any user interface could then easily be built on the engine. As an example: A html version would be hard to implement as the data that would otherwise be suitable to display in a table is already joined together as a string when reaching the view.

The implementation of the view also has some bugs, as Rikard notes, which shouldn’t be too hard to fix and would have improved the overall impression of the implementation.

Do you think the design/implementation has passed the grade 2 criteria?

We do. Job well done!

References:

1. 1dv607 L04, MVC, Slide “MVC-Problem”
2. 1dv607 L04, MVC, Slide “MVC The Good”