Workshop 2 – Peer review 2

Review Clarity
For documentation/diagrams
In your documentation and diagram you are using English as required without any spelling or grammar issues.

Your instructions for running the application covers Windows and Linux, unfortunately not Mac which I am using. However, I could get the application running by using the Terminal and java -jar ”program.jar” suggested for the Linux users. Maybe you should add those instructions for the Mac users.

Your diagrams are readable, have an ok layout and are consequent in their format for names. The class diagram uses the composition arrow correctly between the MemberRegistry, Member and Boat. However, I would suggest you use the association arrow to show the relationship between the classes instead, since the meaning of the composition arrow is ambiguous according to Tobias Ohlsson (2017).

From Slack, 1dv607-ooad, 2017-10-06:
”hobbe 10:52 AM
yes, basically composition and aggregation are special versions of association
10:54
however, the semantics (meaning) of both composition and aggregation are ambiguous so the advice is to steer clear from them”

Your sequence diagrams are almost correct. I followed the source code to make sure the diagrams really shows what is happening and came up with the following sequence diagrams, that shows the edits that needs to be made:

Sequence diagram – Create a member
http://www.plantuml.com/plantuml/png/nLAnRi8m4DtvYXjXA5BN225LxGWBLQgkHtoWgvpOd2-awTVNhWDjAAhOkbdltdkwzyvb886x9RmmTGMqF9rDqZD1Qwgt91lQSr3f2-26bT7o1qNIU7YDBeJcTRtQiRtEDwWuVwzj0NjIY4n8r1kCqHYelx00Et7rMGlgu444-rLaBh9iEL0JT4vMnj05F77m5dkeNAFYB7Xio4xAKI7tStYCGPG01wCBamqTMWeQo_6jNeYVgJ73u2JAWofRPaDmspUg5AODTNddMOurLlwkDxr8Clv3_9-d6O9Xp8tUf_YPaIvZN7M3GoUiTDejIlvUB-2HllSRbvp-yYS0

Sequence diagram – List members
http://www.plantuml.com/plantuml/png/ZLB1QiCm3BttAtJi3Wkii4iOXR4TMmQ3b5sLH0rYZcraPLtszRERDIKXWrviy_JUyvFb8f30_HOEJASSEbxVhT8nGNjgIv9tQZYez3cmPMKq_4EHD2qkqOMm_jZlZcoM-JKgPj-joQ4XXSY4HBt16EiQsZCMu2Yk7RMW3fv5iDz5vjCWs_vH4pHsLiIcStZbu0tsK3chuWnuj6HMnUH17ZD8Ia1ZBdVEN1uoU76jnqh12MmoE92KBj3CYAtlDAxN41TSKu0l4atnI-SCeVrt1YRk0nRMdglF5EwMOPo4bI2UsJPdaxTE5sH39Y7jn8wqQRzUKcGNcvdNhr_e5m00

The naming of classes follow the rules of thumb, being a noun in singular form.

The association between Member and Boat in your class diagram should use the variable name used in the source code, boats. And the association arrow should show the direction from Boat to Member with an open arrow head.

Lastly, in the class diagram, I would suggest you use the names of the methods parameters from the source code. For example, in the class diagram you use: editMember(String a, b, c), which doesn't tell me anything about what is being sent into the method. In the source code however you use: editMember(selMember, change, newData), which is far more informative.

For code

Indentation is correct in all source code documents. Formatting of blocks is mostly correct, but should be edited to up the readability of the code. As of now there are blocks of code which doesn't even contain so much of a blank line, this should be fixed since it now remarkably obscures the readability (Martin, p. 78, 2008).

Your naming of classes/operations/attributes and arguments are consequent and mostly uses real words. You should review your code and change, for example, method parameters like selMember, selId and newMem to selectedMember, selectedMemberId, newMember. This would enhance the readability since it uses pronounceable names (Martin, p 21-21, 2008).

These suggestions should be made in the View.java and MemberRegistry.java files and would make the code more readable and understandable. I would also suggest you divide the huge ui function in View.java into smaller functions. For example:
A user wants to create a member. if (choice1 == 1).
Call private function wantsToCreateMember which asks for the required input and sends it to members.registerMember(). This encapsulates the information, which is good because it hides the internal structure of the application (Larman, Glossary, 2004).

About comments, the comment in RunApp.java doesn't tell me anything. Does it mean that the application can be run without that class? Or what kind of information do you want to give the reader?

The comments in View.java and MemberRegistry.java are needed since the code is not self explanatory. They are all in english and readable. In MemberRegistry.java on line 391 there is commented out code. This should be removed if it is not used.

Please look over your methods to see if there is whitespace that can be removed. In some places, mainly in the end of functions, there are alot of whitespace that can make it harder to see where a method ends (Martin, p. 80, 2008).

Another tip is to order your methods based on when they are called (Martin, p. 84, 2008). Initialize in MemberRegistry.java for example is the first method called from View.java.

Overall I think the clarity is fine as it is now, but some changes has to be made to make the clarity even better.

Review Completeness

Your hand-in contains all required parts of the workshop. Good!
I was able to run the system and tested different kinds of scenarios on it to make sure it met the listed requirements. These are my remarks on the system:

1. Create a member – Application crashes when providing Firstname: 09, Lastname: 12, Personal number: kaka. Should get error message instead.

2. Create a member – Able to save a member with integers as name Firstname: 09, Lastname: 12, Personal number: 787682. Should get error message.

3. Create a member – Application crashes when providing Firstname: Sofia, Lastname: Kakan, Personal number: 9101270000. Should get error message instead.

4. Create a member – I am able to save several identical members. Should get error message, ”the user already exist”.

5. Edit a member – Wanted to edit one of the three identical members. Ended up with edits on all three of them.

6. Delete a member – Wanted to delete one of the three identical members. Application deleted two of them.

7. Register a boat – I was able to register a boat without specifying a boat type. (Empty input) Should get error message.

8. Register a boat – Application crashes when providing a letter instead of a number in length. Should get error message instead.

9. Edit a boat – Provided a non-existent boatId, message tells me the edits has been made. Should get error message.

10. Edit a boat – Able to exclude boat type. (Empty input) Should get error message.

11. Edit a boat – Able to exclude boat length. (Empty input). This crashes the application and I am not able to open it again. I had to manually edit the data.xml to get the application working again.

12. Delete a boat – Able to delete non-existing boat, by providing a boatId that didn't exist. Should get error message.

13. Delete a boat – Able to delete non-existing boat, by providing a letter as boatId instead of a number. Should get error message.

The application is missing some simple error handling for it to work without crashing. The changes that need to be made are stated in the list above.

Review Content

There is not a clear model-view separation. View.java works as the view, but the MemberRegistry.java does not work as a model, since it handles the formatting of strings sent back to the view for print out (see MemberRegistry methods listMembers and listMember). Changing those methods to not contain any formatting of strings would make the MemberRegistry appear as a model. Alternatively, you could move the methods to the view and only fetch the memberList from the model (Larman, 13.7 Guideline, 2004).

The design and implementation is somewhat object oriented. You have objects connected by association and not keys/ids. Classes have low coupling, since they are not too connected to other entities. You're avoiding the use of static variables and operations as well as global variables, and you use a natural design inspired by the domain model.

What makes it not completely object oriented is that information that should and could be encapsulated is not. To encapsulate: tidy up the View.java ui method as suggested earlier in this review. In the MemberRegistry.java, try putting duplicated code in one or several separate methods for easy reuse, mainly the code for getting and creating elements to write to the XML file. This would also make the code easier to read and understand.

Lastly your design artifacts are not completely in sync with the code. Mainly the sequence diagrams, which I have mentioned and suggested changes to in the review of clarity.

Reference

Larman, C. (2004). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development.* Boston: Addison Wesley Professional.

Martin, R. C. (2008). *Clean Code A Handbook of Agile Software Craftsmanship.* Boston: Pearson Education, Inc.