**Peer-Review WS2\_G2**

**General**

* (+) Runs well from the first try even on Ubuntu with wine-app.
* (+) User-interface is clear but it feels unusual to scan the input without pressing enter. (E.g. typing-mistake = directly wrong)
* (-) It also gives the possibility to save and load, which makes it unclear what file to choose for the user and otherwise the data is not saved at all. A predefined file is better.
* (-) Successful-Messages for saving/editing etc. would be helpful for the user.

**Diagrams**

**ClassDiagram**:

* (-) Transparent PNG is hard to read (no white background).
* (-) No packages Model/View/Controller (!) makes it harder to read and interpret!
* (+) Except the dependency from Controller - Member (try to avoid unnecessary dependencies in intermediate classes), it looks well encapsulated.

**SequenceDiagrams**:  
Input-Add boat:

* (-) Not all pre-calls need to be shown *(add-boat is a “sub-diagram”, e.g. when in the general-flow-SD addBoat() is called you can take a look at this diagram to see how the method call works in detail. All pre-calls how to come to the method just confuse)* -> start when the memberX is selected and getNextActionFromMemberView() is called [1,p224;1,236]
* (-) If you have an instance of something e.g. controller has memberX write it in the model: model::member:a\_member to refer to this a\_member again. [1,p.222]
* (-) Where is the member (inc his new boat) saved in the registry? (If it is direct referencing to the object in the list, which it isn’t in java, then nevermind)

Output: General control flow:

* (-) The task was about choosing one output method from the program and perform a sequence diagram for it. A general control flow is very abstract. A sequence diagram is based on a conversation between objects so by doing an abstract sequence diagram, There will be a big lack of objects conversations.
* (-) The program does not quit at the end of the operation.

**Architecture**

* (+) Model-View-Controller(MVC) Design applied well in the code just packaging is missing.
* (+) Model is well separated from the view. View could easily be replaced just an interface would have been nice.

**OO-Design and GRASP**

* (+) Good Object-oriented-design, e.g. Boat has its own class.
* (+) GRASP Patterns applied, e.g. Information expert - member has methods for adding boat, encapsulated information e.g. onlyReadList<Member> to view, no static variables beside public enums.

**Code**

* (+) Code-Quality looks good. Good naming, code standards and no/not much duplicate code.
* (-) Hard coded seeded data is not data-persistence. Choose a predefined file and then maybe give possibility to change it. (Much more user-friendly, as mentioned in **general**)
* (-) No dependency needed for controller-member. If you pass the member directly in the argument you spare out the dependency. (since you already have it in the view, GRASP Indirection) [1,426+]

**Our point of view**

A very strong point of the design is its simplicity. It is easy to understand and due to the list of possible user-actions to get an overview. The biggest weakness is the UI (Give the users more instructions and messages to confirm) and the error-handling but both not in focus of this submission. Another important point is to update the class diagram, to make it more clear by simply adding a background, adjusting the elements a bit better and put the classes in packages(!). So we would say after changing the diagrams the submission would pass grade 2.

**Bugs** (User-input bugs don’t need to be covered in this grade - still listed)

* Crashed when entering a wrong format personal-number

Reference section

1. Larman C., Applying UML and Patterns 3rd Ed, 2005, ISBN: 0131489062
2. Pascal R. and Franck V., UML 2 en action 4th Ed, 2007, ISBN: 978-2-212-12104-9