# Peer review Workshop 2

For: Markus Alshraydeh (ma223ik) and Daniel Hammerin (dh222gh)  
By: Ola Franzén (of222au)

### Testing the application

First of all I didn’t find any documentation on how to run the program, which for a Windows user might be a little troublesome.

When starting the program I spot some missing functionality – to list members. You can perform a ‘findmember’ operation, but then you also have to actually know the member’s id so I couldn’t test this. Furthermore, when entering a random id in this menu the application crashes without any error information. I also tried to add a new member but it also crashed before it could be added. But this is probably because I’m missing some drivers or other setup. So unfortunately I can’t test run the application much more than the main menu.

### Source code

I don’t have the tools required to compile the java source code, so I will look through the code.

The naming of the functions, fields and classes are in general good and easy to understand. There is some confusion about the MVC principles, more about that later. The code is also mostly well commented. There is mostly no duplication of code but one could argue that some reduction of code could be possible in the DataHandling classes (adding and changing uses mostly the same code).

There are some possible mistakes in for example the createMember function in MemberDataHandling class. When calling the createMemberId function you never catch the returned value, which makes it unsafe to say that the member id is unique.

### Architecture

The good parts are the Member and Boat model classes, which are clean and easily to understand and does what it should do. The other classes have some issues concerning separation into the MVC pattern. For example the Main class (which is the View part) handles calls and logic to the data handling classes, which according to the main separation principle in MVC as described by Larman [1, 209] is wrong. In the opposite way the MemberDataHandling, BoatDataHandling and SQLDAO classes handle some view specific actions, like printing out a message to the console. This is in similar way as described above wrong according to the MVC principles.

### Quality of design

Another concern I have is that in some functions the association to model classes is done by keys/id, for instance in the MemberDataHandling class, the functions changeMember(String ID) and deleteMember(String ID). A better way would be to only reference by association to the objects, and let the DAL classes handle the keys/id connected with the row in the database.

GRASP is used in some places well, like the database methods are separated into specific classes, as Larman exemplifies [1, 290]. However because of the mistakes in the MVC principles as described above, there are also many problems trying to conform to GRASP, like low coupling. As of now most of the classes have a dependency to many of the other classes.

### Diagrams

The class diagram is easy to overview and to understand the general thought of layout of the classes. They do miss the method body as described by Larman [1, 256-257]. There are also some dependencies missing, like the BoatDataHandling has a dependency to SQLDAO.

The sequence diagrams are although a bit simplified easy to understand and gives a good picture of the events that occur.

### Summary

Overall the actual solution could be very well executed, but there are some major problems mainly concerned to the MVC pattern, which is the weakest point in this. Solving this in a better way would probably make the solution all in all a good one.

The strongest point I think would be the technical solution with the database handling.

I think the design needs improvements before it can pass the passing grade.

## References

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