Peer review for gf222ca
Workshop 2

Architecture

- **Is there a model view separation?**

  Yes. But it doesn't seem like it have been done correctly.
  There is nothing wrong with the *View* really. The problems are the *Model* and the *Controller*. The model does not contain any getters and setters, it only contains methods with return values. And there are also some unrelated functions in the Member class.
  For example, the member model should not be responsible for writing and saving data to a file, since it has nothing to do with member model.
  There is a design problem in your application, but it's really hard for me to explain it properly without writing tons of pages.
  You can read about it in the course literature. I've pointed out the chapters below that explains it all.

  Read more about *"Indirection"* in the course litterature.
  (Chapter 25.3. Indirection - Applying UML and Patterns 3rd Edition)

  Also look up *"low coupling"* and *"high cohesion"* in the course litterature.
  (Chapter 17.8. A Short Example of Object Design With GRASP - Applying UML and Patterns 3rd Edition)

- **Is the model coupled to the user interface?**

  Yes. But i'm not sure if it's done right. It's really hard to read the code because it's messy. The reason is just like the previous question.
  I think there is some misunderstandings on the MVC structure.
  Otherwise the member model and boat model is ok, except for that the file handler is placed wrong. The data base should take care of that. If you would like to change type of storage, like from database to a text file, you'll have to go through several classes just to change that functionality. Which also defeats the purpose of the MVC structure.
  My advice to you is to read more about MVC. What Model is and what Controller is, and try to get a better understanding of the differences.

- **Is the model specialized for a certain kind of UI (for example returning formated strings to be printed)**
  - No..

- **Are there domain rules in the UI?**
  - No. The code only contains information that should be seen and controlled by the user.

- **Is the requirement of a unique member id correctly done?**
    - I don't know. I could not test the application since I couldn't get it up and running. I tried following the small quide, but it did not work.

Code quality

- **Code Standards**
    - I have no experience in Java. But to me the code a bit messy and it's hard to understand it. I've have been looking at it, but i'm not sure what all the classes does. I can't understand controller and what it's purpose is in this application.
      Nothing wrong with the view or the model classes. The controller is the biggest problem here.

- **Naming**
    - Naming is good in general. But there are small parts were only a letter is used or shortened words, which can make it harder to understand.
      Like *"personnnalNb"* does not say that much. It would have been much easier to understand if it contained valued words like for example *"personalNumber"*.

- **Duplication**
    - Since the MVC structure is not done right, there are a lot of code repeating here.

- **Dead Code**
    - No?

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

The code is object oriented, but the MVC structure is not done right. I been saying that all the time through this review. But that's because it's such an important aspect. Because if it's not done right, then there is no reason to use the MVC structure.
The biggest problem in the code is that it has a lot of low cohesions and high couplings. Instead it should be designed with low couplings and high cohesion. There are simply too much dependencies that unfortunately breaks the design of the code.

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

The class diagrams does not help me that much. Sure, it gives a good overview of what each class contains. It might come in handy if a code is badly written. Otherwise I see no point of it really.
The sequence diagram on the other hand is really helpful.
When looking thru the code you can understand what it does and what happens. The problem is that it's much harder to see the whole picture of how the application work. Because when looking at code in classes you only see smaller parts of the whole system.
The sequence diagram helps me understand how the whole application works and how all the parts of the code working together.

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

The view. Because there is nothing wrong with it. It's simple and very straight forward to understand. And it does what it's supposed to do.

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

I have been mentioning it before. The MVC structure is done wrong. Not everything is wrong, but some functionality is placed wrong and there are too many classes coupled to each other that causes too much dependencies instead of being more focused.

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

I don't know.