PEER REVIEW HENRY PAP ET AL.

Is the Architecture ok?

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

Yes, the view imports the controller class YachtClub and all the application logic is handled by this class rather than in the view.

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

No, the models are imported but only their toString method is used in order to show their information.

- **Is the model specialized for a certain kind of IU (for example returning formated strings to be printed)**

Yes, there is no method in the models that returns the models information. Rather, they have a toString method that returns formatted strings specialized for a console UI.

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

Yes. There are some things going on in the view that would be more suited to deal with in the controller. We would suggest an object encapsulating the input data to be transferred to the controller for further processing. In the view we can find things such as: Pnr “pnr = new Pnr();”, which should be handled in the model [1, p329].

Is the requirement of a unique member id correctly done?

Yes, there is a genereteID() [sic] method in the YachtClub class that generates a random string of length n of characters [a-z][A-Z][0-9][!?#%&], with a check to ensure that this random string is not repeated in the database, ensuring a new, unique ID each time.

What is the quality of the implementation/source code?

- **Code Standards**

The code is very messy in parts. It would be better to follow some sort of code standard with getters and setters at the end of files, proper indentation of getters (rather than having them entirely on one line), each variable declaration on its own line for clarity, the constructor being the first method after variable declarations, etc. These standards are followed in some files, but not others, and it would be much better for clarity (both for the team and for developers reviewing or extending the code) if the same standard was followed throughout the source code.

- **Naming**

Considering Pnr.java - why not PersonalNumber.java and class PersonalNumber.java? This way you could have the

```
private Database memberDB; //berthRegistrations
```

- We are not sure if we understand this naming and comment correctly - does memberDB contain boats, members, or berths? Does the memberDB only contain berth registrations?

```
this.maxlength = (maxlength > 0 ? maxlength : 70); //standard 70
```

- From knowing the Domain we can infer that this is relating to max length of the boat, but perhaps boatMaxLength would be a better choice as that wouldn't leave the interpretation up to chance. Or clarify it in comments // where does 70 come from? 70 meters sounds long? Was that in the requirements? [2, p49]

- **Duplication**

No code duplication as far as we can tell.

- **Dead Code**

The boolean parameter mem in the genereteID() [sic] method in the YachtClub class appears to never be used.

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?**

see below

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

Pretty much yes. There are some boat data being updated in the Member class so it's not 100% but quite good. [1, p324]

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

Classes do have low coupling but the design won't allow for the view to freely present the data in an independent way due to the strings being served.

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

Statics are avoided.

- **Avoid hidden dependencies.**

No hidden dependencies that we are aware of.

- **Information should be encapsulated.**

Yes. Members are encapsulated and altered using getter and setters

- **Inspired from the Domain Model.**

(should the question be “Inspired from the class diagram.”? That is what we answer)
Yes, it follows the class diagram exactly.

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

We believe that the strings returned from the model would be better handed over to the controller/view in the form of objects. It will be very hard to implement a different user interface when the design looks like it does at the moment.

- **Comments**

Misleading comments for example:

- The methods following “// printing out info” in Member.java does not print anything but returns strings.
- “// secerity can be aplied here” After the setEmail method in the same file. This comment is placed outside of the method.

Comments are mixing English and Swedish, which is fine for the current team and most students at LNU. Good comment practice could be improved here by following the principle of writing comments for “the future programmer”, which could be yourself (after your forgotten your code) or somebody else - somebody who might not know Swedish.

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

The sequence diagram, whilst intimidating at first glance (given its height) is actually very clear - it illustrates where loops are used and which sets of alternatives the user will face. Sure, the level of abstraction “could” be higher, but given the limited scope of this project thats not necessary. Its actually quite helpful to get an overview of the flow of the basic requirements.

The class diagram is nice and simple - it illustrates the most important relationships and an a high level overview of the MVC implementation and also makes it clear that a helper/observer pattern is used. The implementation then follows the class diagrams naming so this is a great snapshot of the program.

What are the strong points of the design/implementation, what do you think is really good and why?
Strong encapsulation. Some explanatory comments. Care was obviously taken to add some nice touches to the user interface, such as the gorgeous ASCII splash screen. The way that id numbers is also good because it ensures that a unique number is generated each time and that they are always the same length.

What are the weaknesses of the design/implementation, what do you think should be changed and why?
Since the team is clearly aiming to expand functionality, it may help moving forward to separate Validation into a separate class. Also, rather than doing Validation directly in the Controller, moving this into the Model would help as the Model is concerned with Data & Rules [3]

Do you think the design/implementation has passed the grade 2 criteria?
Yes. Despite some issues we believe that the design should pass for a grade 2.

Sources.list

[1] Larman, Craig. (2004). Applying UML and Patterns (electronic version). 63rd ed.
[2] Martin, Robert C (2008). Clean Code
[3] 1dv607 Lecture 4 - MVC, Slide “MVC Solution”