# Reviewed by: Sing Trinh dt222cc@student.lnu.se

For group:

- rf222cz Rebecca Fransson
- hj222hi Hampus Jarleborn
- rs222kn Richard Söderman
- https://github.com/RebeccaFransson/1dv607/tree/master/Lab2

**Intro**

Some feedback can be wrong as I do not exactly know if I'm correct or not. Lack of references (no access to the course book atm). The application did not crash but did have some bugs regarding error handling and displaying that information to the user.

**Is the Architecture ok?**

The Architecture needs an improvement. There is MVC separation but there is some violation like Secretary having model responsiblities like adding a new member to the list, which should be handled in a model class. I believe the controller was not supposed to alter the domain but use the models to alter the domains.

Otherwise the architecture is good.

Secretary class are too "large" and have too many responsibilities (grade 2 requirement 12).

The unique member id are being randomised with no need to input a memberId during registration. (I wanted to have some kind of auto memberId but did not implement it). The Id length/size might be a bit overkill. Perhaps use memberId as the input instead of fixed row number. The member on "row" 7 might not be the same member another day.

**What is the quality of the implementation/source code?**

All the CRUD functionalities are included. Comments are somewhat helpful Methods are named well, easy to understand what the methods do. The naming of parameters/arguments are fine: camelCasing.

I see some dead code:

- ConsoleView line 168: ExitProgramPress()
- ConsoleView line 173: memberMenuKeyPress()

Naming conventions: (https://msdn.microsoft.com/en-us/library/ms229002(v=vs.110).aspx)

- Naming of methods: usage of camelCasing, should be PascalCasing.
- Naming of public and private members/fields: camelCasing w/o underscore and PascalCasing (followed but not consistent)

Error handling regarding input. The user should not be able to see some kind of information and should be handled in some way (like display just the message and no code info) Try e.Message if it works for your application as optional to e.ToString()

Plenty of various typographical errors (a.k.a typos) :D

The model throws exceptions(so does mine) and are not specialized to return strings.

There is some repeated code but nothing major. Tip: You can refactor the way your group handles colors and headers.

```
public void displayStartMenu()
{
    DisplayHeader("WELCOME", ConsoleColor.Cyan); // less repeated code
    Console.WriteLine("1. New member");
    Console.WriteLine("2. Detail list members");
    Console.WriteLine("3. Compact list members");
}

private void DisplayHeader(string header, ConsoleColor color) //in ConsoleView
{
    Console.BackgroundColor = color;
    Console.WriteLine("============{0}==============", header);
    Console.BackgroundColor = ConsoleColor.Black // or ResetColor();
}
```

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

Objects are connected using associations instead of using keys/ids, good.

Do more Object Oriented Programming. Instead of sending the different data: example ConsoleView line 71:

```
public void displayCompactList(int i, string name, int memberId, int nrOfBoats)
{
    Console.WriteLine("{0}.Name:{1}   Member id:{2}   Nr of boats:{3}",
    i, name, memberId, nrOfBoats);
}
```

after:

```
public void displayCompactList(int i, model.Member m) // or "member" instead of "m"
{
    Console.WriteLine("{0}.Name:{1}   Member id:{2}   Nr of boats:{3}",
    i, m.Name, m.MemberId, m.Boats.Count); //the Member.cs
}
```

There is no hidden dependencies and no use of static variables or operations (I think) as well as no global variables.

Information should be encapsulated (OOP-Encapsulation). Lots of methods in Secretary being public instead of private. Which means we can do stuff from outside the intended place. Think public when being accessed from outside the class and private when being only used withing the class.

Primitive data types that should really be classes (painted types) like List<model.Member> members to a MemberList.cs. With methods like AddMemberToList() etc (can be wrong about this ofc).

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

The class diagram would not help so much as it was not so well written. When looking at the diagram as the starting point, I get tiny bit confused on what is what without looking at the source code.

List<model.Member> should be moved to/inside c.Secretary as a field and List<model.Boat> should be moved to/inside m.Member as a field (as indicated in the implementation).

The class diagram does not show the same thing as the implementation. The diagram contains the parts needed but presented bad/wrong (note: I do not know how much the diagram was supposed to cover, MVC or not).

The association between m.Member and m.Boat are missing because the m.Member has a list of boats (List<m.Boat>). The association between c.Secretary and m.Boat is not neccessary as the c.Secretary use m.Member to add boats.

---

The sequence diagrams would be helpful. It gives a clear sequence on how this specific method would run. I do get what the method needs to have and on what order I should do stuff. The sequence diagram shows the same thing as implementation and the functionalities are well presented. Alternative paths are mentioned, which is good.

The diagram could be less cluttered. <<return>> is optional and in this case not so useful. Consider not having them for methods with no return values. Do something with clear() as that is not important enough to be in the diagram, have it moved/included in the different display methods instead to make the diagram and code more "clear".

Perhaps "string name = GetUserInput()" instead of just "getInput" (optional? for me it becomes more "helpful and clearer" which is kinda the purpose of the diagrams, this means some changes to the implementation).

Improvement (optional): Perhaps "foreach member in members" instead of "foreach (members)".

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

As mentioned from above. Comments and naming are helpful when looking through on what the methods do. Not so hard understanding the code.

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

Well, the mentioned negative notations from above. Like architecture, design, consistency and etc.

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

I believe it is almost sufficient, there's areas that needs some improvements, regarding code qualities, Object Oriented Programming and the diagrams. With a bit more work with tweaking and refactoring stuff, this group should be able to pass this workshop.

And once again excuse me for not being able to provide more references. Most of this feedback can be considered as opinions rather than facts :D so disregard some notation if you think that I'm at fault/wrong. Just adding my point of view for your work :D