Andreas Bom, ab22cw

Peer-review

Workshop 2,

Peer-review of Sing Trinh, dt222cc

Test the runnable version of the application in a realistic way. Note any problems/bugs.
I tried my best to crash the program, without success. All input errors are handled with exceptions. I deleted the data file that holds the saved information and the result were that an empty list was showed in the application. In other words, I could not make the program crash.

Try to compile/use the source code using the instructions provided. Can you get it up and running? Is anything problematic? Are there steps missing or assumptions made?
The repository includes an instruction file that states where the files are located, and an instruction on how to get the application up and running. Instructions are very clear and easy to understand.

Does the implementation and diagrams conform (do they show the same thing)? Are there any missing relations? Relations in the wrong direction? Wrong relations? Correct UML notation?
(You have multiple diagrams. I have mainly been studying the one under heading 'MVC, Structure of the Application')
The class MemberList is within the model namespace, however the file is located under the folder 'view'. Move this file to the 'model' folder.
You are showing the direction of associations between RegistryControler and 'Every Views', but all other associations show bidirectional associations. Do you have an intention with this? In reality, classes in the model does not know about classes in the view (for instance, MemberListView and MemberList could have an arrow instead of a line).
In the diagram, the enum <<BoatType>> shows an association to boat. But in your code, the enum is nested within the class and is part of the class (The enum is actually not BoatType, but Boat.BoatType.). Enums does not need to be declared inside a class, but could be if desired. This is a design decision you have to make.

Is the Architecture ok?

- ***Is there a model view separation?***
- You have a model-view separation to most of the code. The only exception to this is messages from exception that are thrown from the model, are directly outputted to the view (via the controller). It could be argued if these messages should be a view concern instead of model. For instance, should the same message always be shown the same on a mobile device as on a computer or website? Sometimes this is the case, sometimes not. In your application, there is probably ok to have these messages coming from the model, but you should have this in mind in the future. There are not always good to give the error message in the model.
- ***Is the model coupled to the user interface?***
- No, the model is not coupled to the user interface.

- ***Is the model specialized for a certain kind of IU (for example returning formatted strings to be printed)***
  The string is not formatted in any way. But the message is sent from the model to the view. See answer above.

- ***Are there domain rules in the UI?***
  No, there are no domain rules in the UI.

Is the requirement of a unique member id correctly done?
The unique member id must be set manually by the user, and the uniqueness of the number is checked when the new member is added. This is not a good way to do it. Let say there is 10 000 members in a database. How should the user know which id is free to pick? A better way to do it, is to ignore the user to choose id, and instead let the class MemberList pick a number that is unique and assign this id to the member in the method AddMember().

What is the quality of the implementation/source code?

- ***Naming***
- You have good names on your classes, methods and variables (AddBoat, DeleteBoat, ChangeName etc). You are following the recommended conventions (Capitalizing class names. Private members start with en underscore (_name). You don't have implicit naming on variables (int x = 1. What does x stand for? == implicit naming).
- ***Duplication***
  You do have some code duplication. This is something you are aware of, since there are some comment on this in your code.
- **Dead Code**
  I have not found any dead code in your project.

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

- ***Objects are connected using associations and not with keys/ids.***
  Hamilton & Miles[1] states that an association is a dependency between classes, where the class will contain a reference to an object (or objects) as an attribute. You have been using associations to connect objects.

- ***Is GRASP used correctly?***
  GRASP principle is an aid for using the design principles 'pattern of assigning responsibility'. Responsibilities can be of two types: *Knowing or Doing.* This means that the object can either do something itself, initiate some other object to do something or controlling actions in other objects. Knowing responsibilities means that the object knows about private encapsulated data, knowing about related objects or knowing about things it can obtain or compute[2].

1 Hamilton, K., Miles, R C (2006). Learning UML 2.0. Sebastopol: O'Reilly Media. Chapter 5.1.2 Class Relationships (Heading: *Association*)

2 Larman, C (2005). *Applying UML and Patterns (Third Edition)*. Upper Saddle River: Prentice-Hall. Chapter 17 GRASP: Designing Objects with Responsibility. (Heading: *Guideline: What are suitable attribute types?*)

I will use examples to illustrate GRASP implementation in your application:

CREATOR:
The class RegisterController creates an object from Member. This follows the creator pattern (RegisterController has initialization data for Member).

INFORMATION EXPERT:
Here I would say you violating the principle when it comes to assigning a unique id. You give the responsibility to Member class, and then throws an exception if the id is not unique. You should give the responsibility to assign an id to MemberList and give this id to Member, since MemberList knows all unique id's that is given to existing members.

LOW COUPLING:
You have high coupling between MemberList and MemberListDAL. Let say you want to change how members are saved. Maybe you want to save members in a database instead of a text file. One way to reduce coupling would be if MemberList took an argument in the constructor with an DALinterface.

```
MemberList constructor:
public MemberList (IMemberListDAL memberListDAL)
{
        _mDal = memberListDAL
}


memberList = new MemberList(new mySQL_db_DAL);
or
memberList = new MemberList(new txtfile_DAL);
```

By doing so, you can send in any DAL as long as it implements the IMemberListDAL interface.) This means that you could have multiple ways to store data, and all you have to change is which DAL you send in when creating MemberList.

CONTROLLER:
You are using the controller pattern.

HIGH COHESION:
In my opinion, your classes have high cohesion. One reason to this is that you use inheritance for different views.

You do not have global or static variables. I have not found any hidden dependencies.

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

I think the diagrams would help me. The diagram ‘MVC Structure of the application’ is quite clear and, to me, this together with the other two (MVC, (Same diagram)) I would understand how objects relate to each other.

What are the strong points of the design/implementation, what do you think is really good and why?
The strong points are good exception handling. I could not make the program crash. Also, I think the class diagram with association is easy to read and understand.

What are the weaknesses of the design/implementation, what do you think should be changed and why?
I think you should change how the MemberList is coupled to the MemberListDAL class. The reason to this is that it is difficult to change how data is saved. MemberList does not care how data is saved, it just care that data is delivered.

Do you think the design/implementation has passed the grade 2 criteria?
Yes, I think it has passed grade 2.