Reviewers: Pepyn Swagemakers ps222nq & Sebastian Gustavsson sg222pm

Review of: lo222hd / https://github.com/LindaOtt/1dv607-workshop-2

1: Running the Application
While no instructions for running the program are given, some experience working with C# and Visual Studio are sufficient to get the program to run. It compiles fine and works as expected without encountering any bugs, however as stated in the Readme the application is incomplete and does not meet the described requirements.

2: UML and diagrams
The application and diagrams do not conform, instead the class diagram is more similar to a Domain Model and the sequence diagrams only contain UI and PersistentStorage classes; these classes are not present in the source code.

The diagrams would not be very helpful to a developer at this point, since the naming and classes is only partially implemented in code. The programmer seems to mix Domain modelling with the Class diagram modelling, some of the classes implemented are missing in the Class Diagram.

3: Architecture
While incomplete for now, the architecture is correct with a separation of model, view and controller. The model is not fully implemented but there appears to be no coupling to the user interface. Instead, the model is adaptable to handle different interfaces.The view/UI is not fully implemented but so far does not contain domain rules.

4: User ID
The Member class contains an integer for unique user ID but there is no validation in either the model or the controller to verify correct format or prevent duplications.

5: Code quality of the implementation/source code
Overall the code quality is good with clear variable and method naming. The code is easy to read and understand. There is some duplication in the controller; arguably the two switch statements to handle user input could have been combined into one. Commenting adds value overall but there are some cases of redundant comments, i.e. a comment that states “Write the string to a file” in a function named writeToFile that takes an argument of type String. As far as we could see there is no dead code in the application.

6: Design quality of the implementation
Objects in the software are connected using associations, not by using keys or id. The controller class could be considered too large and we suggest it is split into two different controllers to better adhere to the Low Cohesion principle. In this case both file reading and writing and user input/output handling are in the same controller which makes it harder to adapt the design at a later stage. This is supported by Larman(2005), who states: *“A common defect in the design of controllers results from over-assignment of responsibility. A controller then suffers from low cohesion, violating the High Cohesion Principle”.* To fix this,

Larman gives the following guideline: *"A controller should delegate to other objects the work that needs to be done; it coordinates or controls the activity. It does not do much work itself*".

More guidance from Larman (2005) can be found when studying the signs of bloated controllers he mentions:

- one single controller receives all events
- the controller performs many of the tasks necessary to fulfill system events

The latter item in this list indicates a violation of the Information Expert and High Cohesion principles.

In accordance with the author we propose the following solutions:

- use more than one controller
- delegate functionality from controller to other parts of the application
- remove methods that should be in the Model part of the application from the controller

A similar situation can be found in the view where one single view file is used to create both input and output views, we recommend splitting this up to better achieve separation of concerns as well as increase later adaptability of the application in case one would prefer to change the output view but maintain the same way of input or vice versa.

Having the enum with view types in the controller violates the low coupling principle, we recommend moving this enum into its own file in the view namespace for easier use and reference. This would also resolve the issue of hidden dependencies, since in it current state this enum creates a dependency between the controller and view classes.

The models adhere to GRASP principles with a proper separation of concerns, however need to be expanded to meet all requirements and handle the persistent storage element that is described in the assignment and diagrams.

Use of static and global variables is avoided in the application and the information presented in the models is properly encapsulated using get and set methods.

The code and its elements clearly reflect the domain model: the different options specified in the view correspond to the required use cases and the different models are similar to the elements retrieved from the domain model.

7: Strong points of the design and implementation

So far the models are implemented correctly with information encapsulated through the use of getters and setters. The user interface of the console application is working well and easy to use and understand. The code quality is good as well with clear variable and function names. Well done for a first iteration as validation of concept and idea, code can easily be refactored and refined; even though structure is not optimal right now the programmer clearly understands the domain model and the requirements to be met.

8: Weaknesses of the design/implementation

As of now the implementation is too bloated, too much responsibility is assigned to single classes which should be separated out.

9: Do you think the design/implementation has passed the grade 2 criteria?
Not currently since the requirements are not fully implemented and the design is missing GRASP principles in the existing code, but the programmer is on the right path.

References

Books

Larman, Craig. *Applying UML And Patterns*. Upper Saddle River, N.J.: Prentice Hall PTR, 2005. Print.

Martin, Robert C. *Clean Code*. Upper Saddle River, NJ: Prentice Hall, 2009. Print.