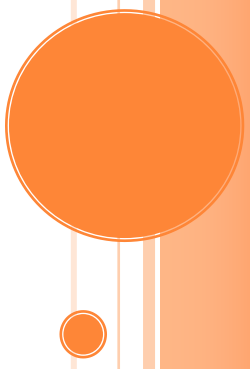
Object Oriented Analysis and Design (1DV600)
Workshop 3, Peer Review

Reviewer: **Janty Azmat**
E-Mail: **ja223gs@student.lnu.se**

A Peer Review For: **Jovydas Urbanavicius**
Mantas Remeika
Jean-Pierre Salum
E-Mail: **ju222bk@student.lnu.se**



In this review I will try to follow the points in the peer review instruction because they seem to be organized in a clear and comprehensible way:

- **Looking At Diagrams And Implementation, Trying And Testing Implementation**

After the first look at the diagrams and checking the implementation, it seems by testing a runnable version of the implementation that the application works nicely, which indicates that the first three points of the grade 2 requirements (including implementing the 'Game::Stand' operation) were done. Although there is an extra step (the 'getVisibleCard' method/function) that is not in the sequence diagram, which seems to be added due to other requirement of the workshop.

- **Does The Implementation And Diagrams Conform?**

The class diagram seems similar to the original one, but there seems to be some parts of the implementation were forgotten and not updated in the class diagrams, e.g.:

- In the 'controller.PlayGame' class; the use of 'model.Game' and 'view.IView' was changed from parameters in the 'Play' method/function to fields/attributes in the implementation (better use 'private' modifier here), but was not updated as an 'association' in the class diagram.
- There is an 'association' between 'model.Player' and 'model.Observer' in the implementation that is not translated into the class diagram.

- **Is The Dependency Between Controller And View Handled?**

The hidden dependency that I consider in this situation is the fact that the checking of input (checking if the input is equal to 'p', 'h', 's', and 'q' in our case) is done in the controller, while it should be done in the view. Here, as it seems, the checking of the input is still in the controller, which entails that the hidden dependency is still not handled (of course this does not mean that there is no other hidden dependency that is already solved).

- **Strategy Pattern For The Rule Variant Soft17**

About solving the soft 17 rule, I have two points:

- The soft 17 case is implemented in the 'model.rules.Soft17Strategy' class but it is actually not used. It supposed to replace the 'model.rules.BasicHitStrategy' in the 'GetHitRule' method/function in the 'model.rules.RulesFactory' class, but instead it was included in the 'GetSoft17Strategy' method/function in the aforementioned class which was never used anywhere in the implementation.
- This one has nothing to do with the design, but with handling a special case (maybe we can call it a bug). In the 'model.rules.Soft17Strategy' class again; the 'DoHit' method/function does not handle the hard 17 with an Ace situation (e.g. a hand with Ace, Ten, and Six -which is not a soft 17 case- is considered a soft 17 case in the implementation).

- **Strategy Pattern For The Variations Of Who Wins The Game**

The strategy pattern for winning is represented in the ‘model.rules.WinRule’ interface, and there is a concrete implementation/realization for it in the case of the user winning on a tie in the class ‘model.rules.PlayerWinsOnEqual’. But as with the soft17 rule, there is a ‘getWinRule’ method/function in the ‘model.rules.RulesFactory’ class that returns an object of the ‘model.rules.PlayerWinsOnEqual’ class (encapsulated in the interface ‘model.rules.WinRule’), but this method/function was never used anywhere in the implementation. Additionally, an unneeded and unused ‘winRule’ field/attribute is put in the ‘model.Game’ class (it is also not translated into the class diagram).

- **Removing Requested Code Duplication Without Adding Any Dependencies**

There was a try on getting rid of the duplication by splitting the steps between the ‘getVisibleCard’ and ‘Stand’ methods/functions in the ‘model.Dealer’ class, but that only got rid of one of the duplications, and there are still few others (one of them is still in the same ‘model.Dealer’ class and few in ‘model.rules.AmericanNewGameStrategy’ and ‘model.rules.InternationalNewGameStrategy’ classes for example). If I may give an advice in that matter, I would suggest using a method (that has the duplicated code) in the class that already has direct access to both the ‘model.Deck’ and ‘model.Card’ classes and replace all the duplicated code with that method (hint: this will lead to the case where the ‘model.rules.INewGameStrategy’ interface and the classes that implement/realize it no longer depend on the ‘model.Deck’ class -which means less dependency-).

- **The Observer Pattern Implementation**

In the case of the observer pattern, I have the following points (starting with the point that I think is good in this case):

- The inclusion of the ‘model.Subject’ interface gives a good idea of the structure of an observer. Although, “the subject class is not required to implement a given interface [1]”, but it seems to be a good practice.
- While the observer pattern is implemented correctly, the part of the requirement that requested the pause between every card hit also states: “the pausing code should be in the user interface (view or controller) and not in the model”, and the current implementation have the pausing code in the model package (in the ‘DealCard’ method/function of the ‘model.Player’ class).

- **The Grade 2 Criteria**

Hoping that those suggestions were not long, I think that they may help fine-tuning this submission to achieve the grade 2 criteria.

References

1- Purdy D., Richter J., "Exploring the Observer Design Pattern." MSDN Library, January 2002. Available at:
https://msdn.microsoft.com/en-us/library/ee817669(pandp.10).aspx.