Peer review Workshop 3

For: Alexandre Driaguine (ad222kr), Wictor Kihlbaum (wk222as) and Niklas Johannesson (nj222dt)
By: Ola Franzén (of222au)
In the course Objecktorienterad analys och design med UML

Compiling and starting up the source code was no problem, and the presentation of the cards with the pauses was done well in the output (console). I do notice an apparent issue though, in that the application stops (quits) when the game is over. It's even hard to tell the outcome of the game (if you won or not) before it has quit. Commenting out the line 28 in PlayGame class solved the quitting issue but as you've commented the "game over" message displays 3 times. Maybe this could be solved in another way like refreshing the view before displaying the game over message? Other than that the game functions well and the pausing is made well when it comes to presentation (output).

The earlier hidden dependency between the controller and view (with the GetInput method) has been solved well, with an action in the form of an enum as return value instead. This removes the dependency and makes it easier to change either class without affecting the other in this part.

The soft 17 hit rule has been solved well by the strategy pattern. The win rule variations have also been solved in accordance to the strategy pattern, as described by Larman [1, p447]. They also don't use any extra dependencies since the scores are sent which is good in accordance to low coupling principle, described by Larman [1, p301].

When it comes to the solution of the duplicate code (the dealing of cards) I can see a few issues though. The first thing is the place where the code for retrieving a card from the deck and deal to a player is put in the new game rules classes. This could be fine if it were only in the new game method this happened but since the same code also is used when hitting a new card later on in the game flow this is not so good in accordance to for example the Information Expert pattern, as described by Larman [1, p296] that you should "assign responsibility […] to the class that has the information necessary to fulfill the responsibility". In this case it would probably be better to let the Dealer handle this, since it has the deck with the cards. It also adds a dependency to Deck in all the new game rule variations. Another issue I notice in the DealCard method in Player is that it performs the pausing of the application. This is probably not the best model to have this responsibility, since you might not always want to have this pausing when using the same Player class in another project. It should probably be considered a controller or view responsibility and therefore performed in one of those.

When it comes to the observer pattern the general structure of those in the Player class is done well. However you have opened up Dealer and Player as public objects in the Game class to make the controller subscribe directly to the Player objects events. It solves the problem but it dependency from the controller directly to the Dealer and Player objects. A solution could be to use the same observer pattern and methods in Game class as well, then the controller could observe the Game class only and no dependencies would be added.

Looking through the updated class diagram most parts of it looks good. It seems to be missing the association from Player to the IBlackJackObserver though. Also the PlayGame controller doesn't have the association to Game, but maybe was left out since only changes were needed to be shown.

All in all I think all parts have been solved and the patterns like the observer pattern have been used. However I think some changes to reduce the dependencies and put some actions in the correct class in accordance to Information Expert pattern etc could be needed to pass the grade 2 criteria.

References

1. Larman C., Applying UML and Patterns 3rd Ed, 2005, ISBN: 0131489062