Peer review of Seldin Music's group Workshop 3 1DV607

Try to compile/use the source code provided. Can you get it up and running? Is anything problematic?

We could compile without any problems, and after some testruns it works without any problems.

Test the runnable version of the application in a realistic way. Note any problems/bugs.

We have not found any bugs, everything works fine.

Does the implementation and diagrams conform (do they show the same thing)? Are there any missing relations? Relations in the wrong direction?

Yes, they do show the same thing and is very easy to follow. Also the relations are correct and nothing to complain on.

Is the dependency between controller and view handled? How? Good? Bad?

This is handled good via a switch-statement. You removed the hardcoded characters and replaced it with 1, 2, 3 etc which is more versatile. If you want to change keys you do this in the responsible class instead, which now is the SimpleView and SwedishView etc.

Is the Strategy Pattern used correctly for the rule variant Soft17?

The strategy pattern is correctly used for the Soft17Strategy. We can easily change the gamestrategy in the rulesfactory, and the factors/incoming data are not known for each case until we compile, and is handled thereafter.
Each algorithm is defined in a new class with a common interface[1, p 447].
Unfortunately you do not handle the "6", you do check if there is an Ace but not if there is 6 combined with an Ace, and that was one of the rules with the Soft17.

Is the Strategy Pattern used correctly for the variations of who wins the game?

It is the same way here as above, nice and clean solution which definitely uses the strategy pattern[1, p 447].

Is the duplicate code removed from everywhere and put in a place that does not add any dependencies (What class already knows about cards and the deck)? Are interfaces updated to reflect the change?

Because of the previous state with duplicate code in two separate classes, the creation of an abstract class is an excellent way of solving this. The abstract class handles the dealing of a card, and then the two classes extends that abstract class and uses that method instead. This removes the duplicate code. This does not add any unnecessary dependencies. Since you created an abstract class for the cardhandling and extended that into the American and international strategies no interfaces needs to be updated to reflect the change.

Is the Observer Pattern correctly implemented?

Yes it is, you got both subscribers and publisher and a notifiermethod. According to Larman's book "Applying UML and Patterns" the publisher can dynamically register subscribers who are interested in an event[1, p 465], which is also the case here.

Is the class diagram updated to reflect the changes?

The class diagram definitely reflect the changes made to the original implementation/class diagram.

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

Both the class diagram and implementation is well performed and executed. We think that you should only fix the Soft17-rule to handle 6 to pass the grade 2 criteria.

References

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