**Instruction**

Look at the models, implementation and any accompanying documentation. Try to have an open mind and focus on trying to understand the materials as it is presented.

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

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

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

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

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

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

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?

Is the Observer Pattern correctly implemented?

Is the class diagram updated to reflect the changes?

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

**Problems/bugs**

When reviewing the code, I have concluded that the code compiles, but does seem to have some bug(s) in it, as the first output is, making unclear for the user which player in the game is which:

*“Name has:*

*0*

*Name has*

*0”*

In able to determine what happens, I debugged the code, and found that in the method *DisplayWelcomeMessage* in the view, the observers are notified, and in order to get the name of the player, the method *GetName* in the class *Player*, which returns the string “name”. This implementation creates a string dependency in the model class *Player*.

When the user chooses to hit or stand, the hand of the second player (presumably Player, not Dealer) is shown, and blinks three times, on account of *DisplayWelcomeMessage* several times, both from the controller class *PlayGame* as well as from the view. The game pauses before showing the hand of the player, however the instructions were to that “The pause should be when any player (dealer or player) gets a card.”[[1]](#footnote-1)

**Dependencies**

The dependency between the controller and the view has been eliminated, by using an enumerable, which is a good solution. Since there is a dependency between the model class *Game* and the view, I would argue that the observer pattern is not correctly implemented.

**Rule Variations**

The strategy patter has not implemented correctly for the soft 17 variation, as the *Player* class has been changed. The method *HasAce* in player is called upon dealer, which derives from the *Player* class, so technically the *Dealer* has not been changed. The method *HasAce* is only called in the *DoHit* method of the *Soft17HitStrategy* class. However, since a reference to an instance of *Dealer* is passed as an argument to the method, the code in the *HasAce* method can be moved to the class specific to the Soft 17 variation, and not affect the underlying code at all, and therefore be an better implementation of the strategy pattern. Furthermore, the game variation which determines who wins seems to be implemented correctly.

**Refactoring/ Duplicate code**

The application still contains duplicate code in the classes which handle initiating a new game, in the namespace rules.

**Class Diagram**

The class diagram looks to be correct in terms of UML notation, it is clear what classes have an association to which, and looks to reflect the changes in the application. The arrows seem to be in the right direction.

**Pass/Fail**

I would conclude that several things need to be corrected in the implementation; this version would not pass the grade 2 criteria. The dependency between the model and view which is added when adding subscribers in the model class must be corrected, as well as refactoring the code so that the soft 17 strategy does not in any way change the underlying code. Also, the pause should be for each card. See the section Problems/bugs.

.

1. https://coursepress.lnu.se/kurs/objektorienterad-analys-och-design-med-uml/workshops-2/workshop-3-design-using-patterns/ [↑](#footnote-ref-1)