Peer review to Tobias Nilsson from Tommy Kronstål, tk222hq@student.lnu.se

Compiling and running
All I had to do to get the application running was importing it to Eclipse and it hit run. No errors/ warnings or compilation problems. Pressing p in the application provided the excitement by presenting a card one at a time and I was also able to hit, stand and quit.

Diagram
The first thing that strikes me is the presence of two DealerWinsWhenEqualStrategy classifiers in rules package. My guess it that one of them represents PlayerWinsWhenEqualStrategy.

Also, American- and International NewGameStrategy is portrayed with realisation arrows towards AbstractGetAndDealCard, which seems to be incorrect since they do not implement an interface of type AbstractGetAndDealCard. In the course literature a good diagram summarise the use of sub class implementation [1, p250].

I shows the added interfaces ICardDealtObserver (good name by the way, exactly the same I decided on) and IWinDeciderStrategy along with the new classes SoftSeventeenHitStrategy, Dealer(Player)WinsWhenEqualStrategy and also the abstract GetAndDealCard and all associations, realisations and dependencies.

Dependency between controller and view
As far as I can see there is still a dependency between the controller and the view regarding the user input. The controller now listens for p, s, h, q which represents the commands for the english version. If we switch to another language it would be a better idea to have the controller determine its evaluation based on booleans and let the view determine which action the user has chosen. I hope you find this suggestion useful.

Strategy pattern
The soft17 is perfectly implemented as a strategy where the rules for how to handle an ace is implemented in its own hit strategy called from the rules factory and realising the IHitStrategy interface.

Who wins the game seems to work out fine, however I would believe there is a way of implementing this without having to use two different strategies. A benefit of using just one would be the simplicity of refactoring and to keep the design as minimalistic as possible. In short, either the dealer wins on equal or not. The same strategy does not have to be implemented for the player.

Duplicated code
The use of the abstract class GetAndDealCard is one solution of the duplicated code problem. One downside of this is that it uses static methods. Examine if there is any class that already knows about deck and players/dealers that could implement this.

Observer pattern
The implementation of the observer pattern seems well designed by letting the controller call a method in the game, which already knows about player and dealer. Finally, letting the observer use methods in the view to display the changes makes the layers working in perfect harmony.

Summary
This workshop is well done and reaches a passing grade, but I would suggest a quick look at making the view and controller less dependent on the users input whatever language used.

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