Task

You will continue the implementation and documentation of a Black Jack game. The design is similar to what we have done on the theory but it is not exactly the same. The focus is not on usability or a nice user interface but to have a robust and well-documented design that can handle change. In the provided code, you find a started but not a playable version of the game in Java. As there are many different variants of the rules one of the aims is to create a flexible design that supports different combinations of rules. There is also a class diagram that describes the packages, classes and the main relations in the implementation (note that there are dependencies in the implementation that are not shown in the diagram for readability purposes, e.g. there are many dependencies to the Card class). There is also a sequence diagram for one, not implemented, part of the game.

Requirements

1. Study the class diagram and the source code to understand the design of the game.
2. Implement the operation Game::Stand using the sequence diagram Game_Stand. The game should now be playable.
3. Remove the bad, hidden, dependency between the controller and view (new game, hit, stand)
4. Design and implement a new rule variant for when the dealer should take one more card. The new variant is *Soft 17*, use the same design pattern already present for *Hit*. Soft 17 means that the dealer has 17 but in a combination of Ace and 6 (for example Ace, two, two, two). This means that the Dealer can get another card valued at 10 but still have 17 as the value of the ace is reduced to 1. Using the soft 17 rule the dealer should take another card (compared to the original rule when the dealer only takes cards on a score of 16 or lower).
   Hint: this is a typical behaviour you would like to have automatic test cases for.
5. Design and implement a variable rule for who wins the game. This variation could, for example, change who wins on an equal score (in one implementation the Dealer wins, in the other the Player). The design should make it easy to add other variants without changing the Dealer. Use the same design pattern as used in the Soft 17 design.
6. The code for getting a card from the deck, show the card and give it to a player is duplicated in a number of places. Make a refactoring to remove this duplication and that supports low coupling/high cohesion (i.e. check how you can evaluate different solutions to the problem and select the one that gives the best result according to low coupling/high cohesion). The code that is duplicated is similar to this:

```
Card c = deck.GetCard();
c.Show(true/false)
player.DealCard(c);
```

8. Use the Observer-pattern to send an event to the user interface that a player (human or dealer) has got a new card in his hand. When the event is handled the user interface should be "redrawn" to show the new hand (with the new card) and the game should be briefly paused to make the game a bit more exciting. The pausing

code should be in the user interface (view or controller) and not in the model. You should design and implement the observer structure yourself and not use any library classes like java.util.observer etc. The pause should be when any player (dealer or human) gets a card.
For example, when starting the game the following “pattern” of pauses should present in the user interface:

```
Dealer:
Player: c1 (player gets the first card)
*pause*
Dealer: c1 (dealer gets the first card)
Player: c1
*pause*
Dealer: c1
Player: c1, c2 (player gets the second card)
*pause*
Dealer: c1, c2 (dealer gets the second card)
Player: c1, c2
```

In this example video, of a graphical user interface to the black jack game, these pauses are used to animate the cards and play a sound.