SYST 17796 Project:

Deliverable 2

For Faculty

Prepared by PO1_1205_11044-7

Sheridan College

Submitted:

20 July 2020

Table of Contents

Contents Page

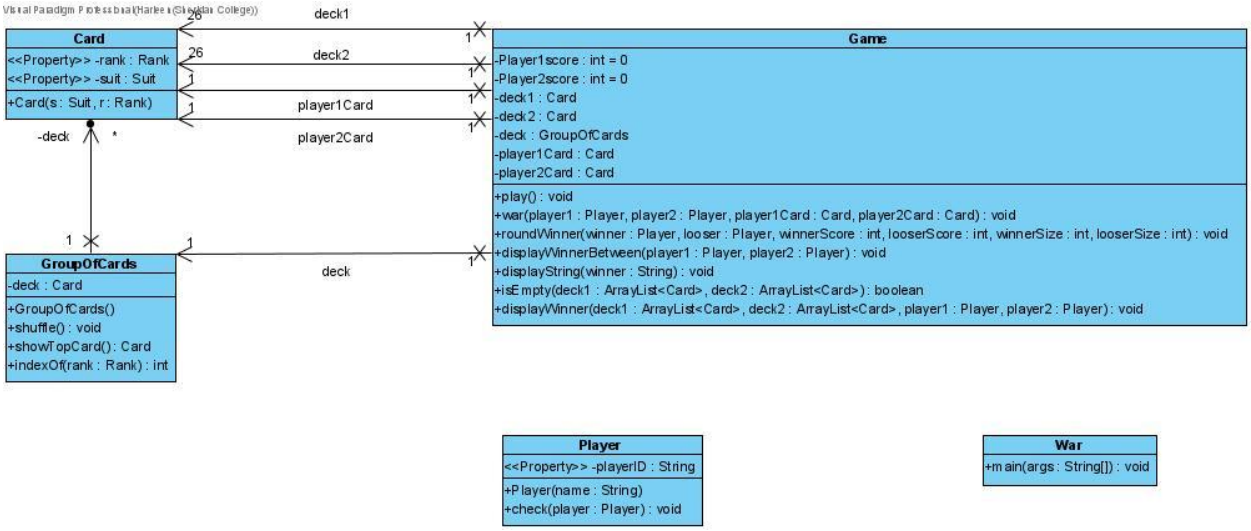
Use Cases

- **Main path-**
  - The game starts and players must enter their name as player ID.
  - The player 1 enters his or her player ID, “Tom” followed by player 2, “Sana”.
  - The first round of the game begins.
- **Alternate path-**
  - The game starts and players must enter their name as player ID.
  - The player 1 enters his or her player ID, “Tom” but player 2 enters without a player ID.
  - A message pops up on terminal saying that the player entered invalid player ID and the game needs to be restarted.

- **Main path-**
  - The game starts and players must enter their name as player ID.
  - The player 1 enters his or her player ID which is unique, “Tom” and player 2 also enters player ID which is unique, “Sana”.
  - The first round of the game begins.
- **Alternate path-**
  - The game starts and players must enter their name as player ID.
  - The player 1 enters his or her player ID which is “player” and player 2 also enters the same player ID “player”.
  - A message pops up on terminal saying that the players should enter different IDs from each other, and the game needs to be restarted.

- **Main path-**
  - The game starts and players must enter their name as player ID.
  - The player 1 enters his or her player ID, “Tom” followed by player 2, “Sana”.
  - The first round of the game begins.
  - Sana wins the first round and the score is displayed on the terminal.
  - The players are asked whether they want to play the next round to which they say yes.
  - The second round starts and continues the same way.
  - After few rounds Tom is left with 1 card only, so he loses the game and the scores as well as the name of the winner is displayed on the terminal.
- **Alternate path-**
  - The game starts and players must enter their name as player ID.
  - The player 1 enters his or her player ID, “Tom” followed by player 2, “Sana”.
  - The first round of the game begins.
  - Sana wins the first round and the score is displayed on the terminal.
  - The players are asked whether they want to play the next round to which they say no.
  - The scores as well as the name of the winner which has more scores is displayed on the terminal.

UML Class Diagram



Design Document Template

Project Background and Description

Our project is to design the card game 'War'. War is played between 2 people using a standard deck of playing cards. The first player to get all the 52 cards wins the game. "The deck is divided evenly, with each player receiving 26 cards, dealt one at a time, face down.

Each player turns up a card at the same time and the player with the higher card takes both cards and puts them, face down, on the bottom of his stack. If the cards are the same rank, it is War. Each player turns up one card face down and one card face up. The player with the higher cards takes both piles (six cards). If the turned-up cards are again the same rank, each player places another card face down and turns another card face up. The player with the higher card takes all 10 cards, and so on." (Bicycle Playing Cards, n.d.)

The Project Starter code written in Java has 4 classes – *Card*, *GroupOfCards*, *Game*, and *Player*. We have used one more class, '*War*' which is the view class for our program.

When *War* class is run, both players are asked to enter their names and each player is given a unique id. After each round, it displays the rank of the cards turned up by both players, who has won the round and the score of each player. We can also see how many cards each player has in total. The game is now designed for unlimited rounds until the players quit or one of them loses. Then we get to see a message showing the winner and his/her score.

Design Considerations

The game that we have chosen, *War*, will have 5 classes. *Card*, which is the basic card class will have 2 enumerations, *Rank* and *Suit* of the card, a parametrized constructor and getter and setter

methods for rank and suit. *GroupOfCards* will be the class where we will make deck of 52 cards. It will have methods for showing the top card of the deck, to shuffle the deck and to return the index value of any card. The *GroupOfCards* class will be associated to Card class as it is shown in UML diagram.

Player class is the class which models the 2 players of the game. Both the players should have unique player ID so this class will have a constructor, getter and setter methods and a method to check that both the players have a unique player ID. The *Game* class is the one with which the user interacts. The *play* method will be the method which prompts for player IDs and begins the game. The *war* method is for when the players get same rank cards. The *roundWinner*, *displayWinner* and *displayWinnerBetween* will be the methods that display the scores as well as winner player ID. *displayString* will be the method which is called by each of the 3 winner displaying methods. *isEmpty* will be the method to check when one of the two decks is left with 1 card only. The *Game* class is associated with Card class because of *deck1*, *deck2*, *plater1Card* and *player2Card* as well as it is associated to *GroupOfCards* class by *deck* as shown in UML class diagram.

Finally, the *War* class, which will have the *main* method, and which calls for *play* method in *Game* class.

Object Oriented Design Choices

- **Encapsulation:**

  Encapsulation is the process of binding data fields and methods in a single class so that it is protected from external interference. Encapsulation even helps in implementing loose coupling.

For example: All the data members in our classes are made private to protect them from external interference and the methods are made public. For instance, in *Card* class, we made the enumerations and data members private and the methods that need to use those data members have a direct access to them as they are included in the same class making it a capsule.

- **Delegation**

  After extending from the base code, we have used 5 classes to write the code for *War* game where each class performs a specific task. *Card* class is the base class for our project. *Game* class is the view class (MVC). It contains methods for playing the game and displaying the winner. *GroupOfCards* is a class used for making deck and includes methods that are related to the deck. *Player* class assigns an ID for each player and checks if it is unique. *War* class is the class which has the *main* method.

- **Cohesion**

  The code that we extended from the base code has high cohesion as all the methods in a class are related to each other and are completing one task together. As an example, *GroupOfCards* class contains the constructor which makes a deck of 52 cards. The methods in this class are the methods that act on that deck of cards like shuffling, showing top cards of the deck and retuning index of a card in the deck.

- **Coupling**

  This code will have low coupling. Any changes made to a method in one class will not make us change the methods in other classes. For example, if we make change in the *Player* class and add a regex for player ID, we will not have to make any change in the *play* method of the *Game* class which prompts for player ID.

- **Inheritance, aggregation, and composition**

  In the extended code, we do not use inheritance, aggregation, and composition because the code in not repeated in any of the classes and to avoid tight coupling. There is no child and parent class in our code.

- **Flexibility/Maintainability**

  The *War* game code that we made can be easily maintained. Each of the 4 members in our group have the access to git repository on GitHub. Anyone of us can pull or fetch and merge the code from the remote repository and make changes as they wish. The code can be easily modified as every class is doing one unique task. It is easy to identify the problem and correct it. Also, we did not use inheritance or abstraction, so it is easy to make change because we don't have to change it in other classes.

References

- Bicycle Playing Cards (n.d.) War – Card Game Rules. Retrieved from https://bicyclecards.com/how-to-play/war/