Sheridan

SYST17796 Fundamentals of Software Design & Development
Template for Project Deliverable #2

OVERVIEW

For the following sections already existing in the 1st deliverable, incorporate the following information:

1. PROJECT BACKGROUND AND DESCRIPTION

Elaborate on the game you chose and the Description provided in Deliverable 1 by providing more detail on the exact scope of your project (i.e. “the game will terminate after four rounds, giving each player a total score”).

The rules of the card game, war, was briefly described in deliverable one. In more depth, the game rounds are based on the number of cards each player has per round, the game is over when one player collects all the cards, and the other loses all the cards. In this case the player who have collected all the cards is the winner, the other player is loser, the game is over. This will result in a game over display that displays total score end of the game of the player who won the game, and the name of that player.

Following are the user stories with acceptance criteria as known as high level requirements, use case diagram that was made based on the high-level requirements and an updated class diagram:

FIGURE 1: HIGH LEVEL REQUIREMENTS

Business Requirements	User Stories
Ability for each player to register with the game	As a player, I want to register with the game, so that I can participate in the game. Acceptance Criteria: Player is successfully registered in the game before entering the game session and is issued a set of cards.
Ability for the game to communicate a win or loss	As a player, I want to know the outcome of the

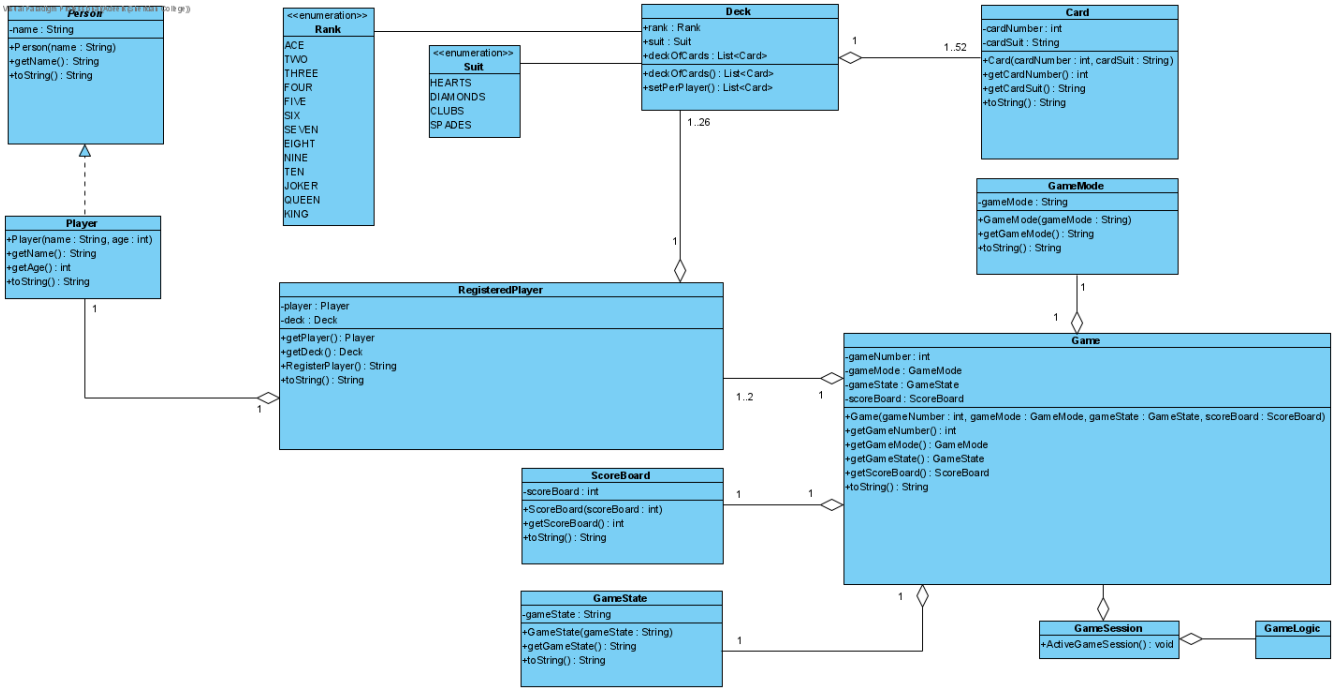
	game, so that I know whether I won or lost a game. Acceptance Criteria: When game over, total score and player winning condition must be displayed.
Ability for players to know their status (score) at all times	As a player, I want to know the score of the game at any time, so that I can know the status at all times. Acceptance Criteria: Display each player score.
Ability for drawing the card from the set	As a player, I want to draw the card each round of the game session, so that I can play my turn in the current round. Acceptance Criteria: Display one drawn card per player each round in normal game mode.
Ability for drawing four cards from the set, top one facing up	As a player, I want to draw four cards if the cards. match for the round, so that I can play my turn for war in current round of war. Acceptance Criteria: Display four drawn card per player each round in war game mode.
Ability for entering a game session	As a player, I want to enter a game session so that I will know that I am in a game. Acceptance Criteria: Players must be displayed with the number of their cards to confirm they have successfully entered the game session.
Ability for viewing opponent's name while in the active game session	As a player, I want to know the name of my opponent, so that I can know whom I am playing against. Acceptance Criteria: Display the opponent's name in the game session
Ability for viewing active turn for player while in the active game session	As a player, I want to know who's turn is it, so that I can know when it is my turn. Acceptance Criteria: Display the active turn
Ability for viewing number of cards in the players pile or stack	As a player, I want to know how many cards I have left per round, so that I can keep track of my cards. Acceptance Criteria: Display number of cards for the player.
Ability for player to know they are in active war	As a player, I want to know the type of game mode, so that I know whether I am in a war mode or a normal mode of the game mode, in the game session. Acceptance Criteria: Display game mode in the game session.
Ability for player to know the round has ended	As a player, I want to know that the round has ended, so that I can keep track of each rounds. Acceptance Criteria: Display number of rounds in the game session.

FIGURE 2: USE CASE DIAGRAM

This user case diagram is based on the high-level requirements and the code that is currently under development, keeping in mind of the high-level requirements.

FIGURE 3: CLASS DIAGRAM

The following class diagram is based on the business requirements (high-level requirements) and the code for the game, that is currently in development.



2. DESIGN CONSIDERATIONS

Describe the Class Diagram you delivered above (it should be described as Figure 1 or Figure x if you have more than one Figure), explaining the associations and multiplicities depicted.
Comment on each of the following as it pertains to the class groupings you have decided upon and if you have included methods, modifiers and return types, comment on those here as well. You may wish to describe any data structures you wish to use (i.e an enumeration) when you are explaining your design choices. Be specific for full

credit.

Since previously, we were instructed to use only aggregation in class diagrams to define the associated links between classes. The diagram shows aggregated connections only and not composited ones.
Here I will describe the class and their associated connections.
I used inheritance for player (sub class of person class,) which extends the person class (super class of player class.) Player class and Deck class are associated with RegisteredPlayer class by composition, Card class is associated with Deck class by composition. RegisteredPlayer is associated with Game class by composition. GameMode class, ScoreBoard class and GameState class are associated with Game class by aggregation, Game class is associated with GameSession by composition and GameLogic class is associated with GameSession class by composition. Rank and Suit, two enumeration classes were made that are associated with Deck class by association. Each model class is fully encapsulated, and I only used getters in the model classes with the initializers (constructors.) I also used an overridden toString() method in each class, so I will have a clear view of what I am doing as I am testing this manually before unit testing while building the code base for the card game war.

After thorough assessment of the requirements, the updated class diagram in (figure 3) follows Model View Controller (MVC) design pattern. Restructure of these classes defined in class diagram is follows:

MODEL:

Person (Abstract)
Player (Subclass, Composite)
Card (Composite)
GameMode (Aggregated of Game, super class of Turn and Round)
GameState (Aggregated class to Game)
ScoreBoard (Aggregated class to Game)

CONTROLLERS:

Rank (Associate of class Deck, Enumeration)
Suit (Associate of class Deck, Enumeration)
Deck (Composite)
RegisteredPlayer (Aggregated)
Game (Composite)
GameSession (Composite)
GameLogic (Composite)

VIEW:

TestSimulation(main method)

I chose this approach because it allowed me flexibility and maintainability of the code. This approach also allowed me to apply inheritance (Player extends person) and encapsulation in the model classes (Person, Player, Card, GameMode, GameState, ScoreBoard.) Using high cohesion and loose coupling in model classes as well as controller classes have given me a clear perspective as to what needs to be done to meet all the requirements in the final version of the project. The code uses delegation in RegisteredPlayer class, Game class and GameSession class.

- Encapsulation:
  All the model classes are fully encapsulated. Each class have attributes respective to that class only, has an initializer (constructor) and getters respective to each attribute of that class.

- Delegation:
  There is a toString method to evaluate that the instances made in the controller classes are accepting the arguments passed to the instance accordingly.

- Cohesion:
  The code structure uses high cohesion. Each controller class has functions that are relevant to that class and perform as such. For example, in figure 3, RegisteredPlayer class, has a method called Register Player, that registers a player and assigns him/her a set of cards before letting him/her enter the game.
- Coupling:
  The code structure uses loose coupling. Each model class is independent of each other.

- Inheritance:
  The code uses inheritance. Player extends person which is an abstract class.

- Aggregation:
  The code structure uses Aggregation. For example as explained above, ScoreBoard class associated with Game class by aggregation.
- Composition:
  The code structure uses Composition. For example as explained above, Card class associated with Deck class by composition.
- Flexibility/Maintainability
  By applying MVC code pattern, the code base of this project has become flexible, I am able to have a clear view of the final project keeping all the user requirements in mind and I am also able to maintain the code with ease.