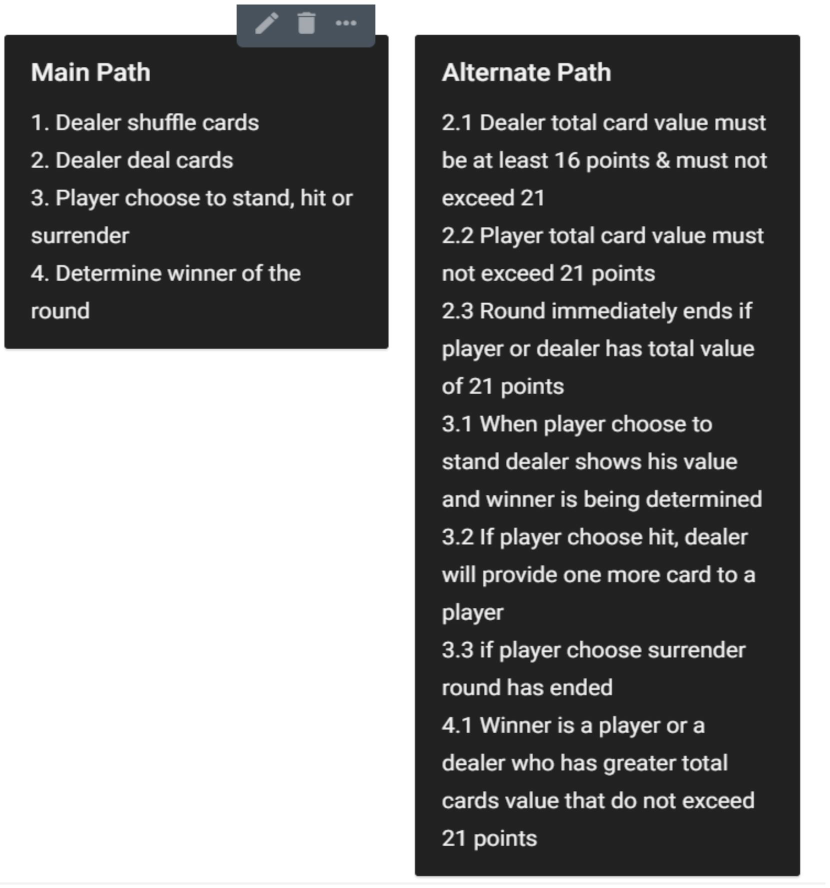
SYST 17796 DELIVERABLE 2
DESIGN DOCUMENT TEMPLATE

OVERVIEW

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”).

Our game of choice is Blackjack card game. Game is being played on a table divided on two sides where on the one side there is a dealer who deals the cards, and on the other side we have players. Main goal of the game is players are trying to beat a dealer by reaching the sum of the cards score as high as possible but not going over value of 21. Game starts when dealer shuffle the cards and start dealing two cards one by one per player and to himself. When the cards are being provided round has started. Player can choose to ask for more cards or to stay with cards already provided. However, when he buys a card and sum exceeds value of 21 the player has lost a round. If his sum of cards value does not go over 21 and it is a greater than dealers’ value, he has won the round. Also, when dealer value exceeds 21 points the player has won the round. Lastly, message saying who is a winner of the round will be displayed. Below Figure 1 shows our main and alternate path steps of how the game round is being processed.



Main Path

1. Dealer shuffle cards
2. Dealer deal cards
3. Player choose to stand, hit or surrender
4. Determine winner of the round

Alternate Path

2.1 Dealer total card value must be at least 16 points & must not exceed 21
2.2 Player total card value must not exceed 21 points
2.3 Round immediately ends if player or dealer has total value of 21 points
3.1 When player choose to stand dealer shows his value and winner is being determined
3.2 If player choose hit, dealer will provide one more card to a player
3.3 if player choose surrender round has ended
4.1 Winner is a player or a dealer who has greater total cards value that do not exceed 21 points

Figure 1

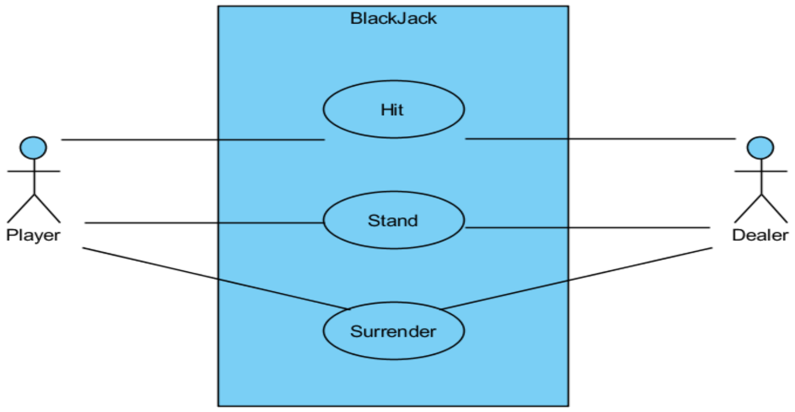


Figure 2

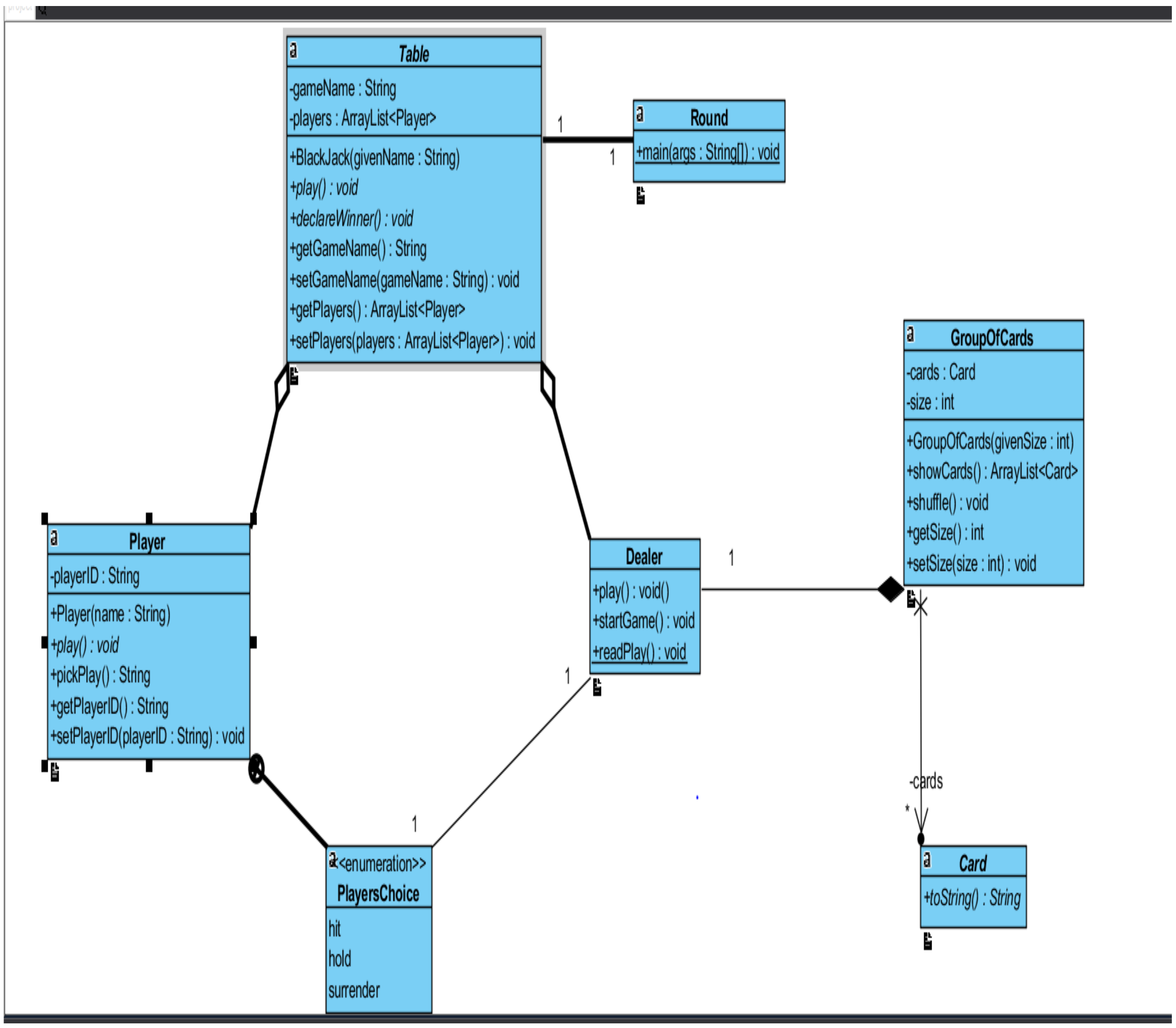


Table
-gameName : String
-players : ArrayList<Player>
+BlackJack(givenName : String)
+play() : void
+declareWinner() : void
+getGameName() : String
+setGameName(gameName : String) : void
+getPlayers() : ArrayList<Player>
+setPlayers(players : ArrayList<Player>) : void
1
1
Round
+main(args : String[]) : void
GroupOfCards
-cards : Card
-size : int
+GroupOfCards(givenSize : int)
+showCards() : ArrayList<Card>
+shuffle() : void
+getSize() : int
+setSize(size : int) : void
Player
-playerID : String
+Player(name : String)
+play() : void
+pickPlay() : String
+getPlayerID() : String
+setPlayerID(playerID : String) : void
Dealer
+play() : void()
+startGame() : void
+readPlay() : void
1
1
1
-cards
*
Card
+toString() : String
<<enumeration>>
PlayersChoice
hit
hold
surrender

Figure 3

2. Design Considerations

Describe the Class Diagram you delivered above (it should be descried 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.

- Encapsulation - is used in order to keep the data more private. This will be done by most of all class variables as setter and getters methods
- Delegation - is achieved by separating our readPlay and pickPlay methods. We do this to pass player input from pickPlay to readPlay in order to dived the tasks and not overload our methods
- Cohesion - by separating the methods play, pickPlay, and readPlay we ensure that while all 3 pieces are needed for 1 players turn, they have clearly defined roles which also makes them more reusable.
- Coupling - we are aiming for loose coupling so that classes are more independent from each other so that when a change is made to one class it will not affect the other
- Inheritance is used with the card and group of cards classes. A card has basic attributes and the group of cards has those attributes as well as others such as number of cards and sequence of cards if the game goes into war mode
- Aggregation - represents a Has-A relationship, Our Table will have aggregation relationships with dealer and player
- Composition - the group of cards will have a Composition relationship because the dealer cannot function without the deck of cards.
- Flexibility/Maintainability - we are trying to keep this project as flexible as possible practicing all OO principles and by adding lots of notes and comments it should easy for anyone to come in and maintain.