ADVANCED OBJECT-ORIENTED PROGRAMMING

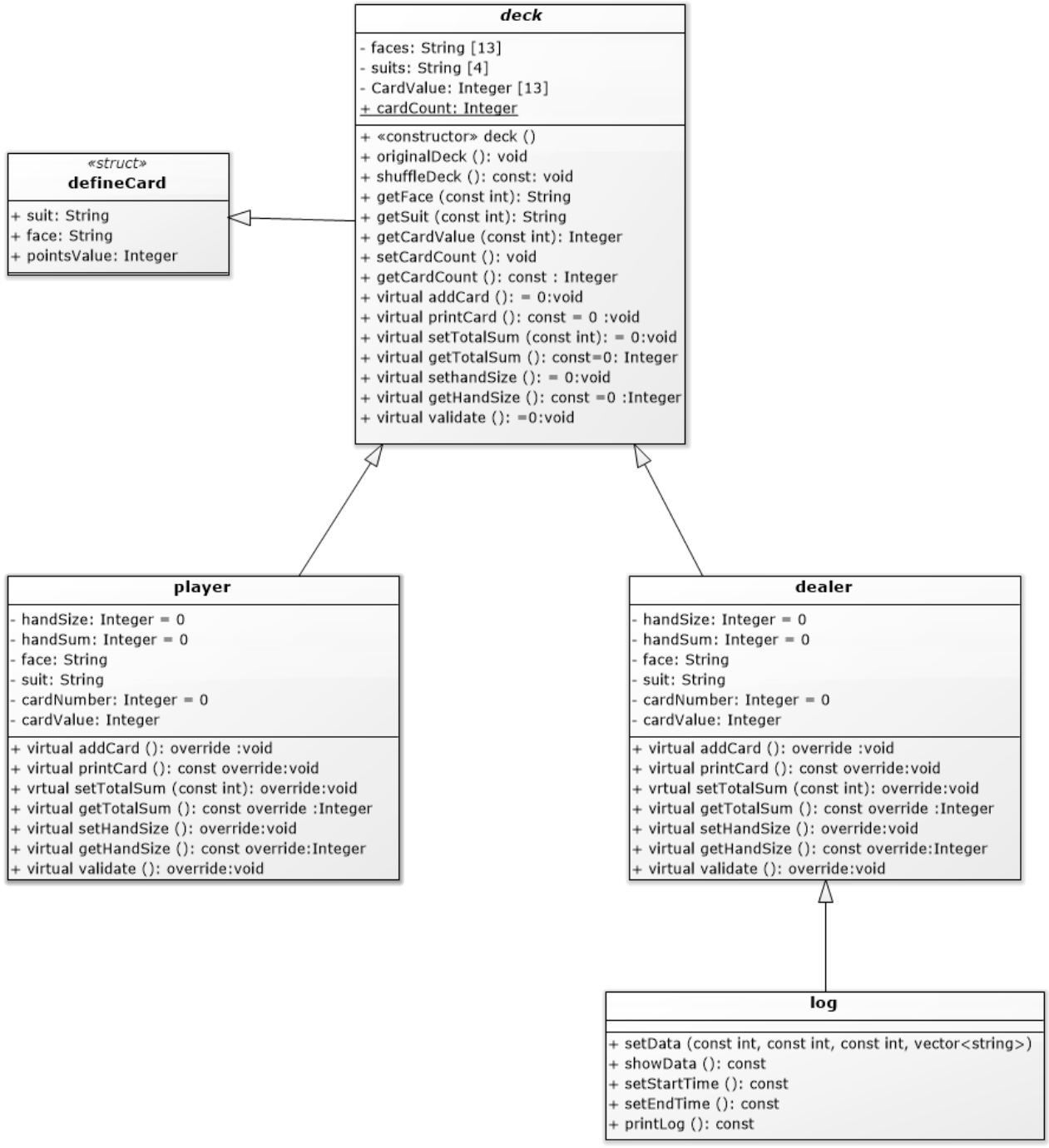
Project Assignment

A Card Game–"Pontoon"

Name:- Anket Dhoble
Student Id:- 180225985

In this project, I use the concept of object-orient programming to create a Card Game- "Pontoon". Here, I used the Eclipse IDE for writing code and MinGW GCC.

- The UML Diagram is shown below describes the flow of my work:-



Cards: -

- As shown in the UML diagram, for storing the card face, suit and value, I used defineCard data structure to store the card face, suit & value because it allows us to store a group of data elements, which grouped together under one name. these data elements are known as members, can have different types and lengths.
- The main advantage of using a data structure is:-
    - Structures basically help you to create your own datatype having a group of primitive types like int, float, char, string.
    - Complexity can be reduced using the concepts of divide and conquer.
    - Logical structures ensure a clear flow of control.
    - Increase in productivity by allowing multiple programmers to work on different parts of the project independently at the same time.
    - Modules can be re-used many times, thus it saves time, reduces complexity and increases reliability.
    - Easier to update/fix the program by replacing individual modules rather than a larger amount of code.

Deck:-

- A deck contains the set of 52 unique playing cards.
- The deck class is used to generate the 52 unique playing cards with a combination of Diamonds, Spades, Hearts, and Clubs.
- In this class, I created a function by name of originalDeck() which is used to generate the straight 52 unique cards and this straight cards will be stored in the defineCard data structure by using the object i.e Deck[52].
- In this class, I have created the string array for faces, suits and integer array for cardValue where the values are statically defined.

❖ originalDeck() :-

- in this function, it will generate 13 unique cards for each suit and after the 13 cards of each suit, it will generate the same 13 unique cards for another suit.
- For changing the suit after 13 cards of each suit, I have to use the modulo operator to determine that 13 unique cards of each suit are generated and now they need to change the suit.
- So, in this way, it will generate the 52 unique cards and stored in the data structure i.e. defineCard.

- The straight cards are generated from the originalDeck() and now I need to shuffle the cards for playing the game. So, for that, another function i.e. shuffleCards() is used to shuffle straight cards of a deck which is stored in the data structure.

- ❖ shuffleDeck()
    - In this function, I have used the rand() to generate the unique values between 1 to 52, which shuffles all my straight cards and again it will store the cards face, suit, and values in the same data structure.
    - For shuffling the cards, I have used two integer variable which stores the random unique values in every iteration and those two variable denotes the index position of the data structure of straight cards which we were stored in the originalDeck() using the Deck object of defineCard.
    - As the two variable which are holding the unique values are used to swap the face, suit, and value of cards in a deck.
    - So, in this way, I'm shuffling the cards of a deck and storing in the same data structure.
- In this class, I have initialized public static int cardCount to determine which position of the card is going to serve from the 52 cards deck. After serving each card the cardCount increments automatically as I have used the set and get the function to increment the cardCount and to read the cardCount.
- This class is also an abstract class because I have created more than one pure virtual function which is going to be overridden in two subclasses i.e. player and dealer.
- This class mainly used to generate straight cards and shuffle that straight cards. Furthermore, the pure virtual functions I have created which is used in player and dealer class and using the get and set function data is validating properly.

Cards validation and adding the card:-

- In gameplay, the bank/computer/dealer deals the card to itself which is hidden from player and the two cards to the player/human which is visible to the player and dealer as well.
- So, the class player is used to add the cards to the player hand and the dealer class is used to add the cards to the dealer's hand.
- In player class, the validate() is used to validate the player hand. So it includes:-
    - If the game is new and the player having zero hands then it will the addCard() which adds 2 cards in the player hand and it will the sum the value of the cards and save it in handSum variable.
    - If the game is in play and then it will check for handSum value if the sum value is less than or equals to 21 then only it will add a card.
    - Again here, I have created the defineCard object for the player to store the 5 hands of the player.
- In dealer class, the validate function works as same as the player but the first two cards are not visible to the players and for the dealer as well, I have created the defineCard object for storing the dealer's hands.

Log generation:-

- For storing the statistic of each game, I have used the file concept.
- For storing the details of each round I have used a vector here for storing the record of round lost or won.
- And the other details like start time and end time are handled by the log class which is having a separate function for startTime() and endTime(), which is used called in the main() when the game starts and at the time of game ends.
- The setData() takes the arguments round, won, lost, roundStatus. So, here the round gives us the number of rounds the player played, won gives the count of how many rounds player won and same for the lost, roundStatus is the vector which gives us the exact status of each round in terms of won or lost.
- Then the setData() writes all the data in the file and printLog() will print the statistics of the file when the player doesn't want to play the game.
- For adding the current time, I have included the time.h file for generating the local time.

Exception Handling:-

- The inputs from the user are not as expected then it will handle the exception by asking again for valid input.
- I have handled the exception when taking input for "[s]tick and [t]wist" & as well as for "do you want to play again [y]es/[n]o".
- In this way, I handled the exceptions and it will protect my program from crashing.

Conclusion:-

According to the specification, I have designed this game modules which can be reused if we want to add some more functionality. And during this project I have learned many things such as how to use the data structure to store the different data types under single name. Furthermore, the constructor of deck class contains two function for generating the straight cards of deck and shuffling function which gets called automatically as the program executes. So, I have achieved the expected output by using object oriented concept.