**Name: Sahenjit Paul**
**ID: 190698348**

# ADVANCED OOP REPORT

## Introduction

In this project I was assigned to create two types of Card Games. The first type of Card Game I was assigned to create was a **Card Guessing Game** whilst the second type of Card Game was a **Blackjack Game.**
The class Hierarchy focuses on Two super classes. The first super class is the **Deck** class and the second super class is the **Game** class. My derived classes from the Deck superclass is **Blackjack Deck** and **Card Guessing Deck.** My derived classes from the Game superclass is **Card Guessing Game** and **Blackjack Game.**
With the help of object-oriented techniques such as inheritance, encapsulation & polymorphism I was able to create generality and reusability so that I can create brand new card games without creating entirely new classes.

## Compiling the code

In order to compile the code, I used the XCode MAC IDE which runs on the clang compiler.

## Approach to the Project

### Card Class:

I started initially by creating a Generic Card class which allows a user to make a playing Card. Within the class, I create two data member variables which all cards have. These are the card face and the card suit (For example: Ace of Spades). I made these data members **private** so that they cannot be accessed directly by the user, this is to ensure data integrity and to provide information hiding.
I created the constructors. The first constructor was a default constructor where I set the face and the suit to **NULL** values. The second constructor was the parametrized constructor where I initialize the data members in my Card class using **this** pointer.
The next step was creating the getters and setters in- order to gain access and change the private data members within the Card class.

### Deck Class:

As mentioned above, the Deck class is my first superclass from which **Blackjack Deck** and **Card Guessing Deck** are derived from. The Deck class is **composed** of cards (Card class). I decided to use composition instead of inheritance since a Card class isn't the parent of a Deck class.
Within my Deck class I have two arrays which are **protected**. The first array contains the 4 different suits every card has. The second array contains the 13 different faces every card has.
Within the **public section** of my Deck class I create an empty vector of type Card (called **deck of cards**) which is yet to be populated. A default constructor is created for initialization. A destructor is created in order to destroy any instance of the Deck class. The data structure I chose is a vector as it has a fast-random access time which is needed when I shuffle the deck at random. Also, it has fast insertion time when elements are being pushed into the vector from either end points.

The **first member function** that is created is the **populate function**. Within this function, I create two for loops. The outer loop that iterates over the suits array (4) whilst the inner loop iterates over the faces array (13). Using these two loops I populate my vector **deck of cards** with the parametrized Card constructor and the two parameters being the suits and faces array.

The **second member function** I create is the print function. Within this function, I create an iterator which iterates over the populated vector created in the previous member function. Each iteration prints sequentially the deck of cards. The iterator prints out the club's suit, the diamond suit, the hearts suit, the spades suit and all the faces that go with those suits.

The **third member function** I create is the shuffle function. Within this function, I create a seed which is based on the current time. Using the **default random engine** which comes in-built with the random STL, I randomize the seed. Using the randomized seed, I shuffle the populated deck using the shuffle function. This randomizes my deck giving me a different value each and every time.

The **fourth member function** I create is the deal function. The deal function is of type Card as I am returning the top card of the deck. Within this function, I create a local object of type Card. In this object I store the card at the back of the shuffled populated deck using the in-built vector function (.back). After storing the card within the object, I delete the card that has been stored in the object using the inbuilt vector function (. pop_back). So, the next time the deal function is called the next card can be dealt. Finally, I return the object as the return type.

The **fifth- and sixth-member functions** I create is the deck conversion suit and face. These functions are abstract in the Deck class as they have no implementation in the superclass and can only be modified in the derived classes. This is because different card decks have different implementations for their respective decks. This allows generality and reusability.

As I have implemented the shuffle, print and populate functions in my superclass I can use them in my derived classes. This allows reusability for any deck classes as you do not have to re design the populate, shuffle and print functions again for the new classes.

## Game:

The Game class is my second superclass, this class is quite simple. I create a default constructor in order to initialize the game class.
I create **two abstract member function** within this class. The first abstract function is menu and the second one is play. These two functions have no implementation is the superclass and can only be modified in the derived class. This allows generality and reusability.

## Card Guessing Deck:

The Card Guessing Deck is one of the two derived classes from the Deck superclass. Within this class, I create my constructor in- order to initialize the class.
I define my abstract member functions (deck conversion suit and deck conversion face) within this class so that they can be used when playing the Card Guessing game.

When creating the deck conversion suites, I rank **clubs as the lowest suit, diamonds, hearts and finally spades as the highest suit.**
When creating the deck conversion faces, I rank **2 as the lowest and Ace as the highest.**

## Black Jack Deck:

The Blackjack Deck is another derived class from the Deck superclass. Within this class, I create my constructor in- order to initialize the class.
I define my abstract member functions (deck conversion suit and deck conversion face) within this class so that they can be used when playing the Blackjack game.
When creating the deck conversion suites, I rank **all suits the same**.
When creating the deck conversion faces, I rank **2 as the lowest point card whilst 10, Jack, Queen, King as 10 points each and Ace as 1 point as per instructed by the rules of the game.**

## Card Guessing Game:

The Card Guessing game is a derived class from the game superclass. This class is **composed** of the Card Guessing Deck not **inherited**.
The **private member variables** of this class are composed of **two objects** of the type **Card Guessing Deck.**
The **public member functions** of this class consist of the **constructor** which is used to initialize the class, **menu and play functions** which are defined within this class and are inherited from the Game superclass where these two functions are abstract.

**Within the play function**, I start by populating the deck by calling the populate function from the deck class thorough the Card Guessing Deck object as the populate function is inherited to the Card Guessing Deck class from the Deck class. I shuffle the deck in a similar fashion in- order to randomize. After randomizing the deck, I deal the top card of the deck using the deal function and store that top card in a Card variable called **top card**.
I pass the top card variable as a parameter for the **deck conversion member functions** within the Card Guessing Deck class in order to change the face and suit from strings into integers. I store the face and suit integers within two local variables with types integer.

The next step I do is create a while loop. Within this loop, I allow the user(player) to choose a face and a suit which he believes has been randomly chosen. I push that face and suit into the card constructor and convert the string inputs into two different integers using the **deck conversion member functions**.
Then using multiple IF statements I compare whether the card the player has chosen is higher or lower than the card selected by the computer at random displaying multiple error messages if this is not the case and increasing the number of attempts after the user keeps trying to guess the card picked at random. Finally, when the user guesses the correct card I show the no. of attempts it took the user to guess the correct card and thus break the loop.

**Within the menu function, I** create another while loop which allows the user to play multiple card guessing games as I give the user the choice to play another Card guessing game if he selects **yes** and if he doesn't want to play another game he can select **no** which breaks the loop and ends the game.

## Blackjack Game:

Similarly, to the Card Guessing game the Blackjack Game is another derived class from the game superclass. This class is **composed** of the Blackjack Deck but not **inherited**.
The **private member variables** of this class are composed two objects of the type Blackjack Deck. It is also three integer variables: the no of rounds the user has played, rounds won and rounds lost.
The public member functions of this class consist of the **constructor** which is used to initialize the class, **menu and play functions** which are defined within this class and are inherited from the Game superclass where these two functions are abstract. **Deal card player** and **Dealer card dealer functions** which keep track of the players and dealer scores. **Round stats function** which outputs how many rounds the player has won and lost. **Player choice function** gives the player the choice in whether to stick or twist.

**Within the play function**, I start by populating the deck by calling the populate function from the deck class thorough the Blackjack Deck object as the populate function is inherited from the Deck class. I shuffle the deck in a similar fashion in- order to randomize. I also create 6 integer variables and a character variable in order to store the **dealer/player score**, the **dealer/player choice,** the number of cards the **dealer/player has chosen** and to allow the user to input their choice in whether to stick or twist. After randomizing the deck, I deal the top two cards of the deck using the deal function twice from the deck class and storing those two top cards in Card variables called p & p1.
I pass the card variable, player score, a Boolean variable to tell the compiler that this is a player and the no of cards the player chosen as parameters to the deal card player function. The players score and the card counter (no of cards) are **passed as references**. This is because I want don't want the memory address of the player score to change but rather be updated.
Within the **deal card player function** I convert the top card given to the player from a string to an integer and store it within a local variable called score. The variable score keeps updating each and every time a new card is given and added to the total. The deal card player function also tells the player which card he has been given at random from the top of the deck and the associated value (points) of that card.
I create a while loop, within this loop I have three IF statements. The first IF statement checks whether the player score is greater than 21 and the number of cards dealt to the player has exceeded 5, if either of these conditions are met the player has lost the game and the loop is broken. The second condition checks whether the player score is equal to 21 and the number of cards dealt has exceeded 5. If these conditions are met the player has won the game and the loop is broken. Finally, my last if statement checks if the player score is less than 21 and the number of cards dealt is less than 5. If this condition is met the player is given a choice whether he wants to stick or twist. If player choose to twist he gets dealt

another card and is re-evaluated at each IF statement. However, if he chooses to stick the loop ends and it's the dealers turn to play.

If the player chooses to stick the first two cards the dealer has been dealt is shown and the respective scores of those two cards are shown using the **Deal card dealer function**.
Similarly, I create a while loop where the dealer's score is evaluated using IF statements.
If the dealer score is greater than 21 and the number of cards dealt is less than or equal to 5 the dealer has lost and the player has won and the loop is broken. If the dealer score is equal to 21 and the number of cards dealt is less than or equal to 5 the dealer has won and the loop is broken. If the dealer score is less than the players score and the number of cards dealt is less than 5 the dealer draws another card. If the dealer score is greater than the players score and the dealer score is less than 21 the dealer has won and the loop is broken.

**Within the menu function, I** create another while loop which allows the user to play multiple Blackjack Games as I give the user the choice to play another Blackjack Game if he selects **yes** and if he doesn't want to play another game he can select **no** which breaks the loop and ends the game.

Within the **Round stats function,** I keep a counter of how many games the player has won and lost and is viewed when the player decided to end the game.

## MAIN:

In the main function, I run my games.
In order to do this, I create an object of the Card Guessing Game and an object of the Blackjack Game. I create a pointer to an object for the Game class and I can't create an actual object of the game class since the class is abstract.
I allow the player to choose which Game he would like to play by giving the player a selection choice. If the player chooses 1 the memory address of the Card guessing Game object is assigned to the game object pointer. Therefore, allowing me to access the menu for the Card Guessing Game.
If the player chooses 2 he can play the Blackjack Game in a similar fashion.

# UML DIAGRAM

Here is a simple UML diagram of the program layout: