**Link to Code:**
**https://repl.it/@RacheySnachey/c11hwstartblackjackcards#main.cpp**

# Goal

Implement a simplified card game of BlackJack, where one player is pitted against a dealer to see who can reach 21 points first, without going over.

- The game will use a Class Hand to describe a collection of cards. The Deck that is dealt from, the Dealer's "current hand of cards" and the - Player's "current hand of cards" will each be represented by an instance of a Hand.
- The Hand, in turn, will be comprised of Card instances stored in a List container from <list>.
- The entire game will be encapsulated by a BlackJack Class (similar to ConnectFour).

# Requirements

- You will need to implement your game using the Card.h, Hand.h, and BlackJack.h prototypes provided, so that preliminary/automated testing can be performed by the testMe() routine included. You must implement according to these include files and you may not change them.
- You must utilize only the <list> container for representation your Hand class (as specified by the Hand.h file). Note that the <list> container does not support the use of brackets [ ] like a vector or an array, and so you will be required to use iterators as discussed in class.

# Getting Started

After reading this README.md file, begin implementing constructors and methods in the order described below in the section "Constructors and Methods you Must Implement"

# Game Rules

Consider this sample as our rough model for how to play the game (ignore any betting - betting is not necessary for this assignment).

- https://html5blackjack.net/ (Links to an external site.)

The following specific game rules must be enforced by your game, in the following priority

1. The Player's and Dealer's card's should be printed face up always (to help us debug the game). This would not normally be done in Vegas!
2. The game starts with a full deck and an empty hand for both the dealer and the player (This would not normally be done in Vegas)
3. Each hand is scored by adding up each card's point value, where Card values will be assigned as shown below:

- A) Aces will be treated as 1 (see the BONUS for treatment of an Aces as 1 or 11)
- B) 2 through 9 will have the corresponding values 2 through 9
- C) Jack, King, and Queen will have a value of 10 always

5. The Player goes first, and then the Dealer
6. The Player gets to "Hit" (take a card) as many times as they like, as long as their point total is <= 21. The player can "Stand" (pass their turn to the dealer) at any time. If the player goes over 21, the game is over (the Dealer wins automatically, without even taking a turn).
7. Otherwise, a winner is not considered until after the Dealer's turn is complete:

- A) If the Dealer's cards have a total value of exactly 21, the Dealer wins no matter what score the player has
- B) If the Dealer's cards exceed 21, and the player's cards are <= 21, the Player wins
- C) If the Dealer's and the Player's score are both equal, the Dealer wins.

# Sample Output

Your game output could look something like this (but it does not have to match it!):

```
Let's Play BlackJack!
Try to get closest to 21 without going over
Dealer's hand: { } 0 points
Player's hand: { } 0 points

Player's Turn: Hit (h) or Stand (s)? h
Player hits...
Dealer's hand: { } 0 points
Player's hand: { Two of Hearts(2), } 2 points

Player's Turn again: Hit (h) or Stand (s)? h
Player hits...
Dealer's hand: { } 0 points
Player's hand: { Queen of Hearts(10), Two of Hearts(2), } 12 points

Player's Turn again: Hit (h) or Stand (s)? h
Player hits...
Dealer's hand: { } 0 points
Player's hand: { Six of Spades(6), Queen of Hearts(10), Two of Hearts(2), } 18 points

Player's Turn again: Hit (h) or Stand (s)? s
Player Stands.
Dealer's hand: { } 0 points
Player's hand: { Six of Spades(6), Queen of Hearts(10), Two of Hearts(2), } 18 points

Dealer's turn...
Dealer Hits..
Dealer's hand: { Seven of Spades(7), } 7 points
Player's hand: { Six of Spades(6), Queen of Hearts(10), Two of Hearts(2), } 18 points

Dealer Hits..
Dealer's hand: { Ace of Diamonds(1), Seven of Spades(7), } 8 points
Player's hand: { Six of Spades(6), Queen of Hearts(10), Two of Hearts(2), } 18 points
```

Dealer Hits..
Dealer's hand: { Three of Spades(3), Ace of Diamonds(1), Seven of Spades(7), } 11 points
Player's hand: { Six of Spades(6), Queen of Hearts(10), Two of Hearts(2), } 18 points

Dealer Hits..
Dealer's hand: { Ten of Clubs(10), Three of Spades(3), Ace of Diamonds(1), Seven of Spades(7), } 21 points
Player's hand: { Six of Spades(6), Queen of Hearts(10), Two of Hearts(2), } 18 points

Dealer wins with 21.
Congratulations: Dealer has won the game!
Goodbye!

# Files and Classes

You will begin with the following files which have been provided for you (and may not be changed):

- **Card.h** - provided for you (may not be modified)
- **Hand.h** - provided for you (may not be modified)
- **BlackJack.h** - provided for you (may not be modified)
- **Card.cpp** - this file has been provided for you, but you may change it as needed (but you should not need to)
- **main.cpp** - this file has been provided for you, but you may change it as needed

You will need to finish implementation of these classes, along with main() via the following files:

- Hand.cpp - you will need to implement the methods in this file according to Hand.h
- BlackJack.cpp - you will need to implement the method in this file according to BlackJack.h

# Constructors and Methods you Must Implement

## BlackJack.cpp

- BlackJack::play() - has been implemented for you.
- BlackJack :: BlackJack( ) - Copy the player names into the appropriate BlackJack member variable. Pass the cardIDs vector to the Hand constructor to create the deck. This vector will ensure that the deck that is used for the game is contains the specific cards requested in main(). Note that the Dealer and Player "hands" ( dealerHand and playerHand ) should remain empty.
- void BlackJack :: printInstructions() - simple explanation of the game.
- void BlackJack :: printBoard() - print both the Dealer and Player hands in their current state (list all cards). ( It is not necessary to print the entire deck that will be dealt from by the dealer).
- int BlackJack :: getwinningPlayerId() - return -1 (no winer yet), 0 (Player has won), or 1 (Dealer has won), based on the private winningPlayerId member variable.

- bool BlackJack::takePlayerTurn() - prompt the player to either hit (take a card) or stand (pass the turn to the dealer). The player can hit as many times as they like until they win or bust. If the player "busts" by going over 21, set the winningPlayerId value to 1 for the dealer and return true. Otherwise set the winningPlayerId value to -1 and return false. If the player's total hand value after a hit is exactly == 21, the player's turn is over (they must stand), and the turn passes to the dealer (return false and set winningPlayerId to -1, because the game is not over). You will find it helpful to print out status messages after each "hit" taken by a player.
- bool BlackJack::takeDealerTurn() - As long as the value of the dealer's hand is <=16, the Dealer MUST hit. If the Dealer's hand exceeds a value of 21, the dealer loses, and you should set the winningPlayerId to 0 and return true, indicating that the game is over. Otherwise if the dealer's hand value > 16, and also => the Player's hand value, the dealer wins; so set the winningPlayerId to 1, and return true (game over). If the Dealer's hand exceeds 16 but is still less than the Player's hand, the Player wins: set winningPlayerId accordingly and return true (game over).
- int BlackJack :: getPlayerPoints() - return the points for the player's hand by calling Hand::playerHand->getPoints()
- int BlackJack :: getDealerPoints() - return the points for the dealer's hand by calling Hand::dealerHand->getPoints()

## Hand.cpp

- Hand :: Hand( vector<int> _initialCards) - construct Card instances based on the vector of integer IDs provided (call the Card constructor with each integer ID). Then place the new instance of each Card into the Hand's "cards" member. Note that the cards member is a <list> container and only supports iterators (no brackets).
- Card Hand :: dealACard() - implement the dealACard() method discussed in class which returns a copy of the top Card in the Hand(and removes it from the cards <list>in this Hand.
- void Hand :: addACard(Card newCard) - add a copy of a newly provided card to a this Hand
- int Hand :: getPoints() - return the total value of this current Hand (used for "hands" of cards held by the dealer and the player. Calculate the total by adding up the value of each of the cards in the cards <list> of the Hand.
- void Hand :: printMe() - print out a representation of the Hand by utilizing the overloaded ostream method in the Card.h file to print out each card in the Hand.
- bool Hand :: hasAnAce() - (BONUS ONLY) return true if this Hand contains a card that is an Ace (has a name == "Ace" )

# Card IDs and Meanings (including Aces)

When we create the initial deck of cards for a game of BlackJack, via a Hand (for example, the full deck that the Dealer deals from), we will describe the deck by a vector of IDs . There are 52 unique ID's we can that represent unique cards: from 0 to 51. Each ID results in the creation of a specific card. The Card constructor has been provided for you so you only need to call it with a valid ID for each card to be created. The table below illustrates the ID that goes with each Card .

Note that each Card has a name, a suit, and a value (used by the getPoints() that are all assigned by the constructor.

| | **Card ID, Suit, & Point Value** | **Card ID, Suit, & Point Value** | **Card ID, Suit, & Point Value** | **Card ID, Suit, & Point Value** |
|---|---|---|---|---|
| ACE | 0 = Ace of Spades (value=1 or 11) | 1 = Ace of Hearts (value=1 or 11) | 2 =Ace of Diamonds (value =1 or 11) | 3 = Ace of Clubs (value =1 or 11) |
| T W O | 4 = Two of Spades (value=2) | 5 = Two of Hearts (value=2) | 6 = Two of Diamonds (value=2) | 7 = Two of Clubs (value=2) |
| T H R E E | 8 = Three of Spades (value=3) | 9 = Three of Hearts (value=3) | 10 = Three of Diamonds (value=3) | 11 = Three of Clubs (value=3) |
| F O U R | 12 = Four of Spades (value=4) | 13 = Four of Hearts (value=4) | 14 = Four of Diamonds (value=4) | 15 = Four of Clubs (value=4 ) |
| F I V E | 16 = Five of Spades (value=5) | 17 = Five of Hearts (value=5) | 18 = Five of Diamonds (value=5) | 19 = Five of Clubs (value=5 ) |
| S I X | 20 = Six of Spades (value=6) | 21 = Six of Hearts (value=6) | 22 = Six of Diamonds (value=6) | 23 = Six of Clubs (value=6) |
| S E V E N | 24 = Seven of Spades (value=7) | 25=Seven of Hearts (value=7) | 26=Seven of Diamonds (value=7) | 27=Seven of Clubs (value=7) |
| E I G H T | 28 = Eight of Spades (value=8) | 29=Eight of Hearts (value=8) | 30=Eight of Diamonds (value=8) | 31=Eight of Clubs (value=8) |
| N I N E | 32 = Nine of Spades (value=9) | 33=Nine of Hearts (value=9) | 34=Nine of Diamonds (value=9) | 35=Nine of Clubs (value=9) |
| T E N | 36 = Ten of Spades (value=10) | 37=Ten of Hearts (value=10) | 38=Ten of Diamonds (value=10) | 39=Ten of Clubs (value=10) |

| | | | | |
|---|---|---|---|---|
| J A C K | 40 = Jack of Spades (value=10) | 41=Jack of Hearts (value=10) | 42=Jack of Diamonds (value=10) | 43=Jack of Clubs (value=10) |
| Q U E E N | 44 = Queen of Spades (value=10) | 45=Queen of Hearts (value=10) | 46=Queen of Diamonds (value=10) | 47=Queen of Clubs (value=10) |
| K I N G | 48 = King of Spades (value=10) | 49=King of Hearts (value=10) | 50=King of Diamonds (value=10) | 51=King of Clubs (value =10) |
| J O K E R | 52 = Ace of Spades (value=-1) | | | |

# Hints

- Any card ID modulo 4 will provide an integer 0 through 3, which can be correlated to spades(0), hearts(1), diamonds (2), or clubs(3)
- any card ID can be divided by 4 using integer division to determine the card's name based on the integer result: 0 (ACE), 1(TWO), 2(THREE), ... 12 (KING), 13 (JOKER)
- The hand class has only one member: std::list<Card> cards. This means that an instance of Hand created by Hand myHand({0,1,2,3}) will have a PRIVATE member this->cards, which which is a <list> contain of Card instances.

# Scoring

- **90 Points**: Automated testing will be responsible for most of your grade (see testing below)
- **10 Points**: Code Cleanlines & code comments, User-Feedback clarity and "neatness" of presentation. Simply consider whether an uninitiated student who does not know how to play the game would be able to play your game (and enjoy it).

# Testing (optional - but highly useful)

A testMe.cpp file and a testMe.h file, along with a testMe() function have been provided. You will also need thetestMeIn.txt file that controls input to the game during testing. When you are ready to assess your grade on this project, you should uncomment the testMe() method in main.cpp and observe your preliminary score. Note that the test will create (overwrite) a verbose output file called testMeOut.txt if you want more detail on why a particular test may have failed. All scores are subject to final review by the instructor. And of course you are not allow to rewrite testMe() to your advantage. :)

# Bonus

Implement the following tests to add more bonus points to your overall score:

- 5 Points: Implement the method Hand :: hasAnAce() - return true if this Hand contains a card that is an Ace (has a name == "Ace" )
- 5 points: If at any time after taking a card (a Hit) the Player has an Ace and any other single card that is worth 10 points, for a total of 21 points, that player wins immediately (and the Dealer loses). Use hasAnAce() to help determine this without changing any .h files

# Submission

If you are using repl.it simply submit your repl link.
If you are working outside of repl.it ONLY submit the files you have modified. (thank you)
No PDFS please.

# Concepts Covered

- Working with Classes (advanced)
- Out-of-Class Implementations for methods and class constructors
- .h files
- Working with multiple embedded (composed) classes
- List Container and iterators (not using brackets!)
- Vectors (of string and int)