Blackjack Report

How to run

This program can be most easily run by opening 'B00746055 OOP CW.sln', and then building and running the program with Visual Studio 2017.

If any issues are present, open main.cpp and go to Project -> Properties -> Linker -> System and ensure SubSystem is set to CONSOLE

On launch, a logo will greet the user and they will be prompted to play or change settings. To play the game, the user must input '1' and then select how much money they have to play with, and how much money they want to bet on each round. The player will then play blackjack against the dealer until they run out of money, or until they decide to stop playing. The user receives double their bet if they win a round, they lose their bet if they lose the round, and their bet is returned on a draw. If a user runs out of money, they have the opportunity to return to the main menu if they wish to play again.

Additional features implemented

This program implements a number of additional features. Upon running the program, if the user selects '2' to change settings, they can change the number of decks being shuffled at once. This can be used if the user wants to simulate French blackjack for example, in which 4 decks are often played at once.
The program includes a betting system; users may input how much money they bring with them to the casino, and then can use this as a pool of money from which they may draw to make bets on each round.
There are checks for blackjack, so he round is not played if the player or dealer has a natural 21.

Requirements table

Requirement	Implemented	Filename	Line
Use of classes/objects	Yes	Hand.cpp Card.cpp Deck.cpp	All over. Main.cpp line 27
Applications of Pointers	Yes	Hand.cpp	56 Hand contains many applications of pointers, especially with c-style arrays
Manipulation of Vectors/Arrays	Yes	Deck.cpp	65 Vectors are manipulated many times across the program
Using functions	Yes	Main.cpp	This entire file is composed of many functions. main() method only has one statement

Exception Handling	Yes	Main.cpp	33 – many try/catches implemented throughout this file
File Manipulation	Yes	Main.cpp Deck.cpp	Main - 440 Deck - 36
Inheritance	No	In my design stage, I decided against using inheritance as it would not have added any value to my program. Include statements for header files provided all the functionality I needed	
Polymophism	Yes	Main.cpp Hand.cpp Card.cpp	Card 13-23 Main 109,110,171,194
Advances Functionality	Yes	In Deck.cpp, a random generator was used. A Map was used in Deck.cpp Object destructors were used throughout for clean-up Functionality for soft-aces was implemented throughout, as well as a betting system and the ability to use multiple decks	

Class Diagram

Aside from the standard main class, 3 other classes were developed and implemented for this project. These are as follows:

Deck
engine : default_random_engine
Deck() setupDeck() getRandomEngine()

Card
value : int suit : string cardNumber : string secondValue : int
Card() getValue() getSecondValue() getSuit() getValueString()

Hand
player : bool cards : vector<Card> value : int[2]
Hand() isPlayer() clear() dealCard() displayHand() calculateHandValue() checkBust() checkForNaturals() displayTopCard() clearCards() getFirstValue() getSecondValue()

Test plan

1.

Test that if the player has a better hand than the dealer, they win the round

```
Your hand:
FIVE of DIAMONDS
FOUR of CLUBS
THREE of CLUBS
TWO of HEARTS
SEVEN of SPADES
Total hand value:       21
The dealer flips over his second card

Dealer hand:
JACK of CLUBS
QUEEN of HEARTS
Total hand value:       20
Congatulations! You have won this round!
```

2.

Test that if the player has a worse hand than the dealer, they lose the round

```
Your hand:
TEN of CLUBS
JACK of DIAMONDS
Total hand value:       20

Dealer hand:
JACK of CLUBS
The dealer's second card is face-down
The dealer is checking for blackjack...
The dealer has blackjack!
You lose this round. You have lost $10
Your current balance is $90
```

3.

Test that if the player and the dealer have hands of the same value, the round ends in a draw

```
Your hand:
TWO of CLUBS
FIVE of DIAMONDS
EIGHT of CLUBS
FIVE of HEARTS
Total hand value:       20
Type '1' to hit or '2' to stand
2
The dealer flips over his second card

Dealer hand:
TEN of HEARTS
JACK of HEARTS
Total hand value:       20
This round ended in a draw
Your current balance is $900
Type '1' to play another round!
```

4.

Test that upon winning a round, the player's balance is increased by 2 * the bet

```
Your current balance is $800
Type '1' to play another round!
1
How much money do you wish to bet on this round?
100

Your hand:
THREE of HEARTS
SEVEN of DIAMONDS
Total hand value:       10

Dealer hand:
FIVE of HEARTS
The dealer's second card is face-down
Type '1' to hit or '2' to stand
2
The dealer flips over his second card

Dealer hand:
FIVE of HEARTS
QUEEN of HEARTS
Total hand value:       15
The dealer hits
The dealer was dealt the JACK of SPADES
Congratulations! The dealer has gone bust
Congatulations! You have won this round!
Your current balance is $900
Type '1' to play another round!
```

Note that once this bet is made, the bet amount ($100) is deducted from the balance so now the balance is $700

5.

Test that upon a draw, the player's balance is returned to its original level by returning the bet (balance += bet)

```
Your hand:
TWO of CLUBS
FIVE of DIAMONDS
EIGHT of CLUBS
FIVE of HEARTS
Total hand value:       20
Type '1' to hit or '2' to stand
2
The dealer flips over his second card

Dealer hand:
TEN of HEARTS
JACK of HEARTS
Total hand value:       20
This round ended in a draw
Your current balance is $900
Type '1' to play another round!
```

(Original balance was $900)

6.

Test that upon losing a round, the player's balance is decreased by their bet.

```
Your hand:
TEN of CLUBS
JACK of DIAMONDS
Total hand value:       20

Dealer hand:
JACK of CLUBS
The dealer's second card is face-down
The dealer is checking for blackjack...
The dealer has blackjack!
You lose this round. You have lost $10
Your current balance is $90
```

7.

Test that if the dealer's first card is an ace or a spade, they check for blackjack

```
Your hand:
TEN of CLUBS
JACK of DIAMONDS
Total hand value:       20

Dealer hand:
JACK of CLUBS
The dealer's second card is face-down
The dealer is checking for blackjack...
The dealer has blackjack!
You lose this round. You have lost $10
Your current balance is $90
```

8.

Test that if the dealer does not have blackjack, the game continues

```
Dealer hand:
JACK of CLUBS
The dealer's second card is face-down
The dealer is checking for blackjack...
The dealer does not have blackjack.
Type '1' to hit or '2' to stand
```

>>Game continues from here

9.

Test that if the dealer or player has blackjack, the round is not played

```
Your hand:
TEN of CLUBS
JACK of DIAMONDS
Total hand value:       20

Dealer hand:
JACK of CLUBS
The dealer's second card is face-down
The dealer is checking for blackjack...
The dealer has blackjack!
You lose this round. You have lost $10
Your current balance is $90
```

10.

Test that multiple decks can be used at once

```
Welcome to blackjack!
Press 1 to play or press 2 to access game settings. Press any other key to exit
2


How many decks do you want to play with?
Enter a value between 1 and 10
10
   ______     _______.
  /     /\   |   .   |\
 /  '  /  \  | .   . |.\
/_____/. . \ |_______|.'|
\ . . \   /  \ ' .   \'|
 \ . . \ /    \___'__\|
  \_____\/

Welcome to blackjack!
Press 1 to play or press 2 to access game settings. Press any other key to exit
```

```
Your hand:
THREE of DIAMONDS
THREE of DIAMONDS
Total hand value:       6
```

With a single deck in use, it would be impossible to have 2 “THREE of DIAMONS” cards since the deck would only have 1

11.

Test that upon running out of money, the player may exit the application or return to the main menu

```
Dealer hand:
SIX of HEARTS
TEN of SPADES
Total hand value:       16
The dealer hits
The dealer was dealt the FIVE of SPADES
Thank you for playing! Unfortunately you have lost all your money.
Please press 1 if you wish to return to the main menu.
```

12.

Test that if the user “hits” and reaches 21, they are no longer prompted for moves and the dealer’s turn begins

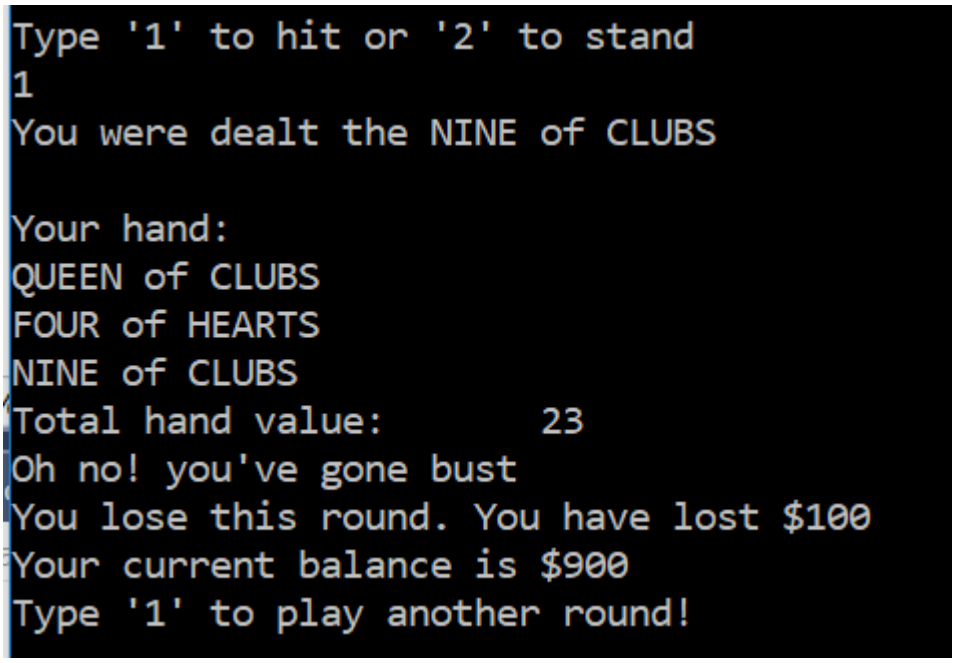
```
Your hand:
FIVE of DIAMONDS
FOUR of CLUBS
THREE of CLUBS
TWO of HEARTS
SEVEN of SPADES
Total hand value:       21
The dealer flips over his second card

Dealer hand:
JACK of CLUBS
QUEEN of HEARTS
Total hand value:       20
Congatulations! You have won this round!
```

No prompt to hit/stand after hand value hits 21

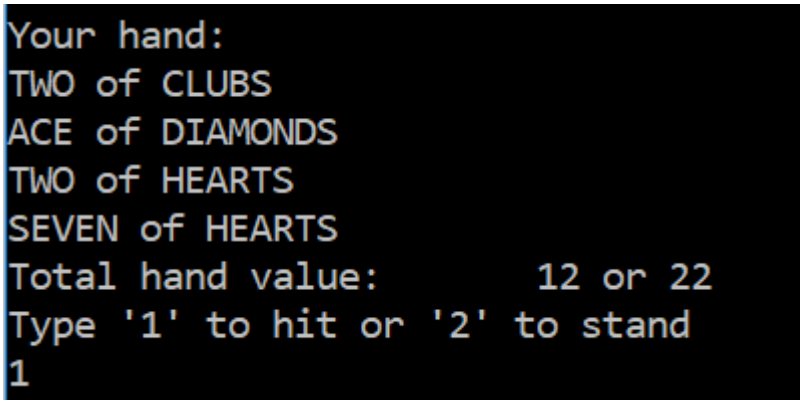
13

Test that if the player hits and goes bust, they lose the round



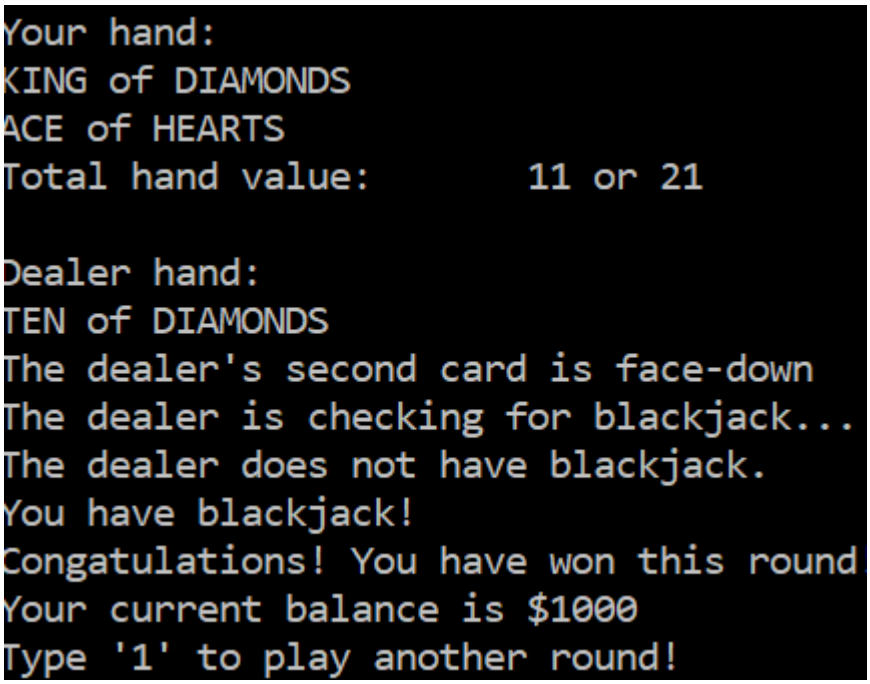
14

Check that if dealt an ace, the player will have 2 possible hand values



15

Check that if the player has blackjack and the dealer does not, the player wins



16.

Test that upon running out of money, player is returned to the main menu

Code Transcript

```
//
//  main.cpp
//  OOP CW
//
//  Created by Michael Dunwoody on 27/10/2019.
//  Copyright © 2019 Michael Dunwoody. All rights reserved.
//

#include <iostream>
#include <vector>
#include <iterator>
#include <map>
#include <algorithm>
#include <fstream>

#include "Deck.hpp"
#include "Card.hpp"
#include "Hand.hpp"

using namespace std;

float balance = 0;
bool handInPlay = true;//control boolean used to make play stop if a hit makes a player bust
float bet = 0;

vector<Card> playingDeck;
Hand playerHandObj = Hand(true);//true as it is a player hand
Hand dealerHandObj = Hand(false);//false as it is a dealer hand
void mainMenu();
```

```
void setBalance(){
   cout << "How much money do you have to play with?"<<endl;
   try {
      cin >> balance;
   }catch(exception ex){
      cout << "Whoa! That's not a number!"<<endl;
      setBalance();
   }
}
void setRoundBet(){//
   cout << "How much money do you wish to bet on this round?"<<endl;
   try {
      cin >> bet;
      if (bet > balance) {
         cout << "You cannot bet more than your balance!"<< " ( $"<<balance<<")"<<endl;
         setRoundBet();
      }
                  else {
                        balance -= bet;
                  }
   }catch(exception ex){
      cout << "Whoa! That's not a number!"<<endl;
      setRoundBet();
   }
   // Player sees the suit and value of both their card and the total value of their hand
   // Can see the value of the dealer's shown cards
   // If a card is an ace, total hand may be calculated as 2 potential values
   // Can decide to 'Hit' or 'Stand'
   // Dealer must hit if their total is under 17
}
```

```
int displayLogo(){
   int choice;

   cout << "   _______     ________."<<endl;
   cout << "  /     /\\   |   .   |\\"<<endl;
   cout << " /  '  /  \\  | .   . |.\\"<<endl;
   cout << "/_____/. . \\ |________|.'|"<<endl;
   cout << "\\ . . \\    /  \\ ' .   \\'|"<<endl;
   cout << " \\ . . \\  /    \\____'__\\|"<<endl;
   cout << "  \\_____\\/"<<endl<<endl;;
   cout << "Welcome to blackjack!\n";

   cout << "Press 1 to play or press 2 to access game settings. Press any other key to exit" << endl;
   try{
      cin >> choice;
   }catch(exception ex){
      return 0;
   }
   return choice;
}

void exitApplication(){
   cout <<"Bye!"<<endl;
   exit(0);
}

void dealCardToPlayer(){
   playerHandObj.dealCard(playingDeck.back());//Gives the player the card at the end of the deck
   playingDeck.pop_back();//Removes that card from the playingDeck
}
```

```
void dealCardToDealer(){
   dealerHandObj.dealCard(playingDeck.back());//Gives the dealer the card at the end of the deck
   playingDeck.pop_back();//Removes that card from the playingDeck
}


void createNewDeck(){
   cout << "Shuffling..."<<endl;
   playingDeck = Deck::setupDeck();
}

void setupGame(){//Method for setting up the beginning of a game
   dealCardToPlayer();
   dealCardToDealer();
   dealCardToPlayer();
   dealCardToDealer();
   playerHandObj.calculateHandValue();
   dealerHandObj.calculateHandValue();
   playerHandObj.displayHand(playerHandObj.isPlayer(), false);
   dealerHandObj.displayHand(dealerHandObj.isPlayer(), true);
}

void clearHands(){
   //Cleanup method to destroy the cards in each player's hand. These cards will not be added back to the deck and are instead 'discarded'. The players will be dealt cards again if they wish to play again, but it will be from an ever-diminishing deck as it would be with a real-life deck of cards.
   playerHandObj.clearCards();
   dealerHandObj.clearCards();
   handInPlay = false;
}
```

```
void displayBalance(){
   cout << "Your current balance is $"<< balance << endl;
}


void playerLoses(){
   handInPlay = false;
   clearHands();
   if (balance > 0) {
      cout << "You lose this round. You have lost $"<<bet<<endl;
      displayBalance();
      //deal them cards again - runGame runs. Maybe run a clearHand method now?
      //prompt for another round?
   }
   else {//If the player loses all their money
      int i = 0;
      cout << "Thank you for playing! Unfortunately you have lost all your money.\nPlease press 1 if
you wish to return to the main menu."<<endl;
      try {
         cin >> i;
      } catch (exception e) {
         exitApplication();
      }
      if (i == 1) {
         cout << "\n\n\n\n\n"<<endl;
         mainMenu();//Make them set a new balance before runGame takes over and prompts bets
      }
      else {
         exitApplication();
      }
   }
}
```

```
void runDraw(){
   handInPlay = false;
   clearHands();
   balance += bet;
   cout << "This round ended in a draw"<<endl;
   displayBalance();
   bet = 0;
}
void playerWins(){
   handInPlay = false;
   clearHands();
   float winnings = 2* bet;
   balance += winnings;
   bet = 0;
   cout << "Congatulations! You have won this round!" <<endl;
   displayBalance();
}
void hitPlayer(){
   dealCardToPlayer();//Deal card to player and check if the player is bust
   playerHandObj.displayTopCard(true);
   playerHandObj.calculateHandValue();
   playerHandObj.displayHand(1, 0);
   if (playerHandObj.checkBust()){
      cout << "Oh no! you've gone bust"<<endl;
      handInPlay = false;
   }
   if (playerHandObj.getFirstValue() == 21 || playerHandObj.getSecondValue() == 21) {
      handInPlay = false;
   }
}
```

```
void hitDealer(){
   cout << "The dealer hits"<<endl;
   dealCardToDealer();//Deal card to player and check if the player is bust
   dealerHandObj.displayTopCard(false);
   dealerHandObj.calculateHandValue();
   if (dealerHandObj.checkBust()){
      cout << "Congratulations! The dealer has gone bust"<<endl;
      handInPlay = false;
   }
}

void revealSecondCard(){
   cout << "The dealer flips over his second card" <<endl;
   dealerHandObj.displayHand(false,false);
}

bool dealerPlay(){
   int hand1 = dealerHandObj.getFirstValue();
   int hand2 = dealerHandObj.getSecondValue();
   if (hand1 < 17 || hand2 < 17) {
      //Next up is programming the dealer 'AI'
      //if both are less than 17, dealer must hit
      hitDealer();
      if (hand1 > 21 && hand2 > 21) {
         return false;
      }
      return true;
   }
   return false;
}
```

```
void determineWinner(int p1, int p2, int d1, int d2){
   if (p1 > 21 && p2 > 21) {
      playerLoses();
   }
   if (d1 > 21 && d2 > 21) {
      playerWins();
   }
   else {
      if (p1 == p2 && d1 == d2) {//no aces
         if (p1 > d1) {
            playerWins();
         }
         else if (d1 > p1) {
            playerLoses();
         }
         else {
            runDraw();
         }
      }
      if (p1 != p2 && d1 == d2) {//if player has an ace and dealer does not
         if (p2 > 21){//player's 11 score is bust
            if (p1 > d1) {
               playerWins();
            }
            else if (d1 > p1){
               playerLoses();
            }
            else {
               runDraw();
            }
         }
```

```
    else {//p2 not bust, therefore we know it's a better hand than p1 where ace=1
        if (p2 > d1){
            playerWins();//player is not bust so if they win either hand they win since d1==d2
        }
        else if (p2 < d1) {
            playerLoses();
        }
        else {
            runDraw();
        }
    }
}
if (p1 == p2 && d1 != d2) {//if dealer has an ace and player does not
    if (d2 > 21) {//if dealer is bust with 11 hand
        if (p1 > d1) {
            playerWins();
        }
        else if (d1 > p1) {
            playerLoses();
        }
        else {
            runDraw();
        }
    }
    else {//d2 is not bust, therefore will be a better hand than d1
        if (p1 > d2) {
            playerWins();
        }
        else if (d2 > p1){
            playerWins();
        }
```

```
            else {
                runDraw();
            }
        }
    }
    if (p1 != p2 && d1 != d2) {//both player and dealer have aces
        if (p2 >21 && d2 > 21) {//Both players 11 value hand would be bust
            if (p1 > d1) {
                playerWins();
            }
            else if (p1 < d1) {
                playerLoses();
            }
            else {
                runDraw();
            }
        }
        else if (p2 > 21){//p2 is bust, d2 not
            if (p1 > d2){//d2 not bust so will be a better hand than d1
                playerWins();
            }
            else if (p1 < d2){
                playerLoses();
            }
            else {
                runDraw();
            }
        }
        else if (d2 > 21){//d2 bust, p2 not
            if (p2 > d1) {
                playerWins();
```

```
            }
            else if (p2 < d1){
               playerLoses();
            }
            else {
               runDraw();
            }
         }
         else {//Neither player bust, therefore p2 > p1 && d2 > d1
            if (p2 > d2) {
               playerWins();
            }
            else if (p2 < d2){
               playerLoses();
            }
            else {
               runDraw();
            }
         }
      }
   }
}

void standPlayer(){
   //So a player is standing. Dealer plays now.
   playerHandObj.calculateHandValue();
   int playerFirstValue = playerHandObj.getFirstValue();
   int playerSecondValue = playerHandObj.getSecondValue();
   if (playerFirstValue > 21 && playerSecondValue > 21) {
      playerLoses();
   }
```

```
    else {
        //First, the dealer flips his covered card
        revealSecondCard();
        bool dealerIsPlaying = true;
        while (dealerIsPlaying) {
            dealerIsPlaying = dealerPlay();//return true on hitDealer
        }//only breaks when the dealer is standing
        dealerHandObj.calculateHandValue();
        int dealerFirstValue = dealerHandObj.getFirstValue();
        int dealerSecondValue = dealerHandObj.getSecondValue();
        //following if statements determine winner
        determineWinner(playerFirstValue,playerSecondValue,dealerFirstValue,dealerSecondValue);
            //first value has ace as 1, second as 11
    }
}

int makeChoice(){
    int choice;
    cout << "Type '1' to hit or '2' to stand"<<endl;
    try {
        cin >> choice;
    }catch(exception ex){
        cout << "Whoa! That's not a number!"<<endl;
        makeChoice();
    }
    switch (choice){
        case 1:
        case 2:
            return choice; break;
        default: return 0;break;
    }
```

```
    return 0;
}

void runChoices(){
    handInPlay = true;
    int choice;
    do{
        choice = makeChoice();
        if (choice == 0) {
            choice = makeChoice();
        }
        switch (choice){
            case 1: hitPlayer();break;
            case 2: standPlayer();break;
            default: exitApplication();break;
        }
    }while (choice == 1 && handInPlay);//Repeat this until the player stands
    if (choice == 1 && !handInPlay) {
        standPlayer();//if the player has hit and the hand is no longer in play, they did not stand but
have reached 21
    }
}

bool runNaturals(){
    bool playerNatural = playerHandObj.checkForNaturals();
    bool dealerNatural = dealerHandObj.checkForNaturals();
    if (dealerNatural) {
        if (playerNatural) {
            cout << "But so do you!"<<endl;
            runDraw();
        }
```

```
        else {
            playerLoses();
        }
    }
    else if (playerNatural) {
        cout << "You have blackjack!"<<endl;
        playerWins();
    }
            if (playerNatural || dealerNatural){
                    return true;
            }
            return false;
}

void initializeGame(){
    // start by asking to press 1 if they want to play or press 2 to access settings
    // Play -> ask them how much money they have to play with
    // Take that value (Use as pointer/reference passes)
    // Create a deck of cards
    // Deck of cards - 13 values * 4 suits (Settings can multiply this again by no. of decks)
    // Get bet value of player for how much they wager on this round
    // Shuffle deck
    // One card given to player(s), then one to dealer (shown)
    // Second card given to player(s) then second to dealer (hidden)
    // Upon each deal, remove that card from the deck
    setRoundBet();
    //Give a card to player then a card to dealer, then a second card to player then second to dealer. Hide the dealer's second card. Deal by popping off vector - dealing as if you're taking a card off the top/bottom of the deck. Hand class has been created for this
    setupGame();
    bool naturals = runNaturals();
    //if both have naturals, cout "but so do you!"
```

```
    if (!naturals)
                        runChoices();
}


void configureSettings(){
    int noOfDecks;
    cout << "\n\nHow many decks do you want to play with?\nEnter a value between 1 and 10" <<
endl;
    try {
        cin >> noOfDecks;
    } catch (exception) {
        cout << "Please enter a numeric value between 1 and 10" << endl;
        configureSettings();
    }
    if (!(noOfDecks >= 1 && noOfDecks <= 10)) {
        cout << "Please enter a numeric value between 1 and 10" << endl;
        configureSettings();
    }
    string filepath = "settings.txt";
    fstream myFile;
    myFile.open(filepath);
    if (myFile.is_open()) {
        myFile << noOfDecks;
    }
    else {
        cout << "an error has occurred. Please check that settings.txt is in the project folder" << endl;
    }
    myFile.close();
    mainMenu();
}
```

```
void runGame(){
   initializeGame();
   int choice;
   cout << "Type '1' to play another round!"<<endl;
   try {
      cin >> choice;
   }catch(exception ex){
      cout << "Whoa! That's not a number!"<<endl;
      exitApplication();
   }
   switch (choice) {
      case 1:
         runGame();//If they're playing again, run this so initializeGame runs then the prompts
         break;

      default: exitApplication();
         break;
   }
}

int main(int argc, const char * argv[]) {
   // insert code here...
   mainMenu();
}

void mainMenu(){
   int mode = displayLogo();
   switch (mode){
      case 1:
         createNewDeck();
         setBalance();
```

```
            runGame();break;
        case 2:
            configureSettings();break;
        default:
            exitApplication();break;
    }
}
//Clear hands after rounds.
//Prompt for playing another round, then runGame() upon end of last round
```

```
//
//  Hand.hpp
//  OOP CW
//
//  Created by Michael Dunwoody on 05/11/2019.
//  Copyright © 2019 Michael Dunwoody. All rights reserved.
//

#include <stdio.h>
#include <string>
#include <vector>
#include "Card.hpp"

class Hand{

public:
    Hand();
    Hand(bool _player);
    bool isPlayer();
    void clear();
    void dealCard(Card cardBeingDealt);
    void displayHand(bool player, bool hide);
    void calculateHandValue();
    bool checkBust();
    bool checkForNaturals();
    void displayTopCard(bool player);//Grabs the card from top of stack - the last one
which was dealt
    void clearCards();
    int getFirstValue();
    int getSecondValue();
private:
    std::vector<Card> cards;
    bool player;
    int value[2];
};
//
//  Hand.cpp
//  OOP CW
//
//  Created by Michael Dunwoody on 05/11/2019.
//  Copyright © 2019 Michael Dunwoody. All rights reserved.
//
#include <stdio.h>
#include <string>
#include <vector>
#include <iostream>

#include "Card.hpp"
#include "Hand.hpp"

using namespace std;

Hand::Hand(){};
Hand::Hand(bool _player)
    :player(_player){};

bool Hand::isPlayer(){
    return player;
}

int Hand::getFirstValue(){
    return value[0];
```

```
}

int Hand::getSecondValue(){
    return value[1];
}

void Hand::displayTopCard(bool player){
    if (player){
        cout << "You were dealt the " << cards.back().getValueString() << " of " <<
cards.back().getSuit() << endl;
    }
    else {
        cout << "The dealer was dealt the " << cards.back().getValueString() << " of "
<< cards.back().getSuit() << endl;
    }
}

void Hand::dealCard(Card cardBeingDealt){
    cards.push_back(cardBeingDealt);
}
void Hand::calculateHandValue(){
    int total1 = 0;
    int total2 = 0;
    for (size_t i = 0; i < cards.size(); i++) {
        total1 += cards[i].getValue();
        if (cards[i].getValueString() == "ACE") {
            total2 += cards[i].getSecondValue();
        }
        else total2 += cards[i].getValue();
    }
    *value = total1;
    *(value + 1) = total2;
}

void Hand::clearCards(){
    cards.clear();
    cards.clear();
}

bool Hand::checkBust(){
    bool isBust = false;
    if (*value > 21 && *(value + 1) > 21) {
        isBust = true;
    }
    return isBust;
}

bool Hand::checkForNaturals(){
    calculateHandValue();
    if (getFirstValue() == 21 || getSecondValue() == 21) {
        return true;
    }
    else return false;
}

void Hand::displayHand(bool isPlayer, bool hideTopCard){

    calculateHandValue();//Calculates the hand's value for the player or dealer hand

    if (isPlayer) {//If the hand belongs to the player...
        cout << endl << "Your hand:"<<endl;
        for (size_t i = 0; i < cards.size(); i++) {
```

```
            cout << cards[i].getValueString() << " of " <<cards[i].getSuit()<<endl;
        }
        if (*value == *(value + 1)) {
            cout << "Total hand value:\t" << *value << endl;
        }
        else{
            cout << "Total hand value:\t" << *value << " or " << *(value+1) << endl;
        }
    }
    else{//If the hand belongs to the dealer
        if (cards.size() == 2) {
            if (hideTopCard) {
                //The dealer's second card is always face down at first
                cout << endl << "Dealer hand:\n"<<cards[0].getValueString() << " of
"<<cards[0].getSuit()<<endl<<"The dealer's second card is face-
down"<<endl;//dealerHand[1] is hidden
            }
            else {
                cout << endl << "Dealer hand:"<<endl;
                for (size_t i = 0; i < cards.size(); i++) {
                    cout << cards[i].getValueString() << " of "
<<cards[i].getSuit()<<endl;
                }
                if (*value == *(value + 1)) {
                    cout << "Total hand value:\t" << *value << endl;
                }
                else{
                    cout << "Total hand value:\t" << *value << " or " << *(value+1) <<
endl;
                }
            }

            // Check for naturals - 10 card + ace.
            // If shown dealer card is a ten card or ace, they check their other card
to see if they have a natural.
            // If so, collect all bets of players that do not have naturals.
            // If player has it and dealer does not, player receives 1.5x bet
            // If both have it, player gets their bet back but no winnings.
            // If neither have it, begin play
            if ((cards[0].getValue() >= 10 || cards[0].getSecondValue() >= 10) &&
hideTopCard) {//if first card is ace or 10
                cout << "The dealer is checking for blackjack..." <<endl;
                if (!(*value == 21 || *(value+1) == 21)) {
                    cout << "The dealer does not have blackjack."<<endl;
                }
                else {
                    cout << "The dealer has blackjack!"<<endl;//After here, main class
continues
                }
            }
        }
        else {
            //In here goes any logic for displaying the dealer hand if they have 'hit'
            cout << endl << "Dealer hand:"<<endl;
            for (size_t i = 0; i < cards.size(); i++) {
                cout << cards[i].getValueString() << " of "
<<cards[i].getSuit()<<endl;
            }
            if (*value == *(value + 1)) {
                cout << "Total hand value:\t" << *value << endl;
            }
            else{
```

```
                cout << "Total hand value:\t" << *value << " or " << *(value+1) <<
endl;
            }
        }
    }
}

void Hand::clear(){
    cards.~vector();
}
```

```
//
//  Deck.hpp
//  OOP CW
//
//  Created by Michael Dunwoody on 27/10/2019.
//  Copyright © 2019 Michael Dunwoody. All rights reserved.
//

#pragma once

#include <stdio.h>
#include <string>
#include <random>
#include "Card.hpp"

class Deck{
public:
    Deck();
    Deck(int mode);
    static std::vector<Card> setupDeck();
    static std::default_random_engine getRandomEngine();
private:
    static std::default_random_engine engine;
};
//
//  Deck.cpp
//  OOP CW
//
//  Created by Michael Dunwoody on 27/10/2019.
//  Copyright © 2019 Michael Dunwoody. All rights reserved.
//

#include "Deck.hpp"
#include "Card.hpp"

#include <iterator>
#include <vector>
#include <map>
#include <algorithm>
#include <array>
#include <random>
#include <fstream>

using namespace std;
typedef enum {CLUBS, DIAMONDS, HEARTS, SPADES} SUITES;
typedef enum {ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN,
KING} NUMBERS;

Deck::Deck(){}
Deck::Deck(int mode){}
default_random_engine Deck::engine(std::random_device{}());

std::default_random_engine Deck::getRandomEngine(){
    return engine;
}
vector<Card> Deck::setupDeck(){
    //for each SUITE, create an ace and then a card of each NUMBERS value
    vector<Card> deck;
    string filepath = "settings.txt";
    //If they're using >1 deck, append playingDeck by the amount of decks minus 1
    fstream myFile;
    myFile.open(filepath);
```

```
    int numberOfDecks = 1;
    int line;
    if (myFile.is_open())
    {
        while (myFile >> line) {
            numberOfDecks = line;
        }
    }
    else numberOfDecks = 1;
    myFile.close();

    map<string, int> numberValues;
    numberValues["TWO"] = 2;
    numberValues["THREE"] = 3;
    numberValues["FOUR"] = 4;
    numberValues["FIVE"] = 5;
    numberValues["SIX"] = 6;
    numberValues["SEVEN"] = 7;
    numberValues["EIGHT"] = 8;
    numberValues["NINE"] = 9;
    numberValues["TEN"] = 10;
    numberValues["JACK"] = 10;
    numberValues["QUEEN"] = 10;
    numberValues["KING"] = 10;

    string suites[4] = {"SPADES","CLUBS", "DIAMONDS", "HEARTS"};

    for (int i = 0; i < numberOfDecks; i++) {
        for (string suite : suites){
            Card temp = Card("ACE", suite, 1, 11);
            deck.push_back(temp);//adds ace to deck
            temp.~Card();
            for (map<string, int>::iterator it = numberValues.begin(); it !=
numberValues.end(); it++) {
                Card temp = Card(it->first, suite, it->second);
                deck.push_back(temp);
                temp.~Card();
            }
        }
    }
    std::shuffle(begin(deck), end(deck), getRandomEngine());
    return deck;
}
```

```
//
//  Card.hpp
//  OOP CW
//
//  Created by Michael Dunwoody on 27/10/2019.
//  Copyright © 2019 Michael Dunwoody. All rights reserved.
//
#pragma once

#include <stdio.h>
#include <string>

class Card{
public:
    Card();
    Card(int value);
    Card(std::string suit);
    Card(int value, std::string suit);
    Card(std::string cardNumber, std::string suit, int value);
    Card(std::string cardNumber, std::string suit, int value, int secondValueForAces);
    int getValue();
    int getSecondValue();
    std::string getSuit();
    std::string getValueString();
private:
    int value;
    std::string suit;
    std::string cardNumber;
    int secondValue;
};
//
//  Card.cpp
//  OOP CW
//
//  Created by Michael Dunwoody on 27/10/2019.
//  Copyright © 2019 Michael Dunwoody. All rights reserved.
//

#include "Card.hpp"
#include <iostream>
using namespace std;

Card::Card():value(0),secondValue(0){}
Card::Card(int value)
    :value(value),secondValue(0){}
Card::Card(string suit)
    :suit(suit),value(0),secondValue(0){}
Card::Card(int value, string suit)
    :value(value),suit(suit),secondValue(0){}
Card::Card(string cardNumber, string suit, int value)
    :cardNumber(cardNumber),suit(suit),value(value),secondValue(0){}
Card::Card(string cardNumber, string suit, int value, int secondValue)
    :cardNumber(cardNumber),suit(suit),value(value),secondValue(secondValue){}

int Card::getValue(){
    return value;
}

int Card::getSecondValue(){
    return secondValue;
}
string Card::getSuit(){
```

```
    return suit;
}
string Card::getValueString(){
    return cardNumber;
}
```