California State University, Long Beach

Blackjack Project Report

Tiffany Lam (015181853), Chelsea Marfil (014400501), Rachel Pai (015555603)

CECS 282, Mon/Wed, 12 PM - 12:50 PM

Professor Minhthong Nguyen

December 12, 2018, 11:59 PM

Introduction:

The purpose of this program is to simulate a Blackjack Slot Machine in C++.

Program Analysis and Algorithm Design:

a. Describe any variables used in the program.

i. Account.h & Account.cpp
   1. **Int mAccountNumber:** the account number (for the player)
   2. **Double mMoney:** the amount of money inside the account.

ii. Card.h & Card.cpp
   1. **String face:** the number value of the card, written in English. (Ex. value of 1 = "one")
   2. **String suit:** the suit of the card (Ie, heart, spade, diamond, clubs)

iii. DeckOfCards.h & DeckOfCards.cpp
   1. **Card* deckofCards:** A pointer to the Card type, representing the deck of cards.
   2. **Int currentCard:** Keeps track of the current card we are on within the deck of cards.

iv. Game.h
   1. **Player mPlayer:** The player that is playing the game.
   2. **Player mDealer:** The dealer playing the game with the player.
   3. **Int playerScore:** Keeps track of the score / total card values for the player.
   4. **Int dealerScore:** Keeps track of the score / total card values for the dealer.
   5. **DeckOfCards mDeck:** The deck of cards for the game.

v. Game.cpp
   1. (Within promptUserForAccountNumber())
      Int acctNum: holds the value of the account number that the user wants to play with.
   2. (Within promptUserForAmountToBet())
      Int betAmt: holds the value of what the user wants to bet from their account.
   3. (Within selectAndShowTwo())
      Card firstCard: the first card that gets dealt to the user's hand.
      Card secondCard: the second card that gets dealt to the user's hand.
   4. (Within selectAndShowOne)
      Card firstCard: the card that gets dealt to the user's hand.

5. (Within stand())
   Int decision: holds the user input when they enter a number to select a card for the dealer.

vi. Player.h & Player.cpp
1. **Account mAccount:** The player's account.
2. **vector<Card> playersHand**: The player's hand, as represented by a vector of cards.
3. **vector<Card> dealersHand**: The dealer's hand as represented by a vector of cards.

vii. Main.cpp
1. **Player p1:** the player of the game.
2. **Double betTracker:** Holds the amount of money that the user bet.
3. **Double moneyWon:** Holds the amount of money that the user won, based on what they bet.
4. **Bool gameEnd**: the boolean value that tells the while loop whether or not it should exit - signifies if the game should continue or end.
5. **DeckOfCards *deck:** The new deck of cards for the game.
6. **bool validAccountNumber:** signifies whether the account number has been created, or has been stored as a valid account already.
7. **int acctNum:** the account number for the player.
8. **bool validBetAmount:** signifies whether the user is allowed to bet the amount of money that they wish to bet. For example, if they were to choose to bet more money than they have in their account, this value would be false.
9. **double betAmount:** the amount of money that the user wants to bet.
10. **double userMoneyAmount:** Holds the money value of the user's account.
11. **vector<Card> playersHand:** What the user has in their hand currently.
12. v**ector<Card> playersCard:** What the user is being shown when two cards are selected and shown to them.
13. **vector<Card> dealersHand:**What the dealer has in their hand.
14. **vector<Card> dealersCards:** What the dealer is being shown when two cards are selected and shown to them.
15. **Bool userStands:** Indicates whether the user decides to stand.
16. **Bool firstPrint:** Indicates if the stand statement printed already. The user will only be prompted for a decision to stand the first time they play the game.
17. **String decision:** Takes in the decision that the user wants, whether they want "stand", "hit", or "split".

18. **int currentMoney:** the current money that the user has in their account. Also used later on to calculate the amount of money they will have left in their account after winning or losing bets.
19. **string userInput:** Takes in the user input for whether they want to play the game again or not.

b. Describe any functions used in the program.

i. Account

1. **Account():** Constructor for a default account.
2. **Account (int accountNum):** Constructor for an account where the account number is assigned via parameter.
3. **Account(int accountNum, double money):** Constructor for an account where the account number and money inside is assigned via parameter.
4. **Int getAccountNumber() const:** Returns the account number of the account.
5. **Double getMoney() const:** Returns the money inside the account.
6. **Void setMoney(double amount):** Allows the parameter's amount to become what money amount inside the account. Allows for change of the amount of money inside the account.

ii. Card

1. **Card():** Default constructor for a card.
2. **Card(string face, string suit):** Constructor that takes in a face (value of a card as displayed by a string. Ie. 1 = "one"), and a suit (ie, heart, spade, club, or diamond).
3. **String print() const:** a toString for a card. Returns the format of "[face] of [suit]" when called.
4. **Int getFaceValue():** Converts the value of the suit of the card into an integer. For example, "one" = 1. Returns the integer value.
5. **String getFace() const:** Returns the face value of the card as a string.
6. **String getSuit() const:** Returns the suit value of the card as a string.

iii. DeckOfCards

1. **DeckOfCards():** Constructor for DeckOfCards. Contains an array of faces and suits with their string equivalents. The currentCard represents the current position within the deck of cards at the moment. (= 0). "Assembles" the deck of cards by pairing a face with a suit with a for loop.
2. **Void shuffle():** Shuffles a deck of cards using srand. With srand, we can generate random values to mix around the deck of cards within each other by swapping positions within the deck.

3. **Card dealCard():** Checks the current position of the deck of cards, through currentCard. If the position is greater than the amount of cards in the deck, then no card will be dealt. If the position is less than the amount of cards in the deck, the card after the currentCard will be dealt.
4. **Void printDeck() const:** Prints out the formatted deck of cards.

iv. Game

1. **Game():** Constructor contains a deck of cards.
2. **Game(DeckOfCards &deck):** Constructor that takes in a deck of cards, and the playerScore is set to 0.
3. **Game(DeckOfCards &deck, Player player, Player dealer):** Constructor that takes in a deck of cards, a player, and a dealer. Both player and dealer scores are set to 0.
4. **Void setPlayer (Player player):** Takes in a player to be a player of the game.
5. **Void setDealer(Player dealer):** Takes in a dealer to be a player of the game.
6. **Player getPlayer() const:** returns the player of the game.
7. **Player getDealer() const:** returns the dealer of the game.
8. **Int promptUserForAccountNumber():** Asks the user for their account number.
9. **Double promptUserForAmountToBet():** Asks the user for the amount they want to bet.
10. **vector<Card> selectAndShowTwo(DeckOfCards &mDeck, vector<Card>, &hand):** Randomly selects and shows the user two cards. Only happens once at the beginning of the game. A deck of cards (the one that's been used in the game) and the players' hand is passed in. Returns the hand of the player as a vector of cards.
11. **Card selectAndShowOne(DeckOfCards &mDeck, vector<Card>, &hand):** Randomly selects and shows the user one card. A deck of cards (the one that's been used in the game) and the players' hand is passed in. Returns the card as a Card type.
12. **Void showHand(vector<Card> &hand):** Acts as a toString for the hand that's passed in. Displays what the user has in their hand.
13. **String askHitStandOrSplit(DeckOfCards &mDeck, vector<Card> &hand, vector<Card> &dealerHand):** Prompts the user if they want to stand, hit, or split. It will call the correct functions that the user wants to select.
14. **Void hit(DeckOfCards &mDeck, vector<Card> &hand, vector<Card> &dealerHand):** The dealer will receive one new card.

Stand will be called if the dealer's score is greater than or equal to 17. If the dealer's score is less than 17, then the dealer will get another card.

15. **Void stand((DeckOfCards &mDeck, vector<Card> &hand, vector<Card> &dealerHand):** If the dealer's score is greater than or equal to 17, then don't do anything. If the dealer's score is less than 17, while it's less than 17, continue drawing cards until it reaches 17. If the player's score is less than the dealer's score, then the player wins. If not, the dealer wins.
16. **Void split(DeckOfCards &mDeck, vector<Card> &hand, vector<Card> &dealerHand, Player p1):** Allows the user to draw another two cards. The user will now have two hands to play with.
17. **Int calcValueOfHand(vector<Card> &hand):** Calculates the current numerical value of the hand that the user has currently. If within the pair, there is an ace and the current score is less than or equal to ten, then the ace is worth 11 points. If the current score is more than ten, then the ace would count as 1 point.

v. Main

1. **Void updateAccount(int accountNumber, double amountToAdd):** If the account number passed into the function matches what's inside the accounts vector, then set the amount of money to what's passed in.
2. **Bool accountExists(int accountNumber):** Checks if the passed in account number is a valid account, checking if it's in the account vector. Returns true or false depending on if it's in the vector or not.
3. **Double getMoneyAmount(int accountNumber):** Returns the amount of money as a double inside the account number that's passed in.
4. **Void printAccounts():** Acts as a toString to the account, printing it out one by one.
5. **Void userWins(Game g, vector<Card> &playersHand, vector<Card> &dealersHand, double userMoneyAmount, double betAmount, int acctNum, int moneyWon):** Displays that the user has won, along with the corresponding scores for both the user and the dealer. The money bet by the user is doubled. The account is also updated with the bet money in addition to what was already existing. The user's balance will also display.
6. **Void userTies(Game g, vector<Card> &playersHand, vector<Card> &dealersHand, double userMoneyAmount, double betAmount, int acctNum, int moneyWon):** Displays that the user has tied with the dealer, along with the corresponding scores for both the user and the dealer. The money bet by the user is cut in half. The account is also updated with the

bet money in addition to what was already existing. The user's balance will also display.

7. **Void dealerWinsGame g, vector<Card> &playersHand, vector<Card> &dealersHand, double userMoneyAmount, double betAmount, int acctNum, int moneyWon):** Displays that the dealer has won, along with the corresponding scores for both the user and the dealer. The money bet by the user is lost. The user's balance will also display, subtracting what bet was given at the beginning of the game.
8. **void askPlayAgain(bool &gameEnd, double betTracker, double moneyWon):** The user will be prompted if they would like to play again. The money won from earlier will be transferred into their account to be able to bet again.

c. Describe algorithm in the main function.

i. The account is first created, along with the user's account number to have a starting balance of $100. The user is assigned as player 1, as declared with the account information. The account is pushed onto the vector that contains all the accounts, and is then displayed.
ii. The betTracker starts at 0, along with the moneyWon. The game has a boolean variable that determines whether the game will continue to run. While the gameEnd is false, continue on.
iii. A new deck of cards and game is created. validAccountNumber checks if the account entered by the user is within the accountsVector. An if-else statement checks and lets the user know whether or not the account number they inputted is valid.
iv. The account number is then assigned to be a player. Ther user will then be asked to enter the amount of money they want to bet. There is a variable to check whether the bet amount is valid according to the amount of money that is already inside the account. If the user bets more than what they have, they must re-enter a new amount. A verification statement is printed out to indicate to the user how much they are betting.
v. The deck of cards is shuffled, and printed out to allow the user to see the order of the cards. (They will not be choosing from it.)
vi. The player's hand is created and they will receive two random cards from the deck. The cards added to the hand are then displayed to the user.
vii. The dealer's hand is created and they will also receive two random cards from the deck. One of the cards is displayed to the user.
viii. The value of the player's hand is calculated and displayed for them.
ix. While the user has not reached a hand value of 21 or more, they can decide if they want to hit, stand, or split.

1. If the user decides to stand, the dealer's cards and its values are displayed to the user. While the dealer's hand contains a value of less than 17, they will continue to draw one card at a time. These are displayed to the user as it happens.
2. If the user decides to hit, the player will get to draw one more card. It is displayed to them.
3. If the user decides to split, the game will create two new hands for the player. The player will then be allowed to hit or stand with either of the 2 decks.

x. To determine the winner of the game, it is based on the calculated value of the hand of both the player and the dealer.
   1. If the player's hand is greater than 21, then the dealer wins. The player will win none of the money that they bet.
   2. If the dealer's hand is greater than 21, then the dealer wins. The player will win double their bet.
   3. If the player and dealer's hand tie, then the player will win half of the betted money.
   4. If the dealer wins, the user will win no money.

xi. Once a win/lose is determined, the user can decide to play again. If at the prompt, they enter "no", then they will exit the game and gameEnd is true. The scores will display for both the user and the dealer.

Program Code:

main.cpp

```
#include <iostream>
#include "DeckOfCards.h"
#include "Game.h"
using namespace std;

static vector<Account> accountsVector;

/**
If the account number passed into the function matches what's inside the accounts vector, then set the amount of money to what's passed in.
@param: int accountNumber
@param: double amountToAdd
*/
void updateAccount(int accountNumber, double amountToAdd) {
   for(int i = 0; i < accountsVector.size(); i++) {
```

```
        if (accountsVector[i].getAccountNumber() == accountNumber) {
            accountsVector[i].setMoney(amountToAdd);
        }
    }
}

/**
Checks if the passed in account number is a valid account, checking if it's in the account vector.
Returns true or false depending on if it's in the vector or not.
@param: int accountNumber
@return: bool
*/
bool accountExists(int accountNumber) {
    for(int i = 0; i < accountsVector.size(); i++) {
        if (accountsVector[i].getAccountNumber() == accountNumber) {
            return 1;
        }
    }
    return 0;
}

/**
Returns the amount of money as a double  inside the account number that's passed in.
@param: int accountNumber
@return: double (money amount)
*/
double getMoneyAmount(int accountNumber) {
    for(int i = 0; i < accountsVector.size(); i++) {
        if (accountsVector[i].getAccountNumber() == accountNumber) {
            return accountsVector[i].getMoney();
        }
    }

    return -1;
}

/**
Acts as a toString to the account, printing it out one by one.
*/
```

```
void printAccounts() {
   for(int i = 0; i < accountsVector.size(); i++) {
      cout << accountsVector[i].getAccountNumber() << ": " << accountsVector[i].getMoney()
<< endl;
   }
}

/**
Displays that the user has won, along with the corresponding scores for both the user and the
dealer. The money bet by the user is doubled. The account is also updated with the bet money in
addition to what was already existing. The user's balance will also display.
@param: Game g
@param: vector<Card> &playersHand
@param: vector<Card> &dealersHand
@param: double userMoneyAmount
@param: double betAmount
@param: int acctNum
@param: int moneyWon
*/
void userWins(Game g, vector<Card> &playersHand, vector<Card> &dealersHand, double
userMoneyAmount, double betAmount, int acctNum, double &moneyWon) {
   cout<<"-----------------------------------------------"<<endl;
         cout <<"You won! "<< endl;

   cout << "The dealer's score: " << g.calcValueOfHand(dealersHand) << endl;
   cout << "Your score: " << g.calcValueOfHand(playersHand) << endl;
   //you will be taking their money, but giving it back to them + the amount they bet
   double currMoney = userMoneyAmount  + betAmount;

    moneyWon += (betAmount); //give their money back and the amount they bet


   updateAccount(acctNum, currMoney);
   cout << "Your balance is now: " << getMoneyAmount(acctNum) << endl << endl;
}

/**
```

Displays that the user has tied with the dealer, along with the corresponding scores for both the user and the dealer. The money bet by the user is cut in half. The account is also updated with the bet money in addition to what was already existing. The user's balance will also display.
@param: Game g
@param: vector<Card> &playersHand
@param: vector<Card> &dealersHand
@param: double userMoneyAmount
@param: double betAmount
@param: int acctNum
@param: int moneyWon
*/
void userTies(Game g, vector<Card> &playersHand, vector<Card> &dealersHand, double userMoneyAmount, double betAmount, int acctNum, double &moneyWon) {
```
   cout<<"-----------------------------------------------"<<endl;
        cout<<"You've tied with the dealer. You will win half of what you've bet." <<endl;
   cout << "The dealer's score: " << g.calcValueOfHand(dealersHand) << endl;
   cout << "Your score: " << g.calcValueOfHand(playersHand) << endl;

   double currMoney = userMoneyAmount + (betAmount / 2);
   moneyWon += betAmount / 2;

   updateAccount(acctNum, currMoney);
   //If the player ties with the dealer, then their betted money will be cut in half.
   cout << "Your balance is now: " << getMoneyAmount(acctNum) << endl << endl;
}
```

/**
Displays that the dealer has won, along with the corresponding scores for both the user and the dealer. The money bet by the user is lost. The user's balance will also display, subtracting what bet was given at the beginning of the game.
@param: Game g
@param: vector<Card> &playersHand
@param: vector<Card> &dealersHand
@param: double userMoneyAmount
@param: double betAmount
@param: int acctNum
@param: int moneyWon
*/

```
void dealerWins(Game g, vector<Card> &playersHand, vector<Card> &dealersHand, double
userMoneyAmount, double betAmount, int acctNum) {
   cout<<"-----------------------------------------------"<<endl;
         cout << "Dealer won." << endl;
   cout << "The dealer's score: " << g.calcValueOfHand(dealersHand) << endl;
   cout << "Your score: " << g.calcValueOfHand(playersHand) << endl;
   double currMoney = userMoneyAmount - betAmount;
   updateAccount(acctNum, currMoney);
   cout << "Your balance is now: " << getMoneyAmount(acctNum) << endl << endl;
}

/**
The user will be prompted if they would like to play again. The money won from earlier will be
transferred into their account to be able to bet again.
@param: bool &gameEnd
@param: double betTracker
@param: double moneyWon
*/
void askPlayAgain(bool &gameEnd, double betTracker, double &moneyWon) {
   cout << "Do you want to play again?" << endl;
   string userInput;
   cin >> userInput;
   if(userInput == "yes" || userInput == "Yes"){

      gameEnd = false;
   } else if (userInput == "no" || userInput == "No") {
      //TODO: not sure how exactly he wants us to calculate moneyWon...
      cout << "Total amount of betting money: " << betTracker << endl;
      cout << "Total amount of money won: " << moneyWon << endl;
      gameEnd = true;
   }
}



int main(int argc, const char * argv[]) {
   Account a(1001, 100);
   Player p1 = Player(a);
   accountsVector.push_back(a);
```

```
    printAccounts();

    double betTracker = 0;
    double moneyWon = 0;

    bool gameEnd = false;
    while(gameEnd == false){
            DeckOfCards *deck = new DeckOfCards();
            Game g(*deck);

            // a. Ask the user to enter the account number, make sure it exists.
            bool validAccountNumber = 0;
            int acctNum;
            do {
               acctNum = g.promptUserForAccountNumber();
               if (!accountExists(acctNum)) {
                  cout << "Account doesn't exist." << endl;
               } else {
                  cout << "Account exists!" << endl;
                  validAccountNumber = 1;
               }
            } while(!validAccountNumber);

        // Set the player of the game.
            g.setPlayer(Player(acctNum));

            // b. Ask the user to enter the amount of money he/she wants to bet, make sure it's a
valid amount.
            bool validBetAmount = 0;
        double betAmount;
            double userMoneyAmount = getMoneyAmount(acctNum);
            cout << "You have $" << userMoneyAmount << " available to bet." << endl;
            do {
               betAmount = g.promptUserForAmountToBet();
               if (betAmount <= userMoneyAmount) {
                  cout << "You are betting $" << betAmount << "." << endl;
                  cout<<"--------------------------------------------"<<endl;
            betTracker += betAmount;
                  validBetAmount = 1;
```

```
            } else {
                cout << "Can't bet more money than you have." << endl;
            }
        } while(!validBetAmount);

    // Shuffle the card deck.
        deck->shuffle();


        // c. The program will select and show the user two cards from the shuffled deck.
            vector<Card> playersHand = g.getPlayer().getHand();
            playersHand = g.selectAndShowTwo(*deck, playersHand);
            cout << "Your cards are: " << endl;
            for(int i = 0; i < playersHand.size(); i++){
                    cout << playersHand[i].print() << endl;
            }

        // d. The program will select another two cards and show one card (dealer's cards).
    vector<Card> dealersHand = g.getDealer().getHand();
    dealersHand = g.selectAndShowTwo(*deck,dealersHand);
    cout << endl << "One of the dealer's cards is: " << endl;
    cout << dealersHand[0].print();
    cout << endl << endl;

        // e. The program will show the value of user's cards

        cout << "The value of your hand is: " << g.calcValueOfHand(playersHand)<<endl;


    bool userStands = false;
    bool firstPrint = true;
    while (!userStands && g.calcValueOfHand(playersHand) < 21 &&
g.calcValueOfHand(dealersHand) < 21) {
        // ask if the user wants to hit or stand or split.
        string user;
        if(firstPrint == true){
                cout<<"Do you want to hit, stand, or split? "<<endl;
                cin>>user;
                        }
```

```
                else{
                        user = g.askHitOrStand(*deck,playersHand,dealersHand,p1);
                }


        // f. If the user decides to stand.
                if (user == "stand") {
                        g.stand(*deck, playersHand, dealersHand, p1);
           userStands = true;
        // If user decides to hit.
        }
                else if (user == "hit") {
                g.hit(*deck, playersHand);
                firstPrint = false;
        }

        // h. TODO: If the user decides to split, the dealer will draw two cards for the user. The
user now has two hands.
                // Also, an additional bet of equal value to the original bet is placed on the
second hand. Proceed the game as in step f and/or g.
        else if (user == "split"){
                //betAmount = betAmount * 2;
                cout<<"-----------------------------------------------"<<endl;
                        cout << "You have decided to split your deck." << endl;
                        cout<<"-----------------------------------------------"<<endl;
                //g.split(g, *deck, playersHand);
                //Seperate the hand into two vectors
                vector<Card> hand1;
                vector<Card> hand2;
                        hand1.push_back(playersHand[0]);
                        hand2.push_back(playersHand[1]);

                        //we want to return the two decks so the player can play those
                        //cout << "The dealer has drawn two cards for you: " <<
selectAndShowTwo(mDeck, hand) << endl;
                        g.split(*deck,hand1,hand2,p1);
                        //the dealer gives each deck another card
                        cout<<"-----------------------------------------------"<<endl;
                        cout<<"The dealer will now deal a card for each hand"<<endl;
```

```
                cout<<"-----------------------------------------------"<<endl;
                hand1.push_back(deck->dealCard());
                hand2.push_back(deck->dealCard());
if(g.calcValueOfHand(hand1) == 21){
        playersHand = hand1;
        g.split(*deck,hand1,hand2,p1);//displays the two hands
        break;
                }
                if(g.calcValueOfHand(hand2) == 21){
                        playersHand = hand2;
                        g.split(*deck,hand1,hand2,p1);//displays the two hands
        break;
                }
                g.split(*deck,hand1,hand2,p1);
string decision;
decision = g.askHitOrStand(*deck, playersHand, dealersHand, p1);
if (decision == "hit"){


        cout<<"What hand would you like to put it in? (1 or 2)? ";
        int userHandChoice;
        cin>>userHandChoice;

        if(userHandChoice == 1){
                g.hit(*deck, hand1);
                playersHand = hand1;

                firstPrint = false;

                        }
                        else if(userHandChoice == 2){
                                g.hit(*deck, hand2);
                                playersHand = hand2;

                                firstPrint = false;
                        }
                }

                else if(decision == "stand"){
```

```
                        cout<<"What hand would you like to stand? (1 or 2)? ";
                int userHandChoice;
                cin>>userHandChoice;
                if(userHandChoice == 1){
                        playersHand = hand1;
                        g.stand(*deck, hand1, dealersHand, p1);
                        firstPrint = false;
                        userStands = true;

                                }
                                else if(userHandChoice == 2){
                                        playersHand = hand2;
                                        g.stand(*deck, hand2, dealersHand, p1);

                                        firstPrint = false;
                                        userStands = true;



                                }
                                else{
                                        cout<<"error"<<endl;
                                        firstPrint = false;
                                }

                        } //end stand
                        else{
                                cout<<"error"<<endl;
                                firstPrint = false;
                        }


          } //end split


        }


// If value of players hand is > 21, dealer wins.
if(g.calcValueOfHand(playersHand) > 21) {
```

```
            dealerWins(g, playersHand, dealersHand, userMoneyAmount, betAmount, acctNum);

            askPlayAgain(gameEnd, betTracker, moneyWon);

        // If value of dealers hand is > 21, player wins.
        } else if(g.calcValueOfHand(dealersHand) > 21) {
            userWins(g, playersHand, dealersHand, userMoneyAmount, betAmount, acctNum,
moneyWon);

            askPlayAgain(gameEnd, betTracker, moneyWon);

        // If value of the dealers hand and players hand are equal, there is a tie.
        } else if(g.calcValueOfHand(playersHand) == g.calcValueOfHand(dealersHand)){
            userTies(g, playersHand, dealersHand, userMoneyAmount, betAmount, acctNum,
moneyWon);

            askPlayAgain(gameEnd, betTracker, moneyWon);

        // If the players hand is greater than the dealers hand and is <= 21, player wins.
        } else if (g.calcValueOfHand(playersHand) > g.calcValueOfHand(dealersHand) &&
g.calcValueOfHand(playersHand) <= 21)
        {
            userWins(g, playersHand, dealersHand, userMoneyAmount, betAmount, acctNum,
moneyWon);

            askPlayAgain(gameEnd, betTracker, moneyWon);

        // Else dealer wins.
        } else {
            dealerWins(g, playersHand, dealersHand, userMoneyAmount, betAmount, acctNum);

            askPlayAgain(gameEnd, betTracker, moneyWon);

        }
    }
    return 0;
}
```

Account.h

```
#ifndef Account_h
#define Account_h

class Account {

public:
   Account();
   Account(int accountNum);
   Account(int accountNum, double money);
   int getAccountNumber() const;
   double getMoney() const;
   void setMoney(double amount);
private:
   int mAccountNumber;
   double mMoney;

};
#endif /* Account_h */
```

Account.cpp

```
#include <stdio.h>
#include "Account.h"
#include <iostream>
using namespace std;

//Constructor for a default account.
Account::Account() {

}

/**
Constructor for an account where the account number is assigned via parameter.
@param: int accountNum
*/
Account::Account(int accountNum) {
   mAccountNumber = accountNum;
}
```

```
/**
Constructor for an account where the account number and money inside is assigned via
parameter.
@param: int accountNum
@param: double money
*/
Account::Account(int accountNum, double money) {
   mAccountNumber = accountNum;
   mMoney = money;
}

/**
Returns the account number of the account.
@return: int mAccountNumber
*/
int Account::getAccountNumber() const {
   return mAccountNumber;
}

/**
Returns the money inside the account.
@return: double mMoney
*/
double Account::getMoney() const {
   return mMoney;
}

/**
Allows the parameter's amount to become what money amount inside the account. Allows for
change of the amount of money inside the account.
*/
void Account::setMoney(double amount) {
   mMoney  = amount;
}
```

Card.h

```
#ifndef Card_h
```

```
#define Card_h
#include <string>

using namespace std;

class Card {
public:
   Card();
   Card(string face, string suit);
   string  print() const;
   int getFaceValue();
   string getFace() const;
   string getSuit() const;

private:
   string face;
   string suit;
};

#endif /* Card_h */
```

Card.cpp

```
#include "Card.h"

//default constructor
Card::Card() {

}
//Card constructor that takes in a face (string), and the suit (string) aka 10, hearts
Card::Card(string mFace, string mSuit) {
   face = mFace;
   suit = mSuit;
}
/**
* @return int - the integer value of the card
*/
int Card::getFaceValue() {
   if (face == "Ace") {
      return 1;
```

```
    } else if (face == "Two") {
        return 2;
    } else if (face == "Three") {
        return 3;
    } else if (face == "Four") {
        return 4;
    } else if (face == "Five") {
        return 5;
    } else if (face == "Six") {
        return 6;
    } else if (face == "Seven") {
        return 7;
    } else if (face == "Eight") {
        return 8;
    } else if (face == "Nine") {
        return 9;
    } else if (face == "Ten") {
        return 10;
    } else if (face == "Jack") {
        return 10;
    } else if (face == "King") {
        return 10;
    } else if (face == "Queen") {
        return 10;
    } else {
        return 0;
    }
}

/*
* print out the face of suit aka Nine of hearts
*@return string - a concatination of the face and suit
*/
string Card::print() const{
        return(face + " of " + suit);
}
/*
*@return string - the face aka Nine
*/
```

```
string Card::getFace() const {
	return  face;
}
/*
*@return string - the suit - SPADES, DIAMONDS, HEARTS, CLUBS
*/
string Card::getSuit() const{
	return suit;
}
```

DeckOfCards.h

```
#ifndef DeckOfCards_h
#define DeckOfCards_h
#include <vector>

#include "Card.h"

using namespace std;
#include <string>
#include <iostream>

const int NUMBER_OF_CARDS = 52;

class DeckOfCards {
public:
   DeckOfCards();
   void shuffle();
   Card dealCard();
   void printDeck() const;
private:
   //deckOfCArds is a pointer to a Card type
   Card* deckOfCards;
   int currentCard;

};

#endif /* DeckOfCards_h */
```

DeckOfCards.cpp

```
#include "DeckOfCards.h"
#include <ctime> //for time
#include <cstdlib> //for srand and rand
#include <iomanip>

/**
Constructor for DeckOfCards. Contains an array of faces and suits with their string equivalents.
The currentCard represents the current position within the deck of cards at the moment. (= 0).
“Assembles” the deck of cards by pairing a face with a suit with a for loop.
*/
DeckOfCards::DeckOfCards() {

   string faces[] = {"Ace", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine",
"Ten", "Jack", "King", "Queen"};

   string suits[] = {"Hearts", "Diamonds", "Clubs", "Spades"};

   currentCard = 0;

   deckOfCards = new Card[NUMBER_OF_CARDS];

   for(int i = 0; i < NUMBER_OF_CARDS; i++) {
      deckOfCards[i] = Card(faces[i % 13], suits[i/13]);
   }
}


/*
 Shuffles a deck of cards using srand
 */
void DeckOfCards::shuffle() {
   //srand for randomly generated time
   srand(time(0));
   //want to loop through the entire deck and randomize each card
   for(int i = 0; i < NUMBER_OF_CARDS; i++){
      int random = rand() % 52; //will give us a position from from 0 to 52
      //Swap method using a temporary variable
```

```
        Card temp = deckOfCards[i]; //set temp to a card in deck
        deckOfCards[i] = deckOfCards[random]; //now set that card to a random card in the deck
        deckOfCards[random] = temp; //finsh the swap by swapping the random card
    }

}
/*
* print out the deck
*/
void DeckOfCards::printDeck() const{
    cout << left;
    for (int i = 0; i < NUMBER_OF_CARDS; i++) {
        cout << setw(19) << deckOfCards[i].print();
        if ((i+1) % 4 == 0)
            cout << endl;
    }
}


/*
*Checks the current position of the deck of cards, through currentCard.
If the position is greater than the amount of cards in the deck, then no card will be dealt.
If the position is less than the amount of cards in the deck, the card after the currentCard will be
dealt.
*/
Card DeckOfCards:: dealCard() {
            //if currentCard - index that moves along the deck of cards
    if (currentCard > NUMBER_OF_CARDS) {
            cout<<"No more cards in deck."<<endl;
    }
    if (currentCard < NUMBER_OF_CARDS) {
            //update the deck of cards by moving index to next card
            return(deckOfCards[currentCard++]);
    }
    return deckOfCards[0];
}
```

Game.h

```
#ifndef Game_h
```

```
#define Game_h

#include "Card.h"
#include "Player.h"
#include "DeckOfCards.h"

#include <vector>

class Game {
public:
   Game(); //done

   Game(DeckOfCards &deck);
   Game(DeckOfCards &deck, Player player, Player dealer); //done
   void setPlayer(Player player); // done
   void setDealer(Player dealer); //done
   Player getPlayer() const; //done
   Player getDealer() const; //done
   void hit(DeckOfCards &deck, vector<Card> &playersHand);
   int promptUserForAccountNumber(); //done
   double promptUserForAmountToBet(); //done

   vector<Card> selectAndShowTwo(DeckOfCards &mDeck, vector<Card> &hand); //done
   Card selectAndShowOne(DeckOfCards &mDeck, vector<Card> &hand); //done
   void showHand(vector<Card> &hand); //done

   string askHitOrStand(DeckOfCards &mDeck, vector<Card> &hand, vector<Card>
&dealerHand, Player p1);

   void split(DeckOfCards &mDeck, vector<Card> &hand1, vector<Card> &hand2, Player p1);
   void stand(DeckOfCards &mDeck, vector<Card> &hand, vector<Card> &dealerHand, Player
p1);
   int calcValueOfHand(vector<Card> &hand);

   void printAccounts();
private:
   Player mPlayer;
   Player mDealer;
   DeckOfCards mDeck;
```

```
    int playerScore;
    int dealerScore;
};

#endif /* Game_h */
```

Game.cpp

```
#include "Game.h"
#include <iostream>
#include<vector>

using namespace std;

//Constructor contains a deck of cards.
Game::Game() {
    mDeck = DeckOfCards();
}

/**
Constructor that takes in a deck of cards, and the playerScore is set to 0.
@param: DeckOfCards& deck
@param: Player player
*/
Game::Game(DeckOfCards& deck, Player player, Player dealer) {
    mDeck = deck;
    mPlayer = player;
    playerScore = 0;
    dealerScore = 0;
}

/**
Constructor that takes in a deck of cards, a player, and a dealer. Both player and dealer scores are
set to 0.
@param: DeckOfCards& deck
*/
Game::Game(DeckOfCards& deck) {
    mDeck = deck;
    playerScore = 0;
    dealerScore = 0;
```

```
}

/**
Takes in a player to be a player of the game.
@param: Player player
*/
void Game :: setPlayer(Player player) {
   mPlayer = player;

}

/**
Takes in a dealer to be a player of the game.
@param: Player dealer
*/
void Game::setDealer(Player dealer){
        mDealer = dealer;
}

/**
returns the player of the game.
@return: mPlayer
*/
Player Game::getPlayer() const {
   return mPlayer;
}

Player Game::getDealer() const{
        return mDealer;
}
/** Ask the user to enter the account number.
 @return acctNum - the user's account number.
 */
int Game::promptUserForAccountNumber() {
   cout << "Enter your account number: ";
   int acctNum;
   cin >> acctNum;
   return acctNum;
}
```

```
/** Ask the user to enter the amount of money he/she wants to bet.
 @return betAmt - the amount of money user wants to bet.
 */
double Game::promptUserForAmountToBet() {
   cout << "Enter the amount of money to bet: ";
   int betAmt;
   cin >> betAmt;
   return betAmt;
}

/** Randomly select and show the user two cards.
 * Only happens once at the beginning of the game
 @return a vector of the two cards
 */
vector<Card> Game::selectAndShowTwo(DeckOfCards &mDeck, vector<Card> &hand) {
        //Deal two from the array
        Card firstCard = mDeck.dealCard();
        Card secondCard = mDeck.dealCard();

        hand.push_back(firstCard);
        hand.push_back(secondCard);

        return hand;

}

/** Randomly select and show the user a card.
 @return their card
 */
Card Game::selectAndShowOne(DeckOfCards &mDeck, vector<Card> &hand) {
   //Deal two from the array
   Card firstCard = mDeck.dealCard();

   //&hand should update the hand because we pass a refrence
   hand.push_back(firstCard);

   return firstCard;
```

```
}

/**
Acts as a toString for the hand that's passed in. Displays what the user has in their hand.
@param: vector<Card> &hand
*/
void Game::showHand(vector<Card> &hand){
	for(int i=0; i < hand.size(); i++){
		cout<<hand[i].print()<<" | ";

	}
	cout<<endl;
}
/**
Calculates the value of the cards in the player's hand.
@return: int - score
@parameter: vector<Card> &hand, the hand of the player.
**/
int Game::calcValueOfHand(vector<Card> &hand){
	playerScore = 0;
	for(int i=0; i < hand.size(); i++){
		//if the card is an Ace...
		if(hand[i].getFaceValue() == 1){
			//And if the score is less than 10
			if(playerScore <= 10){

				//make the ace worth 11 and add it to score
				playerScore += 11;
				//cout<<"playerscore1: "<<playerScore<<endl;
			}
			else{


				playerScore += 1;
				//cout<<"playerscore2: "<<playerScore<<endl;
			}
		}
		else{
```

```
                playerScore += hand[i].getFaceValue();
                //cout<<"playerscore3: "<<playerScore<<endl;

            }

        }
        return playerScore;

}

/**
h. If the user decides to split, the dealer will draw two cards for the user.
The user now has two hands. Also, an additional bet of equal value to the original bet is
placed on the second hand. Proceed the game as in step f and/or g.

Allows the user to draw another two cards. The user will now have two hands to play with.
@param: DeckOfCards &mDeck
@param: vector<Card> &hand1
@param: vector<Card> &hand2
@param: Player p1
**/
void Game :: split(DeckOfCards &mDeck, vector<Card> &hand1, vector<Card> &hand2,
Player p1)
{

        cout << "Your cards in hand 1 are: " << endl;
        for(int i = 0; i < hand1.size(); i++)
        {
                cout << hand1[i].print() << endl;
        }
        cout<<"The value of hand 1 is: "<<calcValueOfHand(hand1)<<endl;
        cout<<endl;
        cout << "Your cards in hand 2 are: " << endl;
        for(int i = 0; i < hand2.size(); i++)
        {
                cout << hand2[i].print() << endl;
        }
        cout<<"The value of hand 2 is: "<<calcValueOfHand(hand2)<<endl;
        cout<<endl;
```

```
}

/**
If the dealer’s score is greater than or equal to 17, then don’t do anything. If the dealer’s score is less than 17, while it’s less than 17, continue drawing cards until it reaches 17. If the player’s score is less than the dealer’s score, then the player wins. If not, the dealer wins.
@param: DeckOfCards &mDeck
@param: vector<Card> &hand
@param: vector<Card> &dealersHand
@param: Player p1
*/
void Game :: stand(DeckOfCards &mDeck, vector<Card> &hand, vector<Card> &dealersHand, Player p1){
   cout << endl << "The dealers hand contains the cards: " << endl;
   for (int i = 0; i < dealersHand.size(); i ++) {
      cout << dealersHand[i].print() << endl;
   }

   cout << endl << "Dealer's hand value before dealing: " << calcValueOfHand(dealersHand) << endl;

   while (calcValueOfHand(dealersHand) < 17) {
      Card newCard = selectAndShowOne(mDeck, dealersHand);
      cout << "Dealer's new card: " << newCard.print() << endl;
   }
   cout << "Dealers hand value after dealing: " << calcValueOfHand(dealersHand) << endl;
}

/**
The dealer will receive one new card. Stand will be called if the dealer’s score is greater than or equal to 17.
If the dealer’s score is less than 17, then the dealer will get another card.

@param: DeckOfCards &deck
@param: vector<Card> &playersHand
*/
void Game :: hit(DeckOfCards &deck, vector<Card> &playersHand) {
   cout<<"You have drawn a: "<< selectAndShowOne(deck, playersHand).print()<<endl;
   cout << "The value of that hand is now: " << calcValueOfHand(playersHand) << endl;
```

```
}

/**
Prompts the user if they want to stand, hit, or split. It will call the correct functions that the user wants to select.
@param: DeckOfCards &mDeck
@param: vector<Card> &hand
@param: vector<Card> &dealerHand
@param: Player p1
*/
string Game :: askHitOrStand(DeckOfCards &mDeck, vector<Card> &hand, vector<Card> &dealerHand, Player p1)
{
   cout<<"Do you want to hit or stand?"<<endl;
   string decision;
   cin >> decision;

   if (decision == "hit" || decision == "Hit")
   {
      //hit(mDeck, hand, dealerHand);
      return "hit";
   }

   else if(decision == "stand" || decision == "Stand")
   {
      //stand(mDeck, hand, dealerHand);
      return "stand";
   }

   else {
      return "error";

   }
}
```

Sample Run:

Case 1: user wins

- shows that account is now updated
- Utilizes **stand** and **hit**

```
1001: 100
Enter your account number: 1001
Account exists!
You have $100 available to bet.
Enter the amount of money to bet: 10
You are betting $10.
---------------------------------------------
Your cards are:
Two of Diamonds
Eight of Hearts

One of the dealer's cards is:
Five of Spades

The value of your hand is: 10
Do you want to hit, stand, or split?
hit
You have drawn a: Ten of Hearts
The value of that hand is now: 20
Do you want to hit or stand?
stand

The dealers hand contains the cards:
Five of Spades
Ace of Diamonds

Dealer's hand value before dealing: 16
Dealer's new card: Two of Clubs
Dealers hand value after dealing: 18
---------------------------------------------
You won!
The dealer's score: 18
Your score: 20
Your balance is now: 110

Do you want to play again?
yes
Enter your account number: 1001
Account exists!
You have $110 available to bet.
```

Case 2:

- You **hit** bust. And lose your bet money

```
Do you want to play again?
yes
Enter your account number: 1001
Account exists!
You have $110 available to bet.
Enter the amount of money to bet: 10
You are betting $10.
---------------------------------------------
Your cards are:
Eight of Hearts
Queen of Diamonds

One of the dealer's cards is:
Seven of Hearts

The value of your hand is: 18
Do you want to hit, stand, or split?
hit
You have drawn a: Six of Diamonds
The value of that hand is now: 24
---------------------------------------------
Dealer won.
The dealer's score: 17
Your score: 24
Your balance is now: 100

Do you want to play again?
```

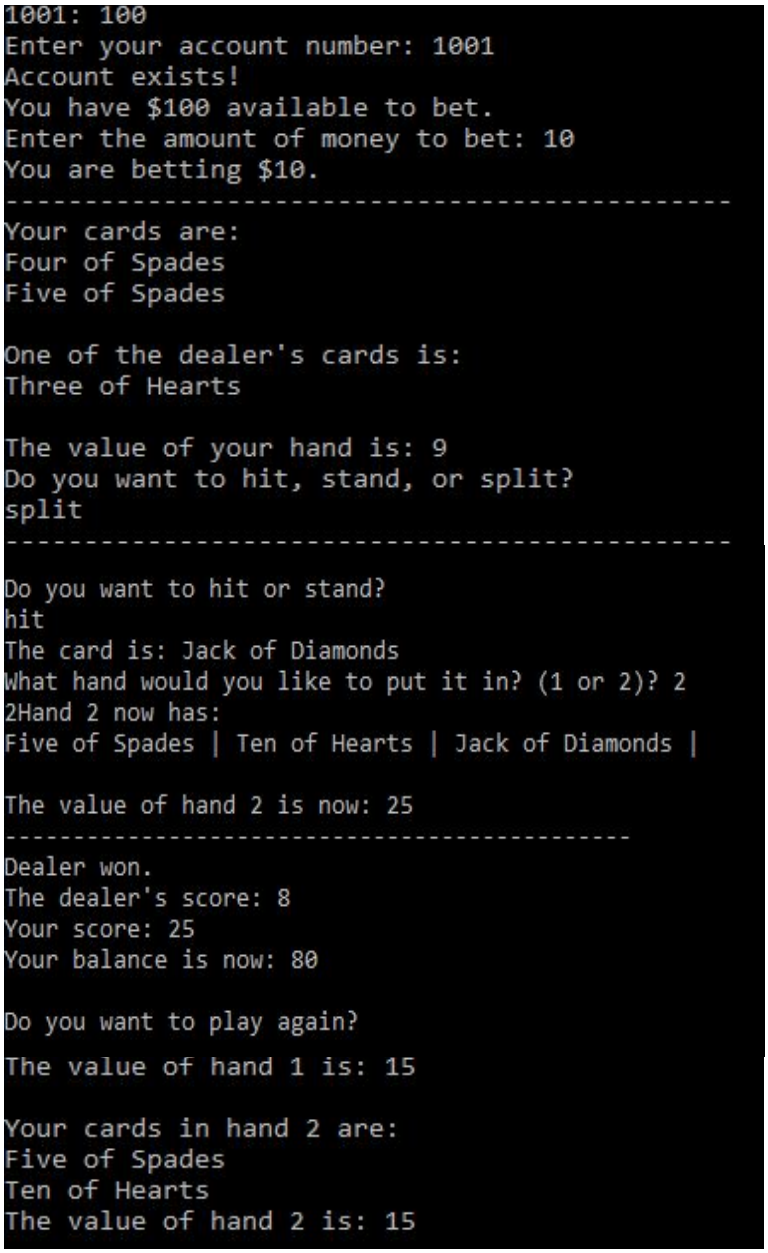
Case 3(right →):

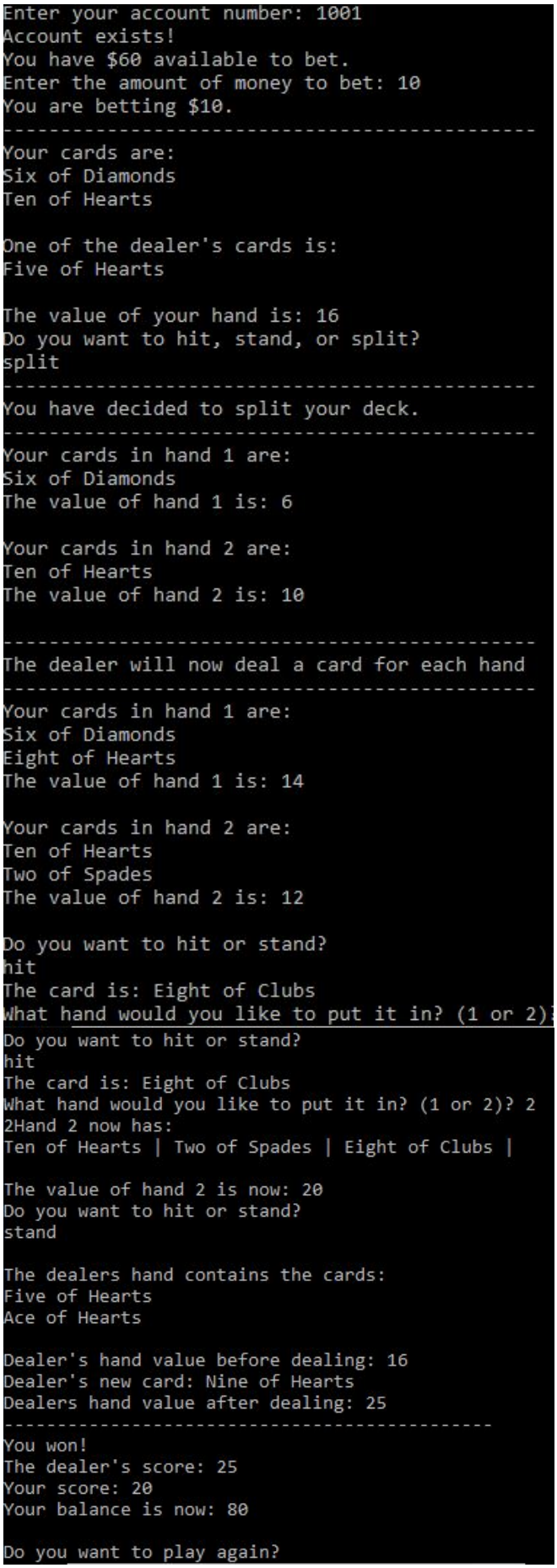
- You **stand** while the dealer draws
- Dealer's score higher than users

```
Do you want to play again?
yes
Enter your account number: 1001
Account exists!
You have $80 available to bet.
Enter the amount of money to bet: 10
You are betting $10.
---------------------------------------------
Your cards are:
Queen of Hearts
Eight of Spades

One of the dealer's cards is:
Jack of Spades

The value of your hand is: 18
Do you want to hit, stand, or split?
stand

The dealers hand contains the cards:
Jack of Spades
Five of Diamonds

Dealer's hand value before dealing: 15
Dealer's new card: Six of Hearts
Dealers hand value after dealing: 21
---------------------------------------------
Dealer won.
The dealer's score: 21
Your score: 18
Your balance is now: 70

Do you want to play again?
```

Case 4:

- Decide to split deck
- Decide to hit hand 2
- End with bust
- Since you split your hands, your bet doubles so you lose 20 dollars leaving you with 80





Case 5:

- Utilizing **split**, **hit**, and **stand**
- User starts with 60 but split so their bet of 10 dollars doubles to 20 and since they won, they get a total of 80 dollars.

Case 6: Tie

- User wins half of what they bet

```
Account exists!
You have $100 available to bet.
Enter the amount of money to bet: 10
You are betting $10.
---------------------------------------------
Your cards are:
Jack of Clubs
Queen of Spades

One of the dealer's cards is:
Ten of Spades

The value of your hand is: 20
Do you want to hit, stand, or split?
stand

The dealers hand contains the cards:
Ten of Spades
Jack of Hearts

Dealer's hand value before dealing: 20
Dealers hand value after dealing: 20
---------------------------------------------
You've tied with the dealer. You will win half of what you've bet.
The dealer's score: 20
Your score: 20
Your balance is now: 105

Do you want to play again?
no
Total amount of betting money: 10
Total amount of money won: 5

--------------------------------
Process exited after 75.58 seconds with return value 0
Press any key to continue . . .
```

UML Class Diagram: