Project 2: Blackjack

Andrew Willis
CIS-17A
Dr. Lehr

Introduction:
Title: Blackjack

This is the popular card game Blackjack. Users start with 1000 dollars to use as capital in placing bets on the game. Players may hit, split, or stand depending on their hand's total and, once a stand is chosen, the dealer draws their cards. Their cards continue to draw until their total is equal to or greater than 17. The hand totals are then compared, the bet is resolved, and the loop continues.

Summary:
Project Size: About 1011 lines (Including All Lines From Specification and Implementation)
Number of Classes: 7
Number of Implementation Files: 5 (AbsBank and BankException are handled in their header files)
Versions: 11 (Missing version 7)

External Overview:

This project contains all of the topics covered in chapters 9 through 12 as shown by the attached check-list file. It has the capacity to be expanded upon by adding insurance and doubling down functions as well as increased capabilities between game sessions.

I'm fairly satisfied with this project, but ran into a few issues. These mainly surrounded dynamic memory allocation and deletion and the scope of the project being much larger than anticipated. I'm confident that there are some simpler ways for this project to be refactored, but the limitations of time and concepts lead me to believe this was the best possible product.

PROJECT 2 ADDENDUM:

Adding classes to this project made it feel much more robust and easier to handle. The focus was mainly on implementing classes where structs used to be rather than the logic of the game functioning. Sadly, due to this, the product is harder for the end user to appreciate, but shows the aspects of the class we've been learning in these recent chapters well.

Internal Overview:

The main function covers the player's three choices when starting the game. They may play the main game, check a previous player's score via random file access, and exiting the game. The score function handles all file reading and writing. The main game leads into the mainGame function which handles all of the logic behind blackjack.

The main game of blackjack requires a deck of cards and an "automated" dealer. This dealer can be considered automated as their actions are determined by the value in their hand. They must hit on a total below 17 and stand on a value over 17. The game begins with both dealer and player drawing 2 cards from the deck and reviewing their totals. The player only sees

the first card the dealer drew. The program checks for early blackjacks or pushes from either side before entering the main game loop.

Within the main game loop, the program checks the two cards the player drew and decides if they are able to be split. In the event of the two cards having the same value, the player may choose to take up a second hand to play. The game generates this hand and plays it first. Once either hand is in play, the player may choose to hit or stand. On a hit, the program draws a card and checks the new values. In the event of a bust (total over 21), the program will attempt to remedy this by adjusting any aces to be a value of 1 instead of 11. If this still results in a bust, the player loses. If the player does not bust, they will be prompted to choose hit or stand again. In the event of a stand, the program compares the final totals of the dealer the player's hand and deals out the proper results. If the player's hand was split, they return to their original hand with a card removed and play it as stated above. The player may not split if they cannot cover the bet amount required to do so (twice their current bet).

Once the player enters -1 as their bet value, they are kicked out of the main game loop and the final total of their money is output to the main function. That value is saved in a Results.txt and Results.bin file to be viewed by the score function later.

PROJECT 2 ADDENDUM:

The entire program has shifted from utilizing struct based objects to class based objects, exhibiting object orientation. The Hand and Card structs are now classes and the Bank, PlayerBank, AbsBank, and BankException classes have been added for more functionality. Each of these classes interacts with one another to provide the same functionality of the previous project's version, but do so in an object oriented way.

Check Off List PROJECT 2:

Classes	**Line Location**
Instance of a Class	340
Private Data Members	11 (Hand.h)
Specification vs Implementation	21 (Card.h) and 25 (Card.cpp)
Inline	9 (Bank.h)
Constructors	15 and 19 (Hand.h)
Destructors	17 (Hand.h)
Array of Objects	8 (Deck.h)

UML	See Included UML diagram

More About Classes

Static	10 (Deck.h)
Copy Constructors	21 (Hand.h)
Operator Overloading	13, 15 ,17 (Card.h) (Declared here, shown in Card.cpp)
Aggregation	8 (Deck.h) (Deck always HAS A array of CARDS)

Inheritance

Protected Members	6 (Deck.h)
Base Class to Derived	6 (PlayerBank.h) (PlayerBank derives from Bank derives from AbsBank)
Polymorphic Association	625 (main.cpp) (PlayerBank calls getSMoney which is derived from Bank which is overridden from AbsBank)
Abstract Classes	13 (AbsBank.h) (virtual getSMoney function)

Advanced Classes

Exceptions	28 (AbsBank.h)
Templates	28 (main.cpp)
STL	374 (main.cpp) (vector class)

Pseudo Code:

```
//Create dynamic array for the deck
    //Create a struct to hold information on all cards generated
    //Fill it from a file with 52 card structs
    //Shuffle that array
    //Create a dynamic hand array
    //Randomly draw a card into the hand array
    //Resize the deck array and remove the card
    //Give the player a set number as starting money
    //Take in a bet
    //Draw another card into the hand array
    //Resize the deck array
    //Create a dynamic array for the dealer's cards
    //Draw a card for the dealer array
    //Resize deck
    //Reveal card drawn by dealer to player
    //Draw another card but don't reveal it
    //Resize deck
    //Check if blackjack is already gained by the player
        //If so check if the dealer also has blackjack
        //If not, give 1.5 times money back to player
        //If so, give 1.0 times money back to player.
    //Check if blackjack is gained by dealer
        //If so, take all bets from player
        //Do not check player's again as its done in previous check
    //If neither, enter Play


    //Check if player can split and give option to
        //If split, create a second dynamic hand array to use temp
        //Put one card into it and loop the play for both of them


    //Player chooses to stand or hit.
        //If stand, continue
        //If hit, deal another card from deck.
            //Resize deck
            //Check card totals
                //If Less than 21, continue
                //If 21, blackjack
```

```
            //If greater than 21, check if an ace is in play
                //If so, change ace to value 1 and continue.
                //If not, bust.  Player loses.
//Repeat stand or hit until stand
//Once stand, reveal dealer's second card.
   //If <17, draw to dealer's hand unti >17
   //Resize deck every time
   //If >17, dealer stands and continues
//Aces always 11 for dealers???
//Repeat until both parties stand
//When both stand, check both totals starting with player.  If >21, bust
//If <21, check dealer
//If >21, bust.  If <21, compare both.  If player > dealer, player wins
//If dealer > player, dealer wins
//Apply changes in money and restart.
//If no money is left, player loses.

//Every time a card is drawn, check size of deck array.  If it's below 10,
//take in all non-used cards (nothing in hands) and add them back in to the deck and shuffle
```

Important Variables:

canSplit - bool utilized to check if the player is able to split their hand

isSplit - bool utilized to see if the player has chosen to split their hand

betValue - float utilized to add or subtract money from the players total

startingMoney - float utilized to keep track of money gained or lost during the game

enterLoop - bool utilized to see if the game has been won or lost early

playerHand, unusedHand, splitHand, dealerHand - Hand structs utilized to handle the game logic

currentTotal - int used to compared against dealerTotal to check for win conditions

dealerTotal - int used to compare against currentTotal to check for win conditions

shownDealerTotal - int used to show the player the dealer's total on the first turn

leavingGame - a superfluous bool variable that maintains the game loop. Never changes so could be replaced with a "while(true)" statement

Important Functions:

fillDynamicDeck(Card*, int); - utilized to fill the unusedCards Deck Class

drawCard(Hand, Hand, int); - utilized to draw cards for use

resizeAfterDraw(Hand, int); - utilized after the drawCard function to resize the array

int mainGame(int, Card*); - the entire game of Blackjack

int adjustForAces(Hand); - allows aces to be both high and low

Important Classes:

Bank - Derives from AbsBank, holds info for returning value in the bank.

PlayerBank - Holds the player's money at runtime.

Deck - Holds information about the full deck of cards.

Hand - Holds information about any given hand derived from Deck.

AbsBank - Holds important operations that are inherited by Bank and PlayerBank.

BankException - Handles overdrafting the bank classes.

Card - Holds all information about each individual card.

Version Differences:
1.0 - Psuedocode
1.1 - Filling the unusedCards struct properly
1.2 - Perfecting the drawCard function and testing
1.3 - Testing win/lose condition checks, testing limits of reuse
1.4 - Added most of the game's logic
1.5 - Added splitting and text file search for scores
1.6 - Refactored code for readability, improved splitting, added binary file compatibility and tweaked the outputs

PROJECT 2 ADDENDUM:

Version Differences:
1.7 - Lost during a power outage, version differences unknown.
1.8 - Began working on Card and Hand classes, implemented usage minimal.
1.9 - Began working on Bank and PlayerBank in terms of Friend Classes and Polymorphism. Friend Class implementation has since been removed. Card and Hand class logic finalized.
1.10 - Addition of a Deck class to demonstrate polymorphism through the Hand class. Improvements made on the Bank class. Full implementation of classes instead of structs in main. Differentiation between Specification and Implementation made.
1.11 - Addition of AbsBank class to demonstrate abstract and derived class functionality. Implementation of exception throwing and STL class vector. Finalization of the project as a whole.

Addendum 11/6/2022

The project deadline has been extended, but I still feel content in leaving some aspects of blackjack out of the program. I intend to add these into the game for the next project.

Addendum 12/16/2022

With the project deadline coming to a close, I would like to reflect on this project as a whole. I've learned so much from programming this game that I simply didn't know existed despite taking higher level C++ classes. I did not add the previous addendum's proposed mechanics, not due to time, but because it was simply not in the best interest for learning. Logic can be reasoned out, but the syntax of classes, templates, and polymorphism as a whole can only be learned by doing.