***California State University, Long Beach***

**CECS 282** *C++ for Java Programmers*

Final Project: Blackjack Slot Machine

December 15, 2017

Steven Garcia & Gabriel Flores

Introduction

The purpose of this program is to simulate a Blackjack Slot Machine. The user is prompted for an account number and the amount of money to bet. Two user cards will then be selected randomly and will be shown to the user. Another two cards will be randomly selected for the dealer’s hand. The program will display the value of the user’s cards and will prompt one of the following actions: hit, stand, or split. If the value of the user’s cards is less than or equal to 21 and worth more than the dealer’s, the user receives double the betted amount. If the values of the cards are the same, there is a tie and the money is split. Otherwise, the user wins nothing. The user can choose to continue playing after each hand.

Program Analysis

Our final project consists of Player, Account, Hand, and Card classes. The Player class represents a player in the blackjack game, the Account class represents the Player’s account, and the Hand class encapsulates the Card class to represent the Player’s blackjack hand.

Player class has get and setter methods for accessing attributes for their corresponding Account and Hand objects. The Player class also has a bet() method to bet a certain amount, a stand() method to stand, a hit() method to hit and get another card, and a split() method to split his or her hand. The usingSplitHand variable is a Boolean that indicates whether to add a card to the primary hand or the secondary hand. The player also has methods winGame(), loseGame(), and tieGame() to indicate whether or not he or she wins.

The Account class has get and setter methods for its attributes. It has private variables accountNumber and totalAmount to store the account number and amount in the account.

The Hand class has private variables cards, valueOfCards, numberOfCards, and cardMemoryCapacity. It stores the Card objects by having the cards variable point to a dynamic Card array. It has the appropriate get and setter methods to retrieve and manipulate the value and number of cards. It also has the allocateDynamicMemory() function to handle a possible memory leak.

The Card class has the get and setter methods to access the cardType and value. The value of the Card is determined by the cardType and is based off of the traditional set of 13 cards.

The program starts by asking the user for his or her account number and the amount of money to bet. Afterwards, a Player object is created for both the user and the dealer. Then the program executes a while loop that begins by dealing two cards to both the user and the dealer. The cards are randomly generated by the generateRandomCard() function. Both the user’s and dealer’s total value from the cards are output to the screen so that the user is aware of the current cards. After dealing the initial cards, another while loop is executed, which prompts the user to enter a move using the getUserDecision() function. The dealer is set to always hit until the value of his or her cards is above 15. If either the user’s or the dealer’s cards go over 21, the other player wins and the user is prompted to play again with the playAgain() function. If the user decides to play again, the loop reiterates. Otherwise, the loop breaks and the program ends.

Description and Analysis

Class: Card

Variables:

* string cardType; =>specifies the name of the card
* int value; =>specifies the associated value with the card

Functions:

* Card();=> to instantiate a card with no default values
* Card(int); => to instantiate a card using factory design pattern
* void setCardType(int); => setter for the card type to identify the card
* string getCardType() const; => getter for the card type
* int getValue() const; => to set the value for corresponding card type
* void setValue(int); => to get the value for corresponding card type

Class: Hand

Variables:

* Card\* cards = nullptr; =>keeps list of the cards that user is holding
* string type; =>specifies what kind of hand is playing (whether a split hand)
* int valueOfCards; =>total value of hand
* int numberOfCards; =>keeps track of amount of cards
* int cardMemoryCapacity; =>keeps track of amount of memory allocated for array

Functions:

* Hand();=> instantiate a hand with default values and set cards to dynamically allocated array
* Hand(string); => to instantiate a hand with specified type, such as Split
* ~Hand();=> destructor to release dynamically allocated array
* void addCard(Card&); => to add card to list of cards
* string getType() const; => to get type of hand
* void setType(string); => to set type of hand
* int getValueOfCards() const; => to get total value of the current hand
* void setValueOfCards(int); => to set the value of the current hand
* int getNumberOfCards() const; => to get amount of cards
* void setNumberOfCards(int); => to set the amount of cards
* void allocateDynamicMemory();=> to handle dynamic memory and reallocate more space for cards pointer

Class: Account

Variables:

* string accountNumber; =>valid account number
* double totalAmount; =>amount that a user can gamble from

Functions:

* Account();=> to instantiate with default values of 0
* Account(string); => to specify the account number
* ~Account();=> destructor for the account
* string getAccountNumber() const; => to get the account number
* void setAccountNumber(string); => to set the account number
* double getTotalAmount() const; => to get the total amount that user can potentially gamble
* void setTotalAmount(double); => to set the total amount that user can potentially gamble

Class: Player

Variables:

* Account playerAccount; => associated account
* Hand hand; => current hand, array of cards
* Hand splitHand; =>split hand, array of cards
* double betAmount; =>amount that user is currently betting
* bool hasSplit; =>whether or not the user has decided to split
* bool usingSplitHand; =>whether or not currently playing with split hand

Functions:

* Player();=> instantiate a player object with default values
* Player(Account); => instantiate a player with a corresponding account
* ~Player();=>destructor for player class
* void setAccount(Account); => set account object
* double getAccount() const; => get total value within the account object
* double bet(double); => take in amount from account to bet on a game of blackjack
* void resetHand();=> create a new hand, destructor of Hand class will be called
* bool isSplit() const; => determine if user has decided to split hands
* void switchHands();=> to swtich to different hand
* string getCurrentHand();=> to get the type of the current hand
* void setNumberOfCardsInHand(int); => set number of cards in hand
* int getNumberOfCardsInHand() const; => get total number of cards in hand
* void addCard(Card&); => to add card to the current hand
* int getValueOfCards();=> to get value of the current hand
* void stand();=> user decide to not receive a card for current iteration
* void hit(Card&); => user decide to receive a card for current iteration to reach closer to 21
* void split(Card&, Card&); => user decide to split hands, which doubles amount being bet
* void winGame(); => user wins game which earms them doubles the amount that they bet
* void loseGame(); => user loses all the money that they bet
* void tieGame();=> user and dealer match, there user loses half of what they bet

Class: SlotMachine

Variables:

* bool isInvalidAccount = true; => to determine if invalid account entered
* bool isInvalidAmount = true; => to determine if invalid amount entered
* bool userWantsToPlay = true; => to determine if need to run game functionality
* int decimalOccurances = 0; => for input validation and counting decimals in double
* string accountNumber; => for the valid integers specified account number
* string userInputDecision; => for user input regarding strings
* double amountBeingGambled; => for user input of the current amount to gamble
* Account userAccount; => object for user account specified number and amount
* Player dealer; => object for dealer and their specified cards
* Player user; => object for the user and their specified cards
* const int POSSIBLE\_CARDS = 13; =>

Functions:

* Card generateRandomCard();=> generates a random card and returns a ponter to the address
* string getUserDecision(); => to accept user input
* double getAmountToBet(string, double); => to accept valid amount to bet, accepting in user input and well as comparing to the max in the user’s account

Program Code

//

//  main.cpp

//  BlackJackSlotMachine

//

//  Created by Steven Garcia on 11/20/17.

//  Copyright © 2017 Steven Garcia. All rights reserved.

//

#include <iostream>

#include <stdlib.h> // for rand() and srand() function

#include <exception>

#include <algorithm> // for transform()

#include <iomanip>

#include "Player.h"

#include "Account.h"

#include "Card.h"

using namespace std;

Card generateRandomCard();

const int POSSIBLE\_CARDS = 13;

bool playAgain();

string getUserDecision();

double getAmountToBet(string, double);

int main(int argc, const char \* argv[]) {

    srand(time(0));

    cout << fixed << showpoint << setprecision(2);

    bool isInvalidAccount = true;

    bool isInvalidAmount = true;

    bool userWantsToPlay = true;

    int decimalOccurances = 0;

    string accountNumber;

    string userInputDecision;

    double amountBeingGambled;

    Account userAccount;

    Player dealer;

    Player user;

    /\* ----------------

     Accept a valid user account number.

     ---------------- \*/

    do {

        cout << "Enter your account number: ";

        try {

            cin >> accountNumber;

            for (int i = 0; i < accountNumber.length(); i++) {

                if (48 > accountNumber[i] || 57 < accountNumber[i]) {

                    throw "Invalid Character";

                }

            }

            isInvalidAccount = false;

            userAccount.setAccountNumber(accountNumber);

        } catch (char const\* exc) {

            cout << "Invalid account number." << endl;

        }

    } while (isInvalidAccount);

    /\* ----------------

     Set amount in account.

     ---------------- \*/

    do {

        try {

            cout << "\nEnter the amount of money you want in account (Ex: 1150.49): ";

            //getline(cin, userInputDecision); // take in input as a string

            cin >> userInputDecision;

            for (int i = 0; i < userInputDecision.length(); i++) {

                if (userInputDecision[i] == 46) {

                    decimalOccurances++;

                    if (decimalOccurances > 1) {

                        throw "Invalid Character, too many decimal points";

                    }

                    continue;

                }

                if (48 > userInputDecision[i] || 57 < userInputDecision[i]) {

                    throw "Invalid Character";

                }

            }

            // str needs to be converted to c\_str then converted to double value

            amountBeingGambled = atof(userInputDecision.c\_str());

            userAccount.setTotalAmount(amountBeingGambled);

            cout << endl; // Output formatting

            cout << "The total amount in your account: $" << amountBeingGambled << endl;

            isInvalidAmount = false;

        } catch (char const\* exc) {

            cout << "Invalid amount entered. Please enter in format example: 2500.00 " << endl;

        }

    } while( isInvalidAmount );

    user.setAccount(userAccount);

    /\* ----------------

     Game functionality.

     ---------------- \*/

    cout << endl; // Output formatting

    cout << "Welcome to Blackjack! \n" << endl;

    // Loop Until User decides not to play

    while (userWantsToPlay) {

        amountBeingGambled = getAmountToBet(userInputDecision, user.getAccount());

    user.bet(amountBeingGambled);

        cout << "Total amount in your account: $" << user.getAccount() << endl;

    user.resetHand();

    dealer.resetHand();

    // generate initial four random cards, 2 for dealer, and 2 for player

    for (int i = 0; i < 2; i++) {

        Card newUserCard = generateRandomCard();

        user.addCard( newUserCard );

        Card newDealerCard = generateRandomCard();

        dealer.addCard( newDealerCard );

    }

    // Infinite Loop will Break when Dealer or User loses/wins

    while(true){

//     while(dealer.getValueOfCards() < 15 ){

//     // Dealer will continue to hit as long as cards are less than 15

//     Card newDealerCard = generateRandomCard();

//     dealer.hit(newDealerCard);

//     if(dealer.getValueOfCards() > 21){

//     break;

// }

// }

        cout << "The value of the dealer's cards is " << dealer.getValueOfCards() << endl;

        cout << "The value of your current hand is " << user.getValueOfCards() <<  endl;

        // add input validation

        userInputDecision = getUserDecision();

        if(userInputDecision == "HIT"){

        cout << endl; // Output formatting

        cout << "Dealing card..." << endl; // Output formatting

        Card newUserCard = generateRandomCard();

            user.hit( newUserCard );

                Card newDealerCard = generateRandomCard();

                dealer.hit(newDealerCard);

                if (dealer.getValueOfCards() == user.getValueOfCards() && user.getValueOfCards() == 21) {

                    user.tieGame();

                    break;

                }

                else if(dealer.getValueOfCards() > 21){

                    user.winGame();

                    break;

                }

            else if(user.getValueOfCards() > 21){

            cout << "The value of your hand is " << user.getValueOfCards() << endl;

            user.loseGame();

            break;

}

else if(user.getValueOfCards() == 21){

cout << "The value of your hand is " << user.getValueOfCards() << endl;

user.winGame();

break;

}

}

else if(userInputDecision == "SPLIT"){

// Need to add Player implementation of split scenario

Card card1 = generateRandomCard();

Card card2 = generateRandomCard();

user.split(card1, card2);

if(user.getValueOfCards() > 21){

            user.loseGame();

            break;

}

}

else if(userInputDecision == "STAND"){

user.stand(); // Arbitrary function; does nothing

                Card newDealerCard = generateRandomCard();

                dealer.hit(newDealerCard);

                if (dealer.getValueOfCards() == user.getValueOfCards() && user.getValueOfCards() == 21) {

                    user.tieGame();

                    break;

                }

if(dealer.getValueOfCards() > 21){

user.winGame();

break;

}

else{

if(user.getValueOfCards() < dealer.getValueOfCards()){

user.loseGame();

break;

}

else{

user.winGame();

break;

}

}

}

}

// Prompt user to play again

userWantsToPlay = playAgain();

}

    return 0;

}

Card generateRandomCard() {

    int randomCardIdentifier = 0;

    // Loop ensures that 0 is not returned

    while(randomCardIdentifier <= 0){

    randomCardIdentifier = rand() % POSSIBLE\_CARDS;

}

    //Card newCard( randomCardIdentifier );

    Card\* newCard = new Card( randomCardIdentifier );

    return \*newCard;

}

bool playAgain(){

char userChoice;

while(true){

// loop only breaks when given valid answer (Y/N)

cout << "Would you like to play again (Y/N)?" << endl;

cin >> userChoice;

if(userChoice == 'Y') return true;

else if (userChoice == 'N') return false;

else cout << "Please enter a valid answer (Y/N)" << endl;

}

}

string getUserDecision(){

string input;

while(true){

cout << "Do you want to HIT, STAND, or SPLIT? : ";

cin >> input;

transform(input.begin(), input.end(), input.begin(), ::toupper);

if(input == "HIT")

    return "HIT";

else if(input == "SPLIT")

return "SPLIT";

else if(input == "STAND")

return "STAND";

else

cout << "Please enter a valid answer. " << endl;

}

}

double getAmountToBet(string userInputDecision, double maxAmount) {

    bool isInvalidAmount = true;

    int decimalOccurances = 0;

    double amountBeingGambled = 0;

    do {

        try {

            cout << "\nEnter the amount of money you want to bet (less than $" << maxAmount << "): ";

            //getline(cin, userInputDecision); // take in input as a string

            cin >> userInputDecision;

            for (int i = 0; i < userInputDecision.length(); i++) {

                if (userInputDecision[i] == 46) {

                    decimalOccurances++;

                    if (decimalOccurances > 1) {

                        throw "Invalid Character, too many decimal points";

                    }

                    continue;

                }

                if (48 > userInputDecision[i] || 57 < userInputDecision[i]) {

                    throw "Invalid Character";

                }

            }

            // str needs to be converted to c\_str then converted to double value

            amountBeingGambled = atof(userInputDecision.c\_str());

            cout << endl; // Output formatting

            if (amountBeingGambled > maxAmount) {

                cout << "Invalid amount. Must enter amount less than or equal to $" << maxAmount << endl;

                continue;

            }

            cout << "The amount being gambled is: $" << amountBeingGambled << endl;

            isInvalidAmount = false;

        } catch (char const\* exc) {

            cout << "Invalid amount entered. Please enter in format example: 2500.00 " << endl;

        }

    } while( isInvalidAmount );

    return amountBeingGambled;

}

#ifndef Hand\_h

#define Hand\_h

#include <string>

#include "Card.h"

using namespace std;

class Hand{

private:

Card\* cards = nullptr;

string type;

int valueOfCards;

int numberOfCards;

int cardMemoryCapacity;

public:

Hand();

Hand(string);

~Hand();

void addCard(Card&);

string getType() const;

void setType(string);

        int getValueOfCards() const;

        void setValueOfCards(int);

        int getNumberOfCards() const;

        void setNumberOfCards(int);

        void allocateDynamicMemory();

};

#endif

/\* ----------

Hand.cpp

---------- \*/

#include "Hand.h"

#include <iostream>

using namespace std;

Hand::Hand(){

this->type = "Primary";

this->numberOfCards = 0;

this->valueOfCards = 0;

this->cardMemoryCapacity = 10;

this->cards = new Card[cardMemoryCapacity];

}

Hand::Hand(string handType){

this->type = handType;

this->numberOfCards = 0;

this->valueOfCards = 0;

this->cardMemoryCapacity = 10;

this->cards = new Card[cardMemoryCapacity];

}

Hand::~Hand(){

    delete [] cards;

}

void Hand::addCard(Card& newCard){

// ensure have enough memory in dynamic array

if(this->numberOfCards < this->cardMemoryCapacity){

    \*(cards + numberOfCards + 1) = newCard;

    if(newCard.getCardType() == "Ace"){

    // Checks the value that a new Ace should be: 1 (default), or 11

    if(this->valueOfCards + 11 == 21)

    setValueOfCards(this->valueOfCards + 11);

}

else{

setValueOfCards(this->valueOfCards + newCard.getValue());

}

    this->numberOfCards++;

        if (this->numberOfCards == this->cardMemoryCapacity-1) {

            allocateDynamicMemory();

        }

}

else{

        // allocate internal memory

        allocateDynamicMemory();

cout << "Card Memory Capacity exceeded. Unable to add new card. " << endl;

}

}

string Hand::getType()const{

return type;

}

void Hand::setType(string handType){

type = handType;

}

int Hand::getValueOfCards() const{

return this->valueOfCards;

}

void Hand::setValueOfCards(int value){

this->valueOfCards = value;

}

int Hand::getNumberOfCards() const{

return this->numberOfCards;

}

void Hand::setNumberOfCards(int num){

this->numberOfCards = num;

}

void Hand::allocateDynamicMemory() {

    cardMemoryCapacity \*= 2;

    Card\* newCardPtr = new Card[cardMemoryCapacity];

    for (int i = 0; i < numberOfCards; i++) {

        \*(newCardPtr + i) = cards[i];

    }

    delete [] cards;

    cards = newCardPtr;

}

#ifndef Card\_h

#define Card\_h

#include <string>

using namespace std;

class Card {

    private:

        string cardType;

        int value;

    public:

        Card();

        Card(int);

        void setCardType(int);

        string getCardType() const;

        int getValue() const;

        void setValue(int);

};

#include "Card.h"

using namespace std;

Card::Card() {

}

/\*\*

 This constructor allows for instantiating

 of any one of possible cards based on the

 argument of a randomly generated number.

 \*\*/

Card::Card(int randomNum) {

    setCardType(randomNum);

}

string Card::getCardType() const {

    return this->cardType;

}

void Card::setCardType(int randomNum) {

    switch (randomNum) {

        case 1:

            // ace

            this->cardType = "Ace";

            this->value = 1; // 1 or 11 based on hand

            break;

        case 2:

            this->cardType = "Face Value";

            this->value = 2;

            break;

        case 3:

            this->cardType = "Face Value";

            this->value = 3;

            break;

        case 4:

            this->cardType = "Face Value";

            this->value = 4;

            break;

        case 5:

            this->cardType = "Face Value";

            this->value = 5;

            break;

        case 6:

            this->cardType = "Face Value";

            this->value = 6;

            break;

        case 7:

            this->cardType = "Face Value";

            this->value = 7;

            break;

        case 8:

            this->cardType = "Face Value";

            this->value = 8;

            break;

        case 9:

            this->cardType = "Face Value";

            this->value = 9;

            break;

        case 10:

            this->cardType = "Face Value";

            this->value = 10;

            break;

        case 11:

            this->cardType = "King (FaceCard)";

            this->value = 10;

            break;

        case 12:

            this->cardType = "Queen (FaceCard)";

            this->value = 10;

            break;

        case 13:

            this->cardType = "Jack (FaceCard)";

            this->value = 10;

            break;

    }

}

int Card::getValue() const {

    return this->value;

}

void Card::setValue(int value) {

    this->value = value;

#ifndef Account\_h

#define Account\_h

#include <string>

using namespace std;

class Account {

    private:

        string accountNumber;

        double totalAmount;

    public:

        Account();

        Account(string);

        ~Account();

        string getAccountNumber() const;

        void setAccountNumber(string);

        double getTotalAmount() const;

        void setTotalAmount(double);

};

#endif /\* Account\_h \*/

#include "Account.h"

Account::Account() {

    accountNumber = "";

    totalAmount = 0.0;

}

Account::Account(string accountNumber) {

    this->accountNumber = accountNumber;

}

Account::~Account(){

}

string Account::getAccountNumber() const {

    return this->accountNumber;

}

void Account::setAccountNumber(string accountNumber) {

    this->accountNumber = accountNumber;

}

double Account::getTotalAmount() const {

    return this->totalAmount;

}

void Account::setTotalAmount(double totalAmount) {

    this->totalAmount = totalAmount;

}

#ifndef Player\_h

#define Player\_h

#include "Account.h"

#include "Hand.h"

#include "Card.h"

#include <string>

class Player {

    private:

        Account playerAccount;

        Hand hand;

        Hand splitHand;

        double betAmount;

        bool hasSplit;

        bool usingSplitHand;

    public:

        Player();

        Player(Account);

        ~Player();

        void setAccount(Account);

        double getAccount() const;

        double bet(double);

        void resetHand();

        bool isSplit() const;

        void switchHands();

        string getCurrentHand();

        void setNumberOfCardsInHand(int);

        int getNumberOfCardsInHand() const;

        void addCard(Card&);

        int getValueOfCards();

        void stand();

        void hit(Card&);

        void split(Card&, Card&);

        void winGame();

        void loseGame();

        void tieGame();

};

#endif /\* Player\_h \*/

#include "Player.h"

#include "Card.h"

#include <iostream>

using namespace std;

Player::Player() {

    this->hasSplit = false;

    this->usingSplitHand = false;

    this->betAmount = 0;

}

Player::Player(Account accountNumber) {

    this->playerAccount = accountNumber;

    this->hasSplit = false;

    this->usingSplitHand = false;

    this->betAmount = 0;

}

Player::~Player(){

}

void Player::setAccount(Account userAccount) {

    this->playerAccount = userAccount;

}

double Player::getAccount() const {

    return playerAccount.getTotalAmount();

}

double Player::bet(double amount){

this->playerAccount.setTotalAmount(this->playerAccount.getTotalAmount() - amount);

this->betAmount += amount;

    return this->betAmount;

}

void Player::resetHand(){

// !!!Potential Memory Leak!!!

Hand\* newHand = new Hand();

this->hand = \*newHand;

}

void Player::switchHands(){

usingSplitHand = !usingSplitHand;

}

bool Player::isSplit() const{

return hasSplit;

}

int Player::getNumberOfCardsInHand() const {

    if(usingSplitHand){

return splitHand.getValueOfCards();

}

    else{

    return hand.getValueOfCards();

}

}

string Player::getCurrentHand(){

if(usingSplitHand){

return splitHand.getType();

}

else{

return hand.getType();

}

}

void Player::addCard(Card& newCard) {

    hand.addCard(newCard);

}

int Player::getValueOfCards() {

if(usingSplitHand){

return splitHand.getValueOfCards();

}

    else{

    return hand.getValueOfCards();

}

}

void Player::stand() {

// Player does nothing

}

void Player::hit(Card& newCard) {

    if(usingSplitHand) {

    splitHand.addCard(newCard);

}

    else{

    hand.addCard(newCard);

}

}

void Player::split(Card& handCard, Card& splitCard) {

this->hasSplit = true;

bet(this->betAmount); // Double the bet

hand.addCard(handCard);

splitHand.addCard(splitCard);

splitHand.setType("Secondary");

}

void Player::winGame() {

    this->playerAccount.setTotalAmount(this->playerAccount.getTotalAmount() + (2 \* this->betAmount));

    cout << "You win $" << betAmount << endl;

    cout << "The total amount in your account is $" << getAccount() << endl;

    betAmount = 0;

}

void Player::loseGame() {

cout << "You lose $" << betAmount << endl;

    cout << "The total amount in your account is $" << getAccount() << endl;

    betAmount = 0;

}

void Player::tieGame(){

// User bet is split in half

this->playerAccount.setTotalAmount(this->playerAccount.getTotalAmount() + (this->betAmount / 2));

    cout << "Tie game! Total you lossed is " << (this->betAmount / 2) << endl;

    cout << "The total amount in your account is $" << getAccount() << endl;

    betAmount = 0;

}