ECE 3220 Final Project Report

Andrew Kirkham & Jeff Schulz

Abstract

S&K Banking is a banking application that allows users to create accounts, view and edit the details to their accounts, and make deposits and withdrawals to their accounts. It also allows a single manager to view all accounts, close accounts, and approve accounts.

Introduction

When the program is executed the user is welcomed to the bank and asked to either enter their user ID or to create a new account. If the ID entered is that of the manager the program asks the user for a password. If the password is correct it will display a menu labeled "Manager Mode" that allows the manager to view all accounts, close a user account, and approve new bank accounts. If the ID entered is that of a user and that user ID does exist then the program will ask the user for a password. If the password is correct it will display a menu welcoming the specific user's name and the option to view the balances of their accounts, deposit money into an account, withdraw money from an account, create a new account (checkings or savings, must be approved by the manager), and to edit details of the account. Our goal was to create a banking system with similar functionality to that of a real bank. Our main motivation for our project was to create a program that would satisfy all the requirements of the course while creating something that both of us are interested in, finance.

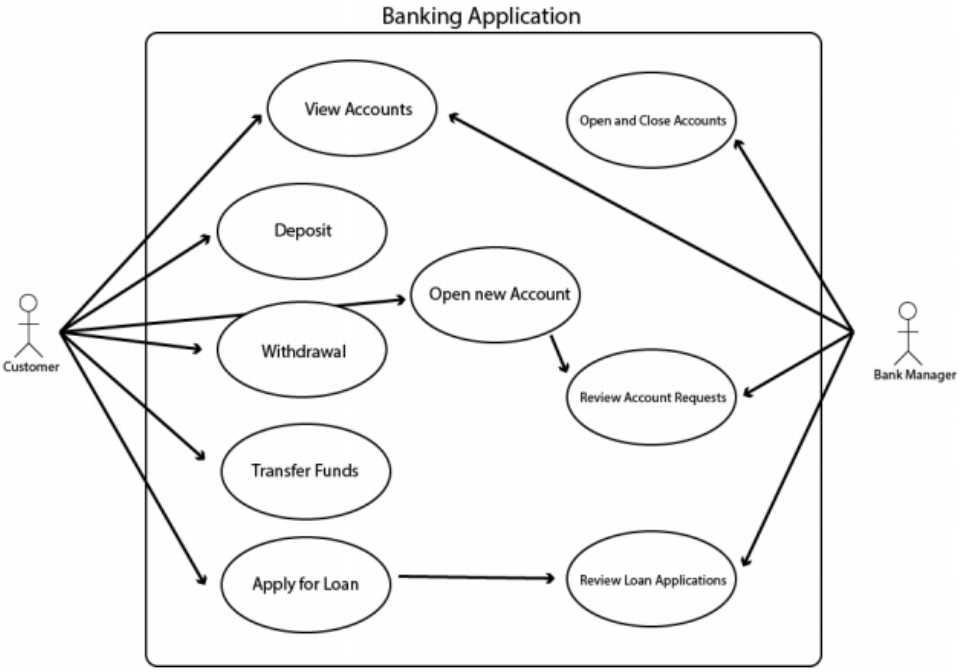
Background

Without getting into too many specifics we are mainly drawing inspiration from various banking applications such as US Bank. A major difference between how we are handling accounts is that we are putting them into a text file rather than using a database so most of the inspiration comes from what the user can do like creating and viewing accounts and making deposits and withdrawals. Our application would be used by people so they can keep their money secure, view it at any time, and earn interest over time if they have a savings account.

Proposed method / System description / Implementation

For the program itself we started with a login so that both a user or a manager could login with a password that is encrypted for security purposes. Afterwards the menus are designed so that the user can interact with all their accounts in an intuitive way. As far was the code itself is concerned many of the functions and other functionalities (such as encryption) are done in separate header files so that the main file isn't as cluttered. Our program went through a few different iterations. The first iteration had limited functionality, the user had the ability to create a new account, deposit or withdraw money into that account, view their balance, display all accounts, close one of their accounts and to edit the details of their account. Later iterations separated some of these functionalities between the users and the manager. Later iterations also include an encrypted password for security purposes. Also savings accounts will now earn interest over time. To go more in depth about how the system works specifically, when the program is opened a welcome message is displayed along with a login that will prompt the user to enter their ID or to create a new one. If the user creates a new account they will enter their first and last names and be given an ID and asked to create a password that is then saved to a file 'logins.dat'. If anything goes wrong during this process proper error messages will be displayed. If at the login screen the user enters their ID (user or manager) they will be prompted to input their password before continuing to the main menu. If the password is incorrect then an error message will be displayed and they will have to try again. If the manager is using the system they can choose to view all accounts which will display all the information about all accounts in the system, close an account which will delete that account from the system, or approve a new account. If a customer is using the system and they log in successfully with their ID then they can choose to view their balances which will display all of their own accounts. If they choose to deposit money into an account they will be asked how much they want to add (non negative). If they choose to withdraw money from an account they will be asked how much and they cannot take out more money than there is in the account (or take out a negative number). If they choose to create a new account they will be asked what type of account they want (checking or savings). If they choose to edit their account they can change their name in the account or what the type of account it is. Some concepts from the course we used include Strings, Arrays, File Handling, Version Control (Github), Object Oriented Programming, Classes, Inheritance (private, public), Dynamic Binding, Abstract Base Classes, UML, and Exception Handling.

Use Case Diagram

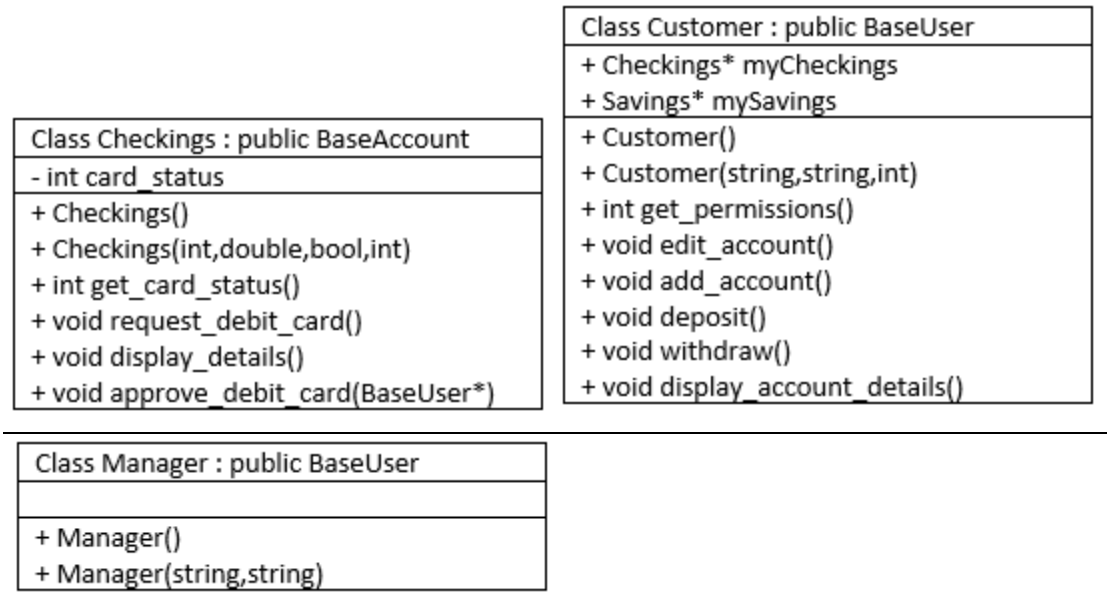


Class error
- string msg
+ error() + void display()

Class BaseUser
int ID # string fname # string lname
+ BaseUser() + BaseUser(string,string,int) + virtual int get_permissions() = 0 + int getID() + string getFName() + string getLName()

Class BaseAccount
int act_num # double balance # string type + bool approved
+ int deposit(double) + int withdraw(double) + void edit() + string get_type() + int get_act_num() + double get_balance() + void approve(BaseUser*)

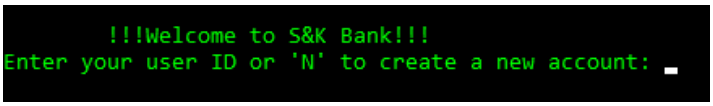
Class Savings : public BaseAccount
- double interest_rate - int transactions_left - int mon
- void calc_limit() + Savings() + Savings(int,double,bool,int,double,int) + int get_mon() + int get_trans() + double get_int_rate() + int withdraw(double) + deposit(double) + void display_details()



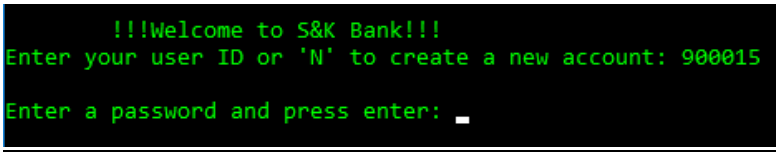
Experiments and Results

Functionality was extensively tested before being implemented into the main program, mainly involving writing to and reading from files. We tested the final project multiple times in order to make sure it was working properly however the final system is not as robust as it could be. Often times the system may crash due to unexpected character inputs when a number should be input instead. Below are some examples of the system in action.

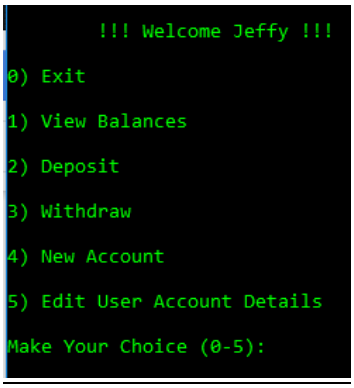
Intro message and login screen



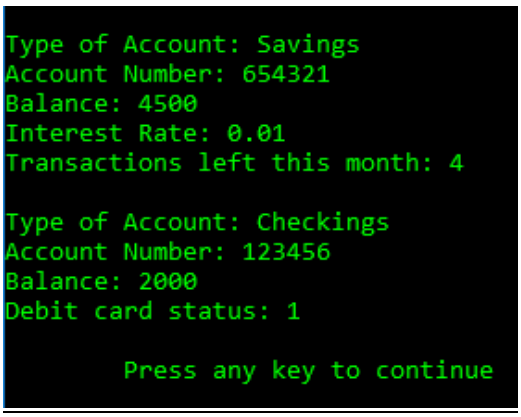
Entering with an existing user ID and password (hidden)



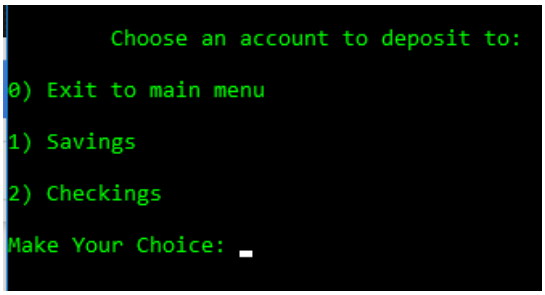
Main Menu



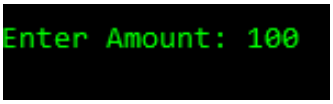
View Balances of accounts



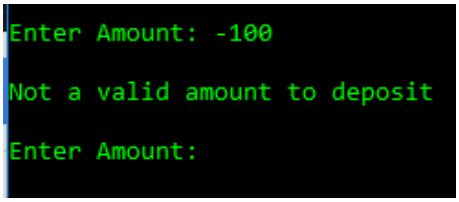
Deposit



Depositing to Savings account



Error message when trying to deposit a negative number (will crash if you enter a character)



Exit screen for Deposit

```
Leaving deposit

        Press any key to continue
```

Withdraw

```
       Choose an account to withdraw from:

0) Exit to main menu

1) Savings

2) Checkings

Make Your Choice:
```

Withdrawing from Checkings

```
Enter Amount: 500
```

Checkings now has $1500, error check for overdrawing (again will not work with characters)

```
Enter Amount: 1501

Attempt to overdraw

Enter Amount: 1500
```

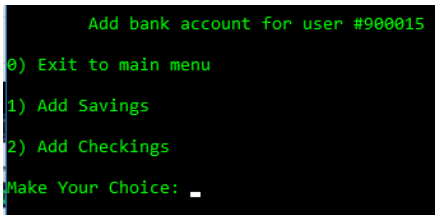
Exit screen for withdraw

```
Leaving withdraw

        Press any key to continue
```

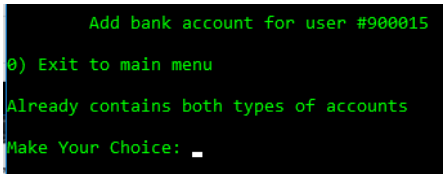
Balance after Withdrawing all money from the Checkings Account

```
Type of Account: Savings
Account Number: 654321
Balance: 4500
Interest Rate: 0.01
Transactions left this month: 4

Type of Account: Checkings
Account Number: 123456
Balance: 0
Debit card status: 1

        Press any key to continue
```

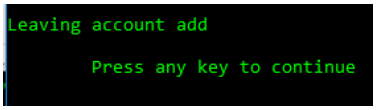
New Account



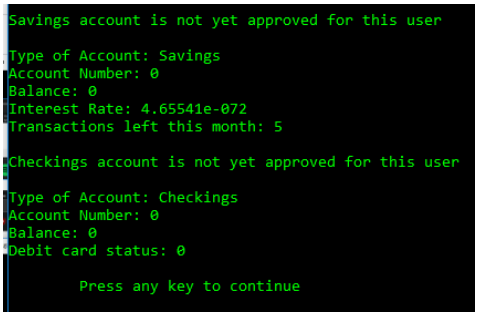
After adding both types of accounts



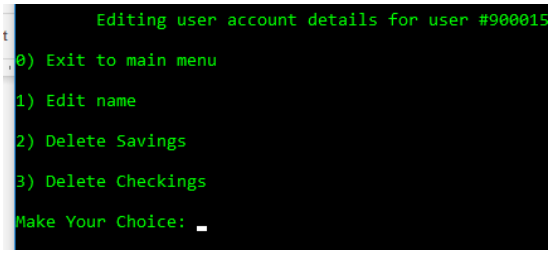
Exiting message for New Account



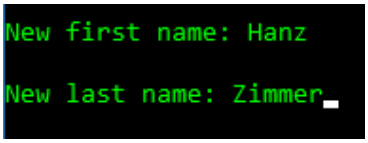
New Accounts



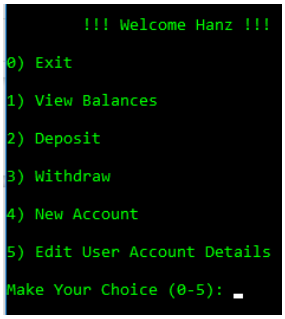
Editing Account Details



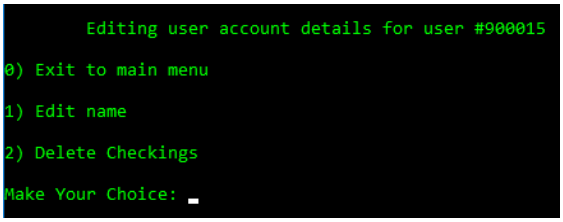
Editing Name



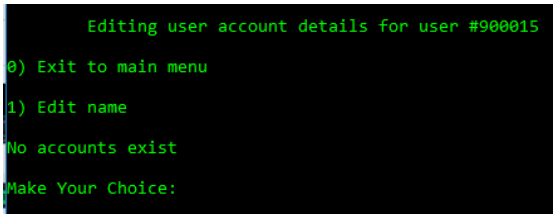
New Name



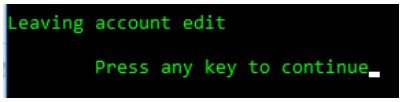
After Deleting Savings



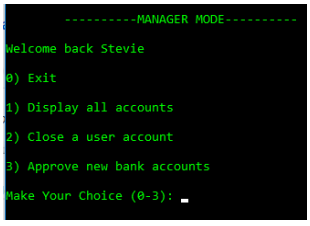
After Deleting Checkings



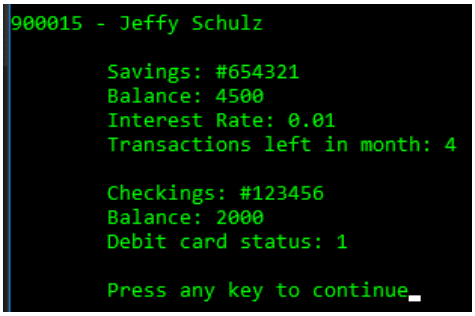
Exit Message



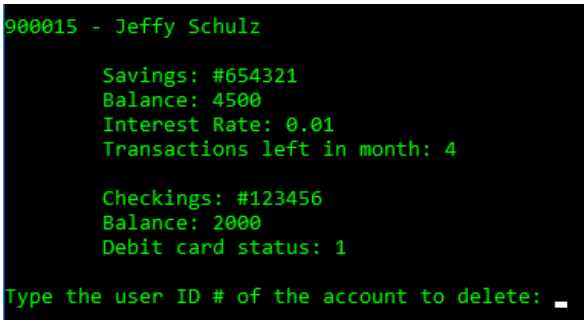
Manager Mode Menu



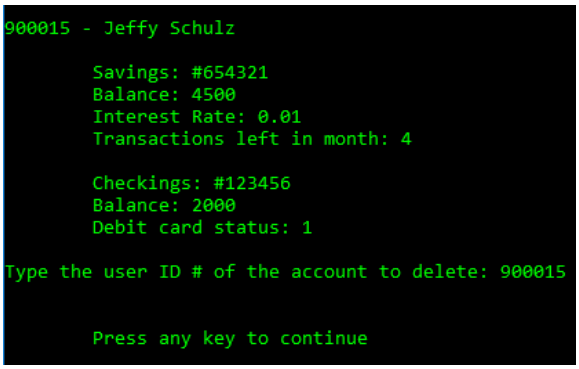
All User Accounts Displayed



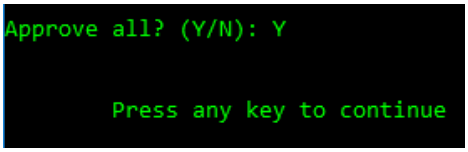
Closing a User Account (program will crash if an invalid input is chosen)



Closing Jeffy's Account



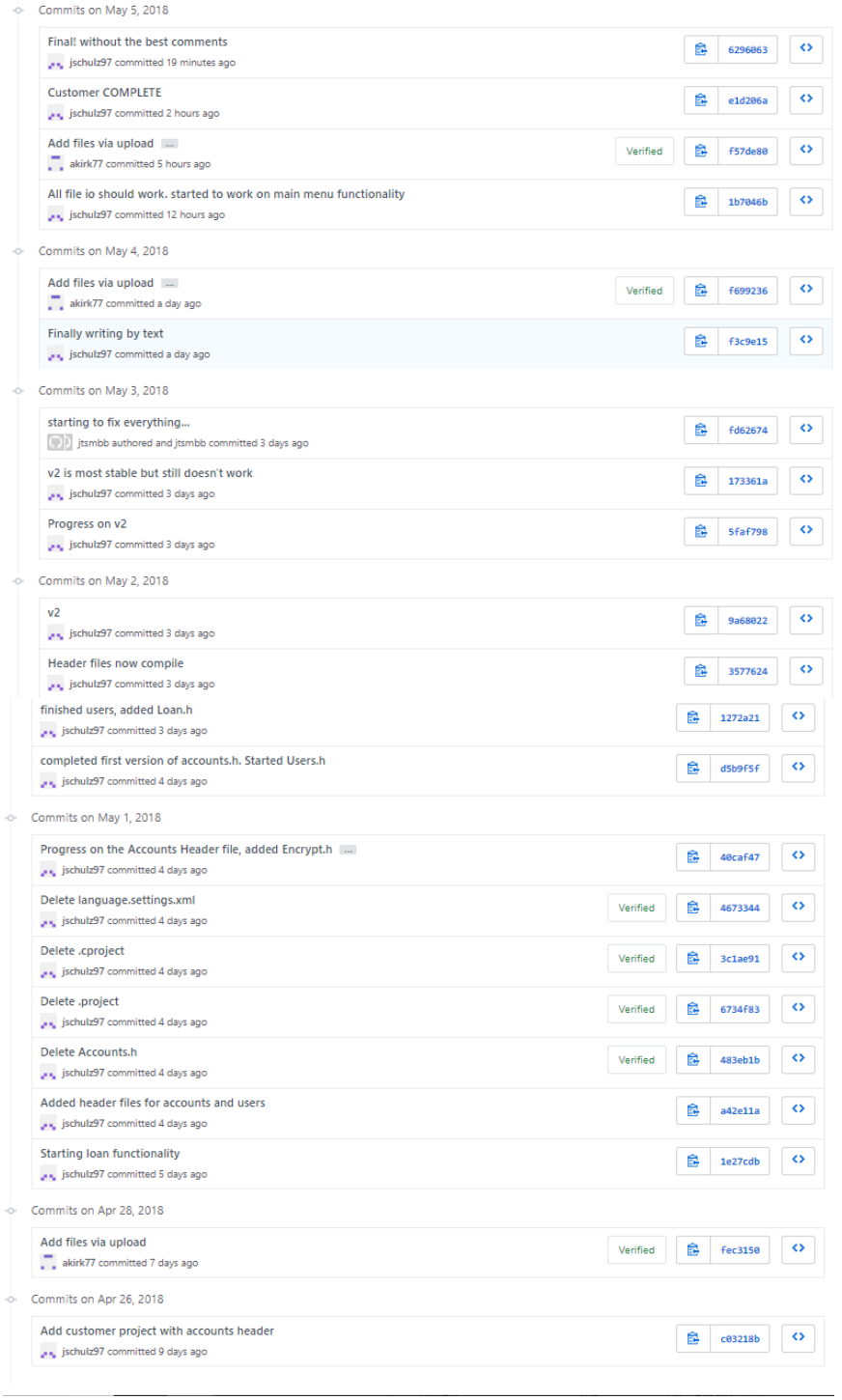
Approving Accounts



Exit Message

```
Press any key to continue_
```

Commits



Discussion and Conclusions

The final project was not exactly as it was first proposed. For example we do not have a method for users to apply for loans or to transfer money between accounts. We were able to make the basics of our system work well and were able to get interest on savings accounts working. Having encrypted passwords was not included in our original proposal but it adds a necessary security feature that we overlooked initially. With the way our final program turned out the results we got were as expected. As far as issues are concerned our biggest problem was actually writing to and reading from the file where we kept the accounts. Initially we used binary files to save memory but it became difficult to figure out what our problems were so we changed to text files. Even then we had issues getting each out on different lines but afterwards the methods were not too difficult to implement. An important thing we learned from this is the importance of time management considering how late we started on the project. If we had more time we could have implemented methods for loans and included more error checking for deposit and withdrawal methods. Overall this project gave us a good way to practice our C++ programming skills that we had worked on throughout the semester and taught us a lot about how all the topics discussed in lecture and in labs can come together in a comprehensive project.

Appendices

Includes Source Code, Encrypt.h, and Users&Accounts.h

Source Code

```
#include<iostream>
#include<fstream>
#include<cctype>
#include<iomanip>
#include<string>
#include<conio.h> //For hiding password with getch()
#include<cmath>

#include "Users&Accounts.h"
#include "Encrypt.h"
//#include "Loan.h"
using namespace std;

/*************************************************************************************
**********************/


/*************************************************************************************
**********************/
```

```
string enter_password();
void create_user_account();
void display_balances(BaseUser*);
BaseUser* sign_in(int,string);
void write_user_to_file(BaseUser*);
BaseUser* get_user_from_file(int id);
bool delete_user_account(BaseUser*);
void display_all_accounts();
void approve_pending_accounts(BaseUser*);

int main()
{
	system("cls");
	cout << endl << "\t!!! Welcome to S&K Bank !!!";
	BaseUser* usr = new Customer();

	//Log in loop
	do {
		try {
			cout << endl << "Enter your user ID or 'N' to create a new account: ";
			string id;
			cin >> id;
			if(id == "N" || id == "n") {
				create_user_account();
			} else {
				//Check if id is fully a number
				int idnum = 0;
				for(auto i : id) {
					if(!isdigit(i))
						throw error("Invalid user ID");
				}

				idnum = stoi(id);

				string password = enter_password();

				usr = sign_in(idnum,password);
			}
		} catch(error e) {
```

```
                e.display();
        }

} while(usr->getID() == 1); //default value for new BaseUser


if(usr->getID() != 1) {
        char choice;
        int num;
        do {
                if(usr->get_permissions() == 2) { //Menu for Bank Manager
                        Manager *user = static_cast<Manager*>(usr);
                        system("cls");
                        cout << endl << endl << "\t----------MANAGER MODE----------";
                        cout << endl << "\nWelcome back " << user->getFName();
                        cout << endl << endl << "0) Exit";
                        cout << endl << endl << "1) Display all accounts";
                        cout << endl << endl << "2) Close a user account";
                        cout << endl << endl << "3) Approve new bank accounts";
                        cout << endl << endl << "Make Your Choice (0-3): ";
                        cin >> choice;
                        system("cls");
                        switch(choice)
                        {
                        case '1':
                                display_all_accounts();
                                break;
                        case '2':
                                display_all_accounts();
                                cout << "\n\nType the user ID # of the account to delete: ";
                                int num;
                                cin >> num;
                                delete_user_account(get_user_from_file(num));
                                break;
                        case '3':
                                approve_pending_accounts(user);
                                break;
                        case '0':
                                break;
                         default:
```

```
                cout << "\a\nIncorrect Input\n";
            }
            cout << "\n\n\tPress any key to continue";
            cin.ignore();
            cin.get();
            system("cls");

        } else { //Menu for regular customers
            Customer *user = static_cast<Customer*>(usr);
            system("cls");
            cout << endl << "\t!!! Welcome " << user->getFName() << " !!!";
            cout << endl << endl << "0) Exit";
            cout << endl << endl << "1) View Balances";
            cout << endl << endl << "2) Deposit";
            cout << endl << endl << "3) Withdraw";
            cout << endl << endl << "4) New Account";
            cout << endl << endl << "5) Edit User Account Details";
            cout << endl << endl << "Make Your Choice (0-5): ";
            cin >> choice;
            system("cls");
            switch(choice)
            {
            case '1': //View account balances
                user->display_account_details();
                break;
            case '2': //Deposit
                user->deposit();
                write_user_to_file(user);
                break;
            case '3': //Withdraw
                user->withdraw();
                write_user_to_file(user);
                break;
            case '4': //Create a new savings/checkings account (only if the user
does not already have one or both)
                user->add_account();
                write_user_to_file(user);
                break;
            case '5': //Edit account details
                user->edit_account();
```

```
                                        write_user_to_file(user);
                                        break;
                                case '0':
                                        break;
                                 default:
                                         cout << "\a\nIncorrect Input\n";
                                }
                                cout << "\n\n\tPress any key to continue";
                                cin.ignore();
                                cin.get();
                                system("cls");
                        }

                }
                while(choice != '0');
                cout << "\nThanks for banking with S&K!\n\n";
        }
        return 0;
}



/**
 * Sign in a user using the logins file. Returns the user object pointer.
 */
BaseUser* sign_in(int usrID, string encrypted_pass) {
        //Open logins.dat
        ifstream loginFile;
        loginFile.open("logins.dat");
        if(!loginFile)
        {
                cout << "Something went wrong, logins.dat couldn't be opened in sign_in(). Press
any key to continue.";
                return new Customer();
        }

        //See if password matches one in file
        bool found = false;
        string word;
        while(loginFile >> word) {
```

```
			if(stoi(word) == usrID) {
				found = true;
				//cout << endl << "AA" << log->getPass() << "AA";
				string p;
				loginFile >> p;
				if(encrypted_pass != p) {
					cerr << "\nPassword is incorrect\n";
					return new Customer();
				}
			}
			string temp;
			getline(loginFile,temp);
		}
		loginFile.close();

		if(!found) {
			cout << "\nUser does not exist. Press any key to continue.";
			return new Customer();
		}


		//Find user data, return user object pointer
		return get_user_from_file(usrID);
}



/**
 * Gets next available ID for users by analyzing the logins file
 */
int getNextID() {
		ifstream inFile;
		inFile.open("logins.dat");
		if(!inFile)
		{
			cout << "Something went wrong, logins.dat couldn't be opened. Press any key to
continue.";
			inFile.close();
			return 0;
		}
```

```
	//Lowest ID is 900000
	int nextID = 900000;

	//Find next ID by taking highest + 1
	string word;
	while(inFile >> word) {
		if(stoi(word) >= nextID) {
			nextID = stoi(word) + 1;
		}
		string temp;
		getline(inFile,temp);
	}

	inFile.close();
	return nextID;
}


/**
 * Allows the user to enter password without it appearing in the terminal
 */
string enter_password() {
	cout << "\nEnter a password and press enter: ";
	char c;
	string pass = "";

	while((c = getch()) != '\r') {
		pass += c;
	}
	//cout << endl << "AA" << pass << "AA";
	pass = encrypt(pass);
	//cout << endl << "AA" << pass << "AA";
	return pass;
}



/**
 * Creates user account, writes user object to users file, and creates their login
```

```
 */
void create_user_account() {
	cout << endl << "\t**Create a new user account**";
	cout << endl << "Enter First Name: ";
	string fname;
	cin >> fname;

	cout << endl << endl << "Enter Last Name: ";
	string lname;
	cin >> lname;

	//Use helper function to figure out next available ID
	int id;
	if(!(id = getNextID())) {
		cout << "\nUser creation failed in create_user_account() & getNextID(). Press any
key to continue.\n";
		return;
	}
	cout << "\nYour user ID will be: " << id;

	//Make new user - will always be a customer
	BaseUser* newUser = new Customer(fname,lname,id);

	//Self explanatory
	write_user_to_file(newUser);

	//Create and save password
	string pass = enter_password();

	//Save login object to file
	ofstream outFile;
	outFile.open("logins.dat",ios::app);
	if(!outFile)
	{
		cout << "Something went wrong, logins.dat couldn't be opened in
create_user_account(). Press any key to continue.";
		return;
	}
	outFile << endl << id << " " << pass;
	outFile.close();
```

```
	cout << endl << "\n\tAccount Created. User ID: " << id;
}


/**
 * Displays the balances of the current user's accounts
 */
void display_balances(BaseUser* usr) {
	Customer* cust = static_cast<Customer*>(usr);
	if(!(cust->myCheckings == NULL)) {
		cust->myCheckings->display_details();
	}
	if(!(cust->mySavings == NULL)) {
		cust->mySavings->display_details();
	}
}


/**
 * Writes a user object to the users file
 */
void write_user_to_file(BaseUser* usr) {
	delete_user_account(usr);
	ofstream of("users.dat",ios::app);
	if(!of) {cerr << "\nCould not open file users.dat in write_user_to_file()";};

	if(usr->getID() == 100000) {
		ofstream of("users.dat",ios::app);

	}
	else {
		//cout << "\nInside customer print";
		Customer *user = static_cast<Customer*>(usr);

		of << user->getID() << " " << user->getFName() << " " << user->getLName();
		if(user->mySavings != NULL) {
			//cout << "\nPrinting Savings";
			of << " " <<  user->mySavings->get_type() << " " <<  user->mySavings-
>get_act_num() << " "
```

```
                                << user->mySavings->get_balance() << " " << user->mySavings->approved << " "
                                << user->mySavings->get_mon() << " " << user->mySavings->get_int_rate() << " "
                                << user->mySavings->get_trans();
                }
                if(user->myCheckings != NULL) {
                        //cout << "\nPrinting Checkings";
                        of << " " << user->myCheckings->get_type() << " " << user->myCheckings->get_act_num() << " "
                                << user->myCheckings->get_balance() << " " << user->myCheckings->approved
                                << " " << user->myCheckings->get_card_status();
                }
                of << endl;
        }
        of.close();
}


/**
 * Retrieves user object from the users file
 */
BaseUser* get_user_from_file(int id) {
        ifstream is("users.dat");
        if(!is) {cout << "\nCould not open file users.dat in get_user_from_file()";};
        string input;
        while(is >> input) {
                //cout << "\nCheck if: " << endl << input << endl << to_string(id);
                if(input == to_string(id)) {
                        if(input == "100000") {
                                string fn,ln;
                                is >> fn;
                                is >> ln;
                                Manager *man = new Manager(fn,ln);
                                is.close();
                                return man;
                        } else {
                                int id,act,mon,tr,card;
                                string fn,ln;
```

```
                                double bal,inter;
                                bool app;
                                is >> fn >> ln;
                                id = stoi(input);
                                Customer *c = new Customer(fn,ln,id);

                                string temp;
                                is >> temp;
                                if(temp == "Savings") {
                                        is >> act >> bal >> app >> mon >> inter >> tr;
                                        c->mySavings = new Savings(act,bal,app,mon,inter,tr);
                                        is >> temp;
                                }
                                if(temp == "Checkings") {
                                        is >> act >> bal >> app >> card;
                                        c->myCheckings = new Checkings(act,bal,app,card);
                                }
                                is.close();
                                return c;
                        }
                } else {
                        //advance file pointer to next line
                        string temp;
                        /*do {
                                is >> temp;
                        } while(temp != "|");*/
                        getline(is,temp);
                        //cout << "\nThis is getline: " <<  temp;
                }
        }
        cout << "\nUnable to get user from file";
        is.close();
        return NULL;
}


/**
 * Deletes a specific user from the records
 */
bool delete_user_account(BaseUser* user) //delete an account
```

```
{
	ifstream inFile;
	ofstream outFile;
	inFile.open("users.dat");
	if(!inFile)
	{
		cout << "Something went wrong, the File users.dat couldn't be opened. Press any
key to continue";
		return false;
	}
	outFile.open("temp.dat");
	inFile.seekg(0,ios::beg);
	string input;
	bool del = false;
	while(inFile >> input)
	{
		if(input == to_string(user->getID()))
		{
			string temp;
			getline(inFile,temp);
			del = true;
		} else {
			string line;
			getline(inFile,line);
			outFile << input + line << endl;
		}
	}
	inFile.close();
	outFile.close();
	remove("users.dat");
	rename("temp.dat", "users.dat");

	inFile.open("logins.dat");
	if(!inFile)
	{
		cout << "Something went wrong, the File users.dat couldn't be opened. Press any
key to continue";
		return false;
	}
	outFile.open("temp.dat");
```

```
	inFile.seekg(0,ios::beg);

	while(inFile >> input)
	{
		if(input == to_string(user->getID()))
		{
			string temp;
			getline(inFile,temp);
			del = true;
		} else {
			string line;
			getline(inFile,line);
			outFile << input + line << endl;
		}
	}
	inFile.close();
	outFile.close();
	remove("logins.dat");
	rename("temp.dat", "logins.dat");

	if(del){
		return true;
	} else {
		return false;
	}
}


/**
 * yes
 */
void display_all_accounts() {
	ifstream logins("logins.dat");
	Customer *cust = new Customer();

	string word;
	while(logins >> word) {
		if(stoi(word) != 100000) {
			cust = static_cast<Customer*>(get_user_from_file(stoi(word)));
```

```
                        cout << endl << endl << cust->getID() << " - " << cust->getFName() << "
" << cust->getLName();
                        if(cust->mySavings != NULL) {
                                cout << endl << endl << "\tSavings: #" <<  cust->mySavings-
>get_act_num()
                                        << "\n\tBalance: " << cust->mySavings->get_balance()
                                        << "\n\tInterest Rate: " << cust->mySavings-
>get_int_rate()
                                        << "\n\tTransactions left in month: " << cust->mySavings-
>get_trans();
                        }
                        if(cust->myCheckings != NULL) {
                                cout << endl << endl << "\tCheckings: #" <<  cust->myCheckings-
>get_act_num()
                                        << "\n\tBalance: " << cust->myCheckings->get_balance()
                                        << "\n\tDebit card status: " << cust->myCheckings-
>get_card_status();
                        }
                }
                string temp;
                getline(logins,temp);
        }

        logins.close();
}


/**
 *
 */
void approve_pending_accounts(BaseUser *manager) {
        ifstream logins("logins.dat");
        Customer *cust = new Customer();
        vector<BaseUser*> savingsVec;
        vector<BaseUser*> checkingsVec;

        string word;
        while(logins >> word) {
                if(stoi(word) != 100000) {
                        cust = static_cast<Customer*>(get_user_from_file(stoi(word)));
```

```
                        if(cust->mySavings != NULL) {
                                if(!(cust->mySavings->approved)) {
                                        cout << endl << endl << cust->getID() << " - " << cust-
>getFName() << " " << cust->getLName();
                                        savingsVec.push_back(cust);
                                        cout << endl << "\tSavings";
                                }
                        }
                        if(cust->myCheckings != NULL) {
                                if(!(cust->myCheckings->approved)) {
                                        cout << endl << endl << cust->getID() << " - " << cust-
>getFName() << " " << cust->getLName();
                                        checkingsVec.push_back(cust);
                                        cout << endl << "\tCheckings";
                                }
                        }
                }
                string temp;
                getline(logins,temp);
        }

        cout << "\n\nApprove all? (Y/N): ";
        char c;
        cin >> c;
        if(c == 'y' || c == 'Y') {
                for(auto i : savingsVec) {
                        static_cast<Customer*>(i)->mySavings->approve(manager);
                        write_user_to_file(i);
                }
                for(auto i : checkingsVec) {
                        static_cast<Customer*>(i)->myCheckings->approve(manager);
                        write_user_to_file(i);
                }
        }


        logins.close();
}
```

Encrypt.h

```
/**
 *
 */
std::string encrypt(std::string m) {
	int count = 0, keycount = 0;
	std::string key = "banking";
	for(auto i : m) {
		if(keycount == 7)
			keycount = 0;
		m[count] = i+key[keycount];
		while(m[count] < 33) {
			m[count]+=93;
		}
		while(m[count] > 126) {
			m[count]-=93;
		}
		count++;
		keycount++;
	}
	return m;
}
```

Users&Accounts.h

```
#include<ctime>
#include<vector>

/**
 *
 */
class error {
private:
	std::string msg;
public:
	error(std::string m) {msg = m;};
	void display() {std::cerr << std::endl << msg << std::endl;};
};
```

```
/*****************************************************************************
**********************/


/**
 *
 */
class BaseUser {
protected:
	int ID = 1;
	std::string fname;
	std::string lname;
public:
	BaseUser() {};
	BaseUser(std::string fn, std::string ln, int i) {fname = fn; lname = ln; ID = i;};
	virtual int get_permissions() = 0;
	int getID() {return ID;};
	std::string getFName() {return fname;};
	std::string getLName() {return lname;};
};


/*****************************************************************************
**********************/


/**
 *
 */
class Login {
private:
	int ID;
	std::string enc_pass;
public:
	Login() {};
	Login(int i,std::string p) {ID = i; enc_pass = p;};
	int getID() {return ID;};
	std::string getPass() {return enc_pass;};
```

```
};



/*****************************************************************************
**********************/


/**
 *
 */
class BaseAccount {
protected:
	int act_num = 0;
	double balance = 0;
	std::string type;
public:
	//BaseAccount();
	virtual void display_details() const = 0;
	bool approved = false;

	int deposit(double);
	int withdraw(double);
	void edit();
	std::string get_type() const {return type;};
	int get_act_num() const {return act_num;};
	double get_balance() const {return balance;};
	void approve(BaseUser *usr);
};


/**
 *
 */
int BaseAccount::withdraw(double x) {
	try {
		if(!approved)
			throw error("Account is not yet approved");
		if(x > balance)
			throw error("Attempt to overdraw");
```

```
		if(x <= 0)
			throw error("Not a valid amount to withdraw");
		balance -= x;
		return 1;
	} catch(error e){
		e.display();
		return 0;
	}
}


/**
 *
 */
int BaseAccount::deposit(double x) {
	try {
		if(!approved)
			throw error("Account is not yet approved");
		if(x <= 0)
			throw error("Not a valid amount to deposit");
		balance += x;
		return 1;
	} catch(error e){
		e.display();
		return 0;
	}
}


/**
 *
 */
void BaseAccount::approve(BaseUser *usr) {
	try {
		if(usr->get_permissions() < 2)
			throw error("You do not have permission to complete this task");
		approved = true;
	} catch(error e) {
		e.display();
		throw;
```

```
        }
}


/**
 *
 */
void BaseAccount::edit()
{
        std::cout << std::endl << "Account Number: " << act_num;
        std::cout << std::endl << "Edit Balance amount : ";
        std::cin >> balance;
}


/**************************************************************************************
**********************/

/**
 *
 */
class Savings : public BaseAccount {
private:
        double interest_rate;
        int transactions_left = 5;
        void calc_limit();
        int mon;
public:
        Savings();
        Savings(int,double,bool,int,double,int);
        int get_mon() {return mon ;};
        int get_trans() {return transactions_left;};
        double get_int_rate() {return interest_rate;};
        int withdraw(double);
        int deposit(double);
        void display_details() const;
};

/**
 *
```

```
 */
Savings::Savings() : BaseAccount() {
	balance = 0.0;
	type = "Savings";
	approved = false;
	interest_rate = 0.0;

	time_t theTime = time(NULL);
	struct tm *startTime = localtime(&theTime);
	mon = startTime->tm_mon;
}


/**
 *
 */
Savings::Savings(int act,double x,bool app,int m,double inter,int tr) : BaseAccount() {
	act_num = act;
	balance = x;
	type = "Savings";
	approved = app;
	mon = m;
	interest_rate = inter;
	transactions_left = tr;
}


/**
 *
 */
int Savings::withdraw(double x) {
	try {
		if(transactions_left <= 0)
			throw error("Reached Transaction Limit");
		int ret = BaseAccount::withdraw(x);
		if(ret)
			calc_limit();
		return ret;
	} catch(error e) {
		e.display();
```

```
                return 0;
        }
}


/**
 *
 */
int Savings::deposit(double x) {
        try {
                if(transactions_left <= 0)
                        throw error("Reached Transaction Limit");
                int ret = BaseAccount::deposit(x);
                if(ret)
                        calc_limit();
                return ret;
        } catch(error e) {
                e.display();
                return 0;
        }
}


/**
 *
 */
void Savings::display_details() const {
        try {
                if(!approved)
                        throw error("Savings account is not yet approved for this user");
                std::cout << std::endl << "Type of Account: " << type;
                std::cout << std::endl << "Account Number: " << act_num;
                std::cout << std::endl << "Balance: " << balance;
                std::cout << std::endl << "Interest Rate: " << interest_rate;
                std::cout << std::endl << "Transactions left this month: " << transactions_left <<
std::endl;
        } catch(error e) {
                e.display();
        }
}
```

```
/**
 *
 */
void Savings::calc_limit() {
	//Save previously fetched month, get current time
	int month = mon;

	time_t theTime = time(NULL);
	struct tm *aTime = localtime(&theTime);

	//If it's a different month, reset the count
	if(month != aTime->tm_mon)
		transactions_left = 4;
	else
		transactions_left--;
	std::cout << std::endl << "You have " << transactions_left << " transactions left this
month." << std::endl;
}


/**************************************************************************************
**********************/

/**
 *
 */
class Checkings : public BaseAccount {
private:
	int card_status = 0;
public:
	Checkings();
	Checkings(int,double,bool,int);
	int get_card_status() {return card_status;};
	void request_debit_card();
	void display_details() const;
	void approve_debit_card(BaseUser*);
};
```

```
/**
 *
 */
Checkings::Checkings() : BaseAccount() {
	balance = 0;
	type = "Checkings";
	approved = false;
	card_status = 0;
}


/**
 *
 */
Checkings::Checkings(int act,double x,bool app,int card) : BaseAccount() {
	act_num = act;
	balance = x;
	type = "Checkings";
	approved = app;
	card_status = card;
}


/**
 *
 */
void Checkings::request_debit_card() {
	try {
		if(!approved)
			throw error("Checkings account is not yet approved for this user");
		card_status = -1; //Card status -1 means card is waiting on approval
	} catch(error e) {
		e.display();
	}
}


/**
 *
```

```
 */
void Checkings::approve_debit_card(BaseUser *usr) {
	try {
		if(usr->get_permissions() < 2)
			throw error("You do not have permission to complete this task");
		card_status = 1;
	} catch(error e) {
		e.display();
		throw;
	}
}


/**
 *
 */
void Checkings::display_details() const {
	try {
		if(!approved)
			throw error("Checkings account is not yet approved for this user");
		std::cout << std::endl << "Type of Account: " << type;
		std::cout << std::endl << "Account Number: " << act_num;
		std::cout << std::endl << "Balance: " << balance;
		std::cout << std::endl << "Debit card status: " << card_status;
	} catch(error e) {
		e.display();
	}
}



/*************************************************************************
**********************/

/**
 * Customer, a type of BaseUser
 */
class Customer : public BaseUser {
public:
	Checkings* myCheckings = NULL;
```

```
	Savings* mySavings = NULL;
	//std::vector<BaseLoan*> loanVector;

	Customer() {};
	Customer(std::string fn, std::string ln, int i) : BaseUser(fn,ln,i) {};
	int get_permissions() {return 1;};
	void edit_account();
	void add_account();
	void deposit();
	void withdraw();
	void display_account_details();
};


/**
 * Displays all bank account details
 */
void Customer::display_account_details() {
	if(mySavings != NULL)
		mySavings->display_details();
	if(myCheckings != NULL)
		myCheckings->display_details();
}


/**
 * Handles deposit with dynamic menu
 */
void Customer::deposit() {
	bool swap = false;
	bool two = false;
	int choice = 1;
	while(choice != 0) {
		system("cls");
		std::cout << std::endl << "\tChoose an account to deposit to: ";
		std::cout << std::endl << std::endl << "0) Exit to main menu";
		if(mySavings != NULL && myCheckings == NULL) {
			std::cout << std::endl << std::endl << "1) Savings";
		} else if(mySavings == NULL && myCheckings != NULL) {
			swap = true;
```

```
        std::cout << std::endl << std::endl << "1) Checkings";
} else if(mySavings == NULL && myCheckings == NULL) {
        std::cout << "\n\nNo accounts exist";
} else {
        two = true;
        std::cout << std::endl << std::endl << "1) Savings";
        std::cout << std::endl << std::endl << "2) Checkings";
}
std::cout << std::endl << std::endl << "Make Your Choice: ";
std::cin >> choice;
system("cls");

if(swap && choice == 1) { //Correct for menu dynamics
        choice = 2;
}

switch(choice)
{
case 1: //Savings
        double amt;
        std::cout << "\nEnter Amount: ";
        std::cin >> amt;
        if(mySavings->deposit(amt))
                std::cout << "\n\tSuccessfully deposited.";
        else
                std::cout << "\n\tDeposit failed.";
        std::cin.ignore();
        std::cin.get();
        break;
case 2: //Checkings
        if(two) {
                double amt;
                std::cout << "\nEnter Amount: ";
                std::cin >> amt;
                if(myCheckings->deposit(amt))
                        std::cout << "\n\tSuccessfully deposited.";
                else
                        std::cout << "\n\tDeposit failed.";
                std::cin.ignore();
                std::cin.get();
```

```
                        } else {
                                std::cerr << std::endl << std::endl << "Incorrect Input" <<
std::endl;
                        }
                        break;
                case 0:
                        std::cout << "\n\nLeaving deposit";
                        break;
                default:
                        std::cerr << std::endl << std::endl << "Incorrect Input" << std::endl;
                        break;
                }
        }

}


/**
 * Handles withdrawal with dynamic menu
 */
 void Customer::withdraw() {
        bool swap = false;
        bool two = false;
        int choice = 1;
        while(choice != 0) {
                system("cls");
                std::cout << std::endl << "\tChoose an account to withdraw from: ";
                std::cout << std::endl << std::endl << "0) Exit to main menu";
                if(mySavings != NULL && myCheckings == NULL) {
                        std::cout << std::endl << std::endl << "1) Savings";
                } else if(mySavings == NULL && myCheckings != NULL) {
                        swap = true;
                        std::cout << std::endl << std::endl << "1) Checkings";
                } else if(mySavings == NULL && myCheckings == NULL) {
                        std::cout << "\n\nNo accounts exist";
                } else {
                        two = true;
                        std::cout << std::endl << std::endl << "1) Savings";
                        std::cout << std::endl << std::endl << "2) Checkings";
                }
```

```
		std::cout << std::endl << std::endl << "Make Your Choice: ";
		std::cin >> choice;
		system("cls");

		if(swap && choice == 1) { //Correct for menu dynamics
			choice = 2;
		}

		switch(choice)
		{
		case 1: //Savings
		double amt;

			std::cout << "\nEnter Amount: ";
			std::cin >> amt;
			if(mySavings->withdraw(amt))
				std::cout << "\n\tSuccessfully withdrew.";
			else
				std::cout << "\n\tWithdrawal failed.";
			std::cin.ignore();
			std::cin.get();
			break;
		case 2: //Checkings
			if(two) {
				double amt;

				std::cout << "\nEnter Amount: ";
				std::cin >> amt;
				if(myCheckings->withdraw(amt))
					std::cout << "\n\tSuccessfully withdrew.";
				else
					std::cout << "\n\tWithdrawal failed.";
				std::cin.ignore();
				std::cin.get();
			} else {
				std::cerr << std::endl << std::endl << "Incorrect Input" <<
std::endl;
			}
			break;
		case 0:
```

```
                std::cout << "\n\nLeaving withdraw";
                break;
            default:
                std::cerr << std::endl << std::endl << "Incorrect Input" << std::endl;
                break;
            }
        }
}


/**
 * Edits account details, including name and savings/checkings accounts
 */
void Customer::edit_account() {
        bool swap = false;
        bool three = false;
        int choice = 1;
        while(choice != 0) {
            system("cls");
            std::cout << std::endl << "\tEditing user account details for user #" << ID;
            std::cout << std::endl << std::endl << "0) Exit to main menu";
            std::cout << std::endl << std::endl << "1) Edit name";
            if(mySavings != NULL && myCheckings == NULL) {
                std::cout << std::endl << std::endl << "2) Delete Savings";
            } else if(mySavings == NULL && myCheckings != NULL) {
                swap = true;
                std::cout << std::endl << std::endl << "2) Delete Checkings";
            } else if(mySavings == NULL && myCheckings == NULL) {
                std::cout << "\n\nNo accounts exist";
            } else {
                three = true;
                std::cout << std::endl << std::endl << "2) Delete Savings";
                std::cout << std::endl << std::endl << "3) Delete Checkings";
            }
            std::cout << std::endl << std::endl << "Make Your Choice: ";
            std::cin >> choice;
            system("cls");

            if(swap && choice == 2) { //Correct for menu dynamics
                choice = 3;
```

```
                }

                switch(choice)
                {
                case 1: //Edit name
                        std::cout << "\nNew first name: ";
                        std::cin >> fname;
                        std::cout << "\nNew last name: ";
                        std::cin >> lname;
                        break;
                case 2: //Delete Savings
                        mySavings = NULL;
                        std::cout << "\n\tSavings account deleted";
                        std::cin.ignore();
                                std::cin.get();
                        break;
                case 3: //Delete Checkings
                        if(three) {
                                myCheckings = NULL;
                                std::cout << "\n\tCheckings account deleted";
                                std::cin.ignore();
                                std::cin.get();
                        } else {
                                std::cerr << std::endl << std::endl << "Incorrect Input" <<
std::endl;
                        }
                        break;
                case 0:
                        std::cout << "\n\nLeaving account edit";
                        break;
                default:
                        std::cerr << std::endl << std::endl << "Incorrect Input" << std::endl;
                        break;
                }
        }
}


/**
 * Adds bank account to user with dynamic menu
```

```
 */
void Customer::add_account() {
	bool swap = false;
	bool two = false;
	int choice = 1;
	while(choice != 0) {
		system("cls");
		std::cout << std::endl << "\tAdd bank account for user #" << ID;
		std::cout << std::endl << std::endl << "0) Exit to main menu";
		if(mySavings == NULL && myCheckings != NULL) {
			std::cout << std::endl << std::endl << "1) Add Savings";
		} else if(mySavings != NULL && myCheckings == NULL) {
			swap = true;
			std::cout << std::endl << std::endl << "1) Add Checkings";
		} else if(mySavings != NULL && myCheckings != NULL) {
			std::cout << "\n\nAlready contains both types of accounts";
		} else {
			two = true;
			std::cout << std::endl << std::endl << "1) Add Savings";
			std::cout << std::endl << std::endl << "2) Add Checkings";
		}
		std::cout << std::endl << std::endl << "Make Your Choice: ";
		std::cin >> choice;
		system("cls");

		if(swap && choice == 1) { //Correct for menu dynamics
			choice = 2;
		}

		switch(choice)
		{
		case 1: //Add Savings
			mySavings = new Savings();
			std::cout << "\n\tSavings account created, waiting on approval of bank
manager.";
			std::cin.ignore();
				std::cin.get();
			break;
		case 2: //Add Checkings
			if(two) {
```

```
                                myCheckings = new Checkings();
                                std::cout << "\n\tCheckings account created, waiting on approval
of bank manager.";
                                std::cin.ignore();
                                std::cin.get();
                        } else {
                                std::cerr << std::endl << std::endl << "Incorrect Input" <<
std::endl;
                        }
                        break;
                case 0:
                        std::cout << "\n\nLeaving account add";
                        break;
                default:
                        std::cerr << std::endl << std::endl << "Incorrect Input" << std::endl;
                        break;
                }
        }
}


/**
 * Manager, a type of BaseUser. Contains no accounts but has admin access
 */
class Manager : public BaseUser {
public:
        Manager() {};
        Manager(std::string fn, std::string ln) : BaseUser(fn,ln,0) {ID = 100000;};
        int get_permissions() {return 2;};
};
```