README

CALEB SMITH

1. Introduction

This is a `C#` code sample. It is a console application which allows the user to perform basic banking functions. It has the following capabilities:

- Create a new user
- Login
- Create a new bank account
- Deposit into an account
- Withdraw from an account
- Check the balance of an account
- View the transaction history of an account
- Log out

2. Concise User guide

Push buttons!

More seriously, the console interface for this application is fairly self explanatory. Upon opening the application, a menu is presented to the user, consisting of several lines, each beginning with a number and then a description of the option. Keying the appropriate number and then pressing ENTER will perform the indicated action.

Upon first starting the program, the only options available to the user are to create a new username or to quit. After creating a username, more options become available. Upon logging in with a created username, actual banking actions can be taken.

3. Explanation of Program Structure

There are three main classes which perform the work of this application: BankingUI, Account, and User.

Date: October 2, 2018.

3.1. **The BankingUI class.** BankingUI is the class which is responsible for outputting messages to the user of the application, receiving input from the user and passing the input along to the Account and User classes which perform the actual banking operations on them. It contains the Main() function.

The state of the program is stored in the fields currentUser and users. The first field tracks the user who is currently logged in, and is set to null if there is no user currently logged in. The second is a list which stores all the User instances so far created and thus all the Account instances which they own.

The execution of the program largely occurs in the function ManageInput() which prints a menu of options to the user and then hands off execution to the appropriate function depending on the option chosen.

The other methods in this class are largely either simple helper functions or clients of functions in the User class, serving as a layer between the end user and that class.

3.2. **The User class.** This class encapsulates a user of a banking system. Each instance has a list of accounts which it owns. Again, most of its methods are a thin layer over the Account class. The class is structured this way to make it as difficult as possible to actually gain access to sensitive details about the accounts owned by an instance. Relevant to this idea is the boolean field allowAccess. This field is set to false by default, and can only be changed by the Login() and Logout() functions. If this field is set to false, most of the public facing methods of this class will not perform any function and throw an exception.

3.3. **The Account class.** This class encapsulates a bank account. It has fields to store the balance of the account, past transactions, and to store the hashed password of the User instance that owns the account. All the public facing methods of the class have a passwordHash parameter, which is checked against the stored private field to ensure that only proper users have access to sensitive methods.

Currently, the methods which have access to the internals of the Account class are public (e.g. the Withdraw() method. Ideally, we would want *only* the User class to be able to access these methods; we do not really want other classes or methods playing around with an Account. Since friend classes are not a thing in `C#`, the only way I am aware of doing this is to declare these members protected and then have the User class inherit from Account. Conceptually, this isn't exactly what we want (a user is not a type of account after all), but it would mostly get the job done. This would allow us to do away

with the passwordHash parameter used to validate access to Account. I am not sure which method is actually better, but in my current implementation I used the parameter passing method, mostly because that was the method I thought of first and it seemed to be the simpler method.