Project 2 Design

1. Analysis (Use Cases)
2. System should display a login screen when idle. If a user wishes to log in, they must type the username and password for their account. If they do not match, then user cannot log in (and displays error that one is wrong).
3. If a system admin logs in, they should have more options than a normal branch member. They will have the ability to display, add, and delete branch members.
   1. To add a branch member, they must create the username/password.
   2. To delete branch members, they must enter the username, and confirm the deletion. If the username doesn’t exist, an error is given.
4. Any employee should have the option to change their password after logging in. They must first validate their current password, and then enter the new one.
5. Any employee should be able to access and manage client data.
   1. Client data consist of: name, address, social security number, employer, income, and accounts (if the client has any).
   2. Accounts can only be added to existing clients. Accounts consist of the account number, the type, and the balance.
   3. Client info can be edited, except the name. Changes must be confirmed before they are made.
   4. Accounts can be managed. The user can withdraw or deposit.
6. All client info, account info, and employee info, are saved to pre-designated files.
7. If quitting the system, All info must be properly saved.
8. Design

**EmployeeManager Class**: The class that handles basic employees and their functionality

* **Variables:**
  + **Employee struct:** contains username, password, and job level
  + **Private Employee employeeList[]:** list of employees
  + **Private string employeeFile:** file for employees
* **Functions**
  + **EmployeeManager(string filename):** creates an employee manager using the employees found in the file given.
  + **Private void loadEmployees():** update employee list with employee file
  + **Private void saveEmployees():** save employee list in employee file.
  + **Public bool verifyLogin(string username, string password):** verifies login information
  + **Public bool changePassword(string username, string oldPass, string newPass):** Verifies password of employee, then changes to new password.
  + **Public bool isAdmin(string username):** verifies employee job level
  + **Public bool addEmployee(string username, string password, bool isAdmin):** Creates an employee. Returns true if it works, false if employee exists.
  + **Public bool deleteEmployee(string username):** Deletes an employee. Returns true if successful, false if employee does not exist.
  + **Public void displayEmployees():** Displays employee list. May overload << instead.
* **Class relation:** This class does not rely on any other classes. It is used by the System class to create and use employees.

**ClientManager Class**: The class that handles clients and their functionalities

* **Variables:**
  + **Client struct:** contains name, address, social security number, employer, and a list of account numbers.
  + **Private Client clientList[]:** list of clients
  + **Private string clientFile:** file for clients
* **Functions**
  + **clientManager(string filename):** creates a client manager using the client found in the file given.
  + **Private void loadClients():** update client list with client file
  + **Private void saveClients():** save client list in client file.
  + **Public bool addClient():** Creates a client. Returns true if it works, false if client exists already.
  + **Public bool editClient(string name):** Finds client and displays prompts to edit client information. Returns true if successful, false if client does not exist.
  + **Public bool addAccount(string name, int actNum):** Adds an account to the client account list. Returns false if client does not exist or if account number is not valid.
* **Class relation:** This class does not rely on any other classes. It is used by the System class to create and use clients

**AccountManager Class**: The class that handles accounts and their functionalities

* **Variables:**
  + **Account struct:** contains account number, client name, account type, and balance
  + **Private Account accountList[]:** list of accounts
  + **Private string accountFile:** file for accounts
* **Functions**
  + **accountManager(string filename):** creates an account manager using the accounts found in the file given.
  + **Private void loadAccounts():** update account list with account file
  + **Private void saveAccounts():** save account list in account file.
  + **Public bool addAccount(string name, int accountNum):** Creates an account. Returns true if it works, false if account exists already.
  + **Public bool manageAccount(int accountNum):** Finds account and displays prompts to withdraw or deposit. True if works, false if account does not exist.
* **Class relation:** This class does not rely on any other classes. It is used by the System class to create and use accounts

**System Class:** Handles user input and overall system usage.

* **Variables:**
  + **Private EmployeeManager employeeManager:** Manages employees
  + **Private ClientManager clientManager:** Manages clients
  + **Private AccountManager accountManager:** Manages accounts
  + **Private static const string EMPLOYEE\_FILE:** name of the employee file
  + **Private static const string CLIENT\_FILE:** name of client file
  + **Private static const string ACCOUNT\_FILE:** name of account file
* **Functions**
  + **Public void idle():** When idle, display log in screen.
  + **Public void branchScreen():** Shows the screen for branch employees.
  + **Public void SysAdminScreen():** Shows the system admin screen
  + **Public void logout():** returns to log in screen.

1. **Testing**

Normal Usage:

* If wrong username/password is entered when logging in, system should display an error and return to the “idle” state.
* Branch Employees should never see the system admin screen. System admins should always see the system admin screen upon logging in.
* After adding an employee, an account, or a client, the data should immediately be saved into the corresponding file.
* Deleting an employee should result in the employee being removed from the employee list, and from the employee file.

Abnormal Usage:

* All input should be checked for valid input (integers need to be integers, etc)
* If a customer, account, or employee does not exist, appropriate errors must be displayed.
* If any file does not exist, it needs to be created.
* If the employee file is empty, it needs to be rewritten with a “default” admin.