ACCOUNT MANAGEMENT SYSTEM

NAME: KUMAR SIDDHANT (ks1306)
EMAIL ID: kumar.siddhant@rutgers.edu, ks1306@scarletmail.rutgers.edu
DATE : 12/11/2017

PROJECT URL : https://github.com/Sid231/FinalProject

The account management system is a system which imitates how the stock and bank account system work collectively in real world. In this system the user is allowed to view, buy and sell stocks and side by side view his account details, bank balance and the transaction history of his activity. The user can also wish to deposit and withdraw cash from the account management system and can check the remaining amount in his account and can also check the transaction history of his activities.

Below is a complete explanation of how this system works.

Account_Siddhant.h

In this file, we create the base class called **Account** which is responsible to hold the data related to the account of the user.

Variables defined in the class:
cashBalance : This variable holds the current balance in the user's bank account

Methods defined in the class:
Account() : This is the constructor of the account class.
~Account() : This is the destructor of the account class.
getCashBalance() : This method returns the current balance of the user's account.
setCashBalance() : This method sets the cash balance from a source.

Account_Siddhant.cpp

In this file, we implement the methods that are defined in the **Account_Siddhant.h**.

Account ::Account() : Set the balance.
Account ::~Account() : Not used as it is not required considering the motives of the project.

accountNode_Siddhant.h

In this file, we create a class named **accountNode** whose objects form the linkedlist nodes.

Variables defined in the class:
company : This variable holds the name of the company of the stock purchased/sold.
numberOfShares : This variable holds the number of stocks.
amountPerShareForSorting: This variable holds the price per stock for sorting purpose.
amountPerShare: This variable holds the price per stock.
currentPortfolioNodeVal: This variable holds the gross value of the stocks.

date : This variable holds the date.
***prev** : This pointer points to the previous node of the current object node.
***next** : This pointer points to the next node of the current object node.

Methods defined in the class:
accountNode() : This is the constructor of the class.

accountNode_Siddhant.cpp

In this file, we implement the methods that are defined in the **accountNode_Siddhant.h**.

Account ::accountNode() :Initialize the **prev** and **next** pointers as NULL.

BankAccount_Siddhant.h

In this file we create a class called **BankAccount** which inherits the **Account** class. This class is responsible for handling all the account operations which are related to the banking system like cash deposit, cash withdrawal and in the end give the history of all the transactions.

Variables defined in the class:
depositAmount: This variable holds the amount deposited to the account
withdrawalAmount: This variable holds the amount withdrawn from the account

Method Implementations:
BankAccount() : This is the constructor of the class
~BankAccount() : This is the destructor of the class.
setBalance() : This method sets the cash balance by getting the value from its source
getBalance() : This method returns the cash balance at the places where it is needed
viewBalance(): This method displays the current balance of the account.
depositCashAmount(): This method runs the algorithm after some amount is deposited into the account.
withdrawCashAmount(): This method runs the algorithm after some amount has been withdrawn from the account.
printHistory(): This method prints the history of all the banking transactions.

BankAccount_Siddhant.cpp

In this file we implement the methods defined in the **BankAccount_Siddhant.h**. Below is a brief description of what is done in each method implementations of this file.

BankAccount::BankAccount() : This method is the constructor of the class and is used to initialize the **cashBalance**, **depositAmount** and the **withdrawalAmount** variables.
BankAccount::~BankAccount() : This method is not used as it is not required considering the motives of the project.
BankAccount::setBalance(): This method gets the latest value from the file **cashBalance.txt** and sets the variable **cashBalance** of the parent class **Account**.

BankAccount::getBalance(): This method gets the cash balance value stored in the variable **cashBalance** of the parent class **Account**.
BankAccount::depositCashAmount(): This method runs the algorithm to update and store the data in the account system after a new amount enters into the account system. The timestamp of the transaction is recorded, cash balance is updated in the cashBalance.txt file and the transaction is recorded in the bank_transaction_history.txt file.
BankAccount::withdrawCashAmount(): This method runs the algorithm to update and store the data in the account system after a new amount is withdrawn from the account system. The timestamp of the transaction is recorded, cash balance is updated in the cashBalance.txt file and the transaction is recorded in the bank_transaction_history.txt file.
BankAccount::printHistory(): This method prints the recorded transaction in the bank_transaction_history.txt file in the console.

StockAccount_Siddhant.h

In this file we create a class called **StockAccount** which inherits the **Account** class. This class is responsible for handling all the operations related to stocks and shares like buying shares, selling shares, viewing shares, etc.

Variables defined in the class:
***previousPointer** : This pointer is used for LinkedList traversal operations
***tailPointer**: This pointer points to the last node of the LinkedList
***headPointer**: This pointer points to the first node of the LinkedList.
company: This variable holds the company name of the stock.
amount: This variable holds the price of the stock per share
date: this variable holds the date of the transaction
portValue: This variable holds the total amount of the portfolio
portValue_array: This variable holds the portValue recorded after each transaction.
timestamp_array: This variable holds the time recorded after each transaction.
sizeOfPortValueArray: This variable holds the size of the portValue_array.
sizeOfList: This variable holds the size of the linkedlist.
stockDataMap<string,double>: This variable holds the stock data in the format key = **company** and value = **amount**
portfolioDataMap<string,int>: This variable holds the portfolio data in the format key = **company** and value = number of stocks

Method Implementations:
StockAccount() : Constructor of the class
~StockAccount() : Destructor of the class
displayStockPrice(): This method displays the stock price of the stock asked for.
setBalance(): This method gets the updated current balance in the user's account
getBalance(): This method returns the updated current balance in the user's account
displayCurrentPortfolio(): This method displays the current portfolio
buyShares(): This method is responsible to run the algorithm to buy shares
sellShares(): This method is responsible to run the algorithm to sell shares

sortLinkedListStockData(): This method is responsible to sort the data in the linkedlist
retrieveDataToLInkedList(): This method is responsible to retrieve data from a file to the linkedlist
savePortfolioDataToFile(): This method saves the portfolio information into a file
retrievePortfolioValue(): This method retrieves the portfolio value from a file
setPortfolioValue(): This method saves the portfolio value into a file
printHistory(): This method prints the transaction history
updateSizeOfList(): This method updates the size of the linkedlist in a file
viewgraph(): This method prints the graph

StockAccount_Siddhant.cpp

In this file, we implement the methods defined in the **StockAccount_Siddhant.h** file. Below is a brief description of the implementations of the methods

StockAccount::StockAccount() : This is the constructor of the class. In this method we follow the below operations:

1. Select one file out of Result_1.txt or Result_2.txt to get the details of the stocks
2. Get the linkedList size info from the stock_list_size_data.txt file.
3. If the size is greater than zero, then get the portfolio information and the portfolio value

StockAccount::~StockAccount() : This method is the destructor and is not used as it is not required considering the motives of the project.
StockAccount::displayStockPrice(): This method is responsible to display the stock value details in the console for the stock asked for.
StockAccount::setBalance(): This method gets the updated current balance in the user's account from the source cashBalance.txt
StockAccount::getBalance(): This method returns the updated current balance in the user's account
StockAccount::displayCurrentPortfolio(): This method returns the details of the stocks that have been purchased in the past and their current values at the moment.
StockAccount::buyShares(): This method runs the algorithms to allow the user to buy shares. In this method we follow the below operations:

1. Check if the stock the customer wants to purchase is a valid stock in the stock account system
2. Check if the user has sufficient balance in the user's account to buy the stock
3. Capture the timestamp of the stock purchase
4. Update the new cash balance in the cashBalance.txt file
5. Record the transaction in stock_transaction_history.txt and bank_transaction_history.txt file
6. Update the linkedlist by adding a new node if it's a new stock or update the existing node

StockAccount::sellShares(): This method runs the algorithms to allow the user to sell shares. In this method we follow the below operations:

1. Check if the stock the customer wants to sell is the stock the user has already purchased
2. Check if the number of stocks being sold is not more than the number of stocks purchased
3. Capture the timestamp of the stock selling
4. Update the new cash balance in the cashBalance.txt file
5. Record the transaction in stock_transaction_history.txt and bank_transaction_history.txt file
6. Update the linkedlist by deleting the node if the number of shares become zero for that stock

StockAccount::sortLinkedListStockData(): This method sorts the linkedlist using the bubble sort algorithm and sort them in the descending order of the total amount (calculated using all the stocks)

StockAccount::retrieveDataToLInkedList(): In this method, the state of the portfolio is captured from the portfolioFile.txt and the data is loaded into the linkedlist for further operations
StockAccount::savePortfolioDataToFile(): In this method, the state of the portfolio is recorded in the file portfolioFile.txt
StockAccount::retrievePortfolioValue(): In this method, the portfolio amount is captured from the port_value_data.txt file
StockAccount::setPortfolioValue(): In this method, the portfolio amount is recorded in the port_value_data.txt file
StockAccount::printHistory(): In this method, all the transactions of the operations related to the stock account system is tracked and printed on the console.
StockAccount::updateSizeOfList(): In this method, the linkedlist size is updated in the stock_list_size_data.txt file.
StockAccount::viewgraph(): In this method, the graph of the portfolio value is plotted using the library of MATLAB.

Main_Siddhant.cpp

This is the main function file. This function uses the switch network for choosing between various menu options for the Stock Account and the Bank Account. The first main menu lets user select between the two accounts or exit the program. The Bank Account menu lets user deposit, withdraw, display balance or get transaction history. The menu for Stock Portfolio lets user buy, sell shares, display portfolio values, display the stock prices, or plot the variation in portfolio values at MATLAB.

Apart from the above .cpp and .h files, we used some files to record data to read them in future when the program begins again. Below is their description:

bank_transaction_history.txt: This file records all the transactions of the cash balance of the user's account
cashBalance.txt: This file keeps the updated value of the cash balance at any particular moment
port_value_data.txt: This file keeps the updated value of the portfolio amount value
portfolioFile.txt: This file stores the portfolio information which is the current stocks and the number of shares for each one of them.
stock_list_size_data.txt: This file stores the updated value of the size of the stocks purchased at any moment.
stock_transaction_history.txt: This file records all the transactions of the cash balance of the user's account

A couple of design patterns were used in this project. Below are their details:

'Template method design pattern': We used a map to store data, in which we have a key and a value. So, the template used in this program is of the type 'map' which is a behavioral design pattern which is under the category of the template method design pattern.
'Singleton design pattern': In our project we used BankAccount which has a single global instance, thus the implementation of singleton design pattern.