# 16:332:503 Project:
# Account Management System

## Project Description

Account management system that will manage stock portfolio account and bank account.

Submitted By:-
Mareesh Kumar Issar
186000297
mi251

## ACCOUNT_MAREESH.H

This header file contains the definition for the abstract base class Account.

The class Account has a private data member cash_balance, which is used by the StockAccount and BankAccount classes to keep track of the changes made to the cash_balance of the user after every transaction.

It has the following functions:-

Account()- This is the constructor for the class Account.

getName()- This is a pure virtual function which is used to make the class Account an abstract class.

print_type()- This function is used for implementing the design pattern-template.

set_cash_balance(double)- This function is used for setting the value of cash_balance.

get_cash_balance()- This function is used for getting the value of cash_balance.

## ACCOUNT_MAREESH.CPP

This file contains the definitions for the functions of the class Account.

It uses the text file bank_account_cash_balance.txt. This text file has the value of cash_balance.

It has the following functions:-

Account()- If file bank_account_cash_balance.txt is created for the first time, this function sets the value of cash_balance to $10,000 and stores it in the file. If the file bank_account_cash_balance.txt already exists, this function reads the value stored in the file and then assigns it to cash_balance.

set_cash_balance(double)- It stores the value of the function argument in the file bank_account_cash_balance.txt, which is also the current value of cash_balance.

get_cash_balance()- It reads the file bank_account_cash_balance.txt and returns the value stored in it(after converting it to double), which is the current value of cash_balance.

## BankAccount_Mareesh.h

This header file contains the definition for the class BankAccount which is derived from class Account. This class handles all the bank account related operations.

It has the following functions:-

getName()- This is function returns the type of account for display in the output.

withdraw(double)- This function handles the withdraw operation of the bank account.

deposit(double)- This function handles the deposit operation of the bank account.

get_trans_history()- This function is used for getting the transaction history for the bank account.

## BankAccount_Mareesh.cpp

This file contains the definitions for the functions of the class BankAccount.

It uses the text file bank_transaction_history.txt. This text file has the record of all the transactions made for bank account.

It has the following functions:-

withdraw(double)- The function parameter is the amount of money to withdraw. If the cash_balance of the account is not sufficient to withdraw the amount, error message is shown. Else the amount to be withdrawn is subtracted from the cash_balance and the value of cash_balance is updated and the transaction is recorded in bank_transaction_history.txt deposit(double)- The function parameter is the amount of money to deposit. The amount to be deposited is added to the cash_balance and the value of cash_balance is updated and the transaction is recorded in bank_transaction_history.txt

get_trans_history()- This function prints the data in bank_transaction_history.txt in order of transaction time.

## NODE_STOCKACCOUNT_MAREESH.H

This header file contains the definition for the class Node

The class Node mentions 4 other classes as its friends. It defines a single node for the double linked list(defined by the class StockAccount) with private data members stock_symbol, num_of_shares and price_per_share and two pointers *next and *pre. The constructor initializes the values for stock_symbol, num_of_shares and price_per_share. It also sets the *next and *pre pointers to NULL.

## SORTING_IMPLEMENTATION_MAREESH.H

This header file is used for implementing the design pattern- strategy.

It has a base class SortImpl and two derived classes SortSelection and SortBubble. The base class has a pure virtual function sort(Node *). This function has a return type of Node *. All the classes in this file are able to access Node's data members and member functions because they are friend of the class Node.

The pure virtual function is then redefined in the derived classes to implement, their respective sorting algorithms. Once, the sorting is done these functions return the *Head pointer of the sorted DLL list.

To implement the design pattern- strategy in the file StockAccount_Mareesh.cpp, the class StockAccount has a base class pointer (SortImpl *impl). After the user selects his/her sorting strategy, this pointer then points to the appropriate derived class based upon the user's choice. Then, the sorting function of that class is called, which does the required sorting.

## STOCKACCOUNT_MAREESH.H

This header file contains the definition for the class StockAccount which is derived from class Account. This class handles all the stock account related operations.

It has data members *Head(points to the first node of the doubly linked list), size(stores the number of nodes in doubly linked list) and *impl (base pointer, used for implementing design pattern-strategy)

It has the following functions:-

StockAccount()-This is the constructor for the class StockAccount.

~StockAccount()-This is the destructor for the class StockAccount.

getName()-This is function returns the type of account for display in the output.

size()- This function returns the number of nodes in the doubly linked list or the size of the list or the number of stocks present in our portfolio.

get_stock_price(string, string)- This function returns the stock price for a particular stock symbol from a randomly selected file(to mimic the fluctuations in the stock price).

print_portfolio()- This function prints the current stock portfolio in a particular format.

update_list(string)- This function updates the value of the price per share for all the stocks in our stock portfolio based upon the most recent stock price.

buy_shares(string,string,int,double)- This function handles the buying of shares operation of the stock account.

sell_shares(string,string,int,double)- This function handles the selling of shares operation of the stock account.

store_portfolio()- This function stores the current stock portfolio in the file stock_portfolio.txt. It also stores the current total portfolio value in the file total_portfolio_value.txt

get_trans_history()- This function prints the data in stock_transaction_history.txt in order of transaction time.

get_graph()- This function interfaces c++ with MATLAB to plots the variations in the total portfolio value over a period of time(using data stored in the file total_portfolio_value.txt).

## STOCKACCOUNT_MAREESH.CPP

This file contains the definitions for the functions of the class StockAccount.

It uses the text file stock_portfolio.txt. This text file has the record of all the stocks present in our stock portfolio.

It uses the text file stock_transaction_history.txt. This text file has the record of all the transactions made for stock account.

It uses the text file total_portfolio_value.txt. This text file has the record of all the total portfolio values(stored whenever the user exits the program).

It uses the text files Result_1.txt and Result_2.txt to mimic the changing stock price(of certain stocks present in the file).

It has the following functions:-

StockAccount()-This function sets the *Head to NULL, mySize to 0, points the *impl pointer to SortSelection. If stock_portfolio.txt file is created for the first time, do nothing, because there are no stocks present in your stock portfolio. But, if the file already exists, then make a doubly linked list of all the stocks present in the file by reading values from the file, creating a new node with those values and inserting each new node at the end of the list.

~StockAccount()-This function deletes the doubly linked list(by deleting each node in the list), if the list is not empty.

size()- This function returns the number of nodes in the doubly linked list or the size of the list or the number of stocks present in our portfolio.

get_stock_price(string, string)- This function returns the stock price(if the stock symbol is present in the file) for a particular stock symbol from a randomly selected file(to mimic the fluctuations in the stock price).

print_portfolio()- This function prints the current sorted stock portfolio by traversing sequentially through each node(starting from *Head) in the doubly linked list and printing its data members in a particular format.

update_list(string)- This function updates the value of the price per share for all the stocks in our stock portfolio based upon the most recent stock price by traversing sequentially through each node(starting from *Head) in the doubly linked list.

buy_shares(string,string,int,double)- This function handles the buying of shares operation of the stock account.

If (number of shares * maximum amount per share) > cash_balance then the functions displays an error message and the transaction is failed.

If stock_symbol is not present in our Result_1.txt/Result_2.txt file, then the function displays the message that the symbol was not found.

If current price per share is greater than the maximum amount the user is willing to pay(per share), then the function displays an error message with the appropriate reason for failure of the transaction.

If none of the above three conditions are present then,

The function creates a new node of the doubly linked list. If the list is empty, then this node becomes the head, money for buying the stock is withdrawn from the bank account(therefore gets registered in bank_transaction_history.txt), cash_balance is updated accordingly, and the corresponding transaction is recorded in the stock_transaction_history.txt file. If the list is not empty, and the same stock_symbol is already present in the list, then the function increases its number of shares, money for buying the stock is withdrawn from the bank account(therefore gets registered in bank_transaction_history.txt), cash_balance is updated accordingly, and the corresponding transaction is recorded in the stock_transaction_history.txt file. If the stock_symbol is not present in our existing portfolio then a DLL new_node is created and is added to the back of the DLL, money for buying the stock is withdrawn from the bank account(therefore gets registered in bank_transaction_history.txt), cash_balance is updated accordingly, and the corresponding transaction is recorded in the stock_transaction_history.txt file.

The price per share of all the stocks in the stock portfolio is then updated.

If the number of nodes in the DLL is >1 then the user is asked to select a sorting algorithm(selection or bubble sort). After sorting, the portfolio is printed and store_portfolio() function is called.

sell_shares(string,string,int,double)- This function handles the selling of shares operation of the stock account.

If the DLL list is empty display the message that the stock portfolio is empty.

If the DLL list is not empty, then traverse the list starting from the *Head node. If the stock symbol is found but number of shares of that stock in stock portfolio is less than the number of stocks you want to sell then display an error message. Or if the current price per share is less than the minimum amount you are willing to sell then display an error message.

If none of the above cases occur, then decrease the number of shares of the stock you want to sell. Now, check if the number of shares you just decreased became zero or not. If the number of shares reduced to zero, then remove the node from the DLL.

Then, money for selling the stock is deposited into the bank account(therefore gets registered in bank_transaction_history.txt), cash_balance is updated accordingly, and the corresponding transaction is recorded in the stock_transaction_history.txt file.

If the stock_symbol to sell not found in our stock portfolio then display it in a message.

The price per share of all the stocks in the stock portfolio is then updated.

If the number of nodes in the DLL is >1 and stock_symbol found in our stock portfolio then the user is asked to select a sorting algorithm(selection or bubble sort). After sorting, the portfolio is printed and store_portfolio() function is called.

store_portfolio()- This function stores the current stock portfolio in the file stock_portfolio.txt. It also stores the current total portfolio value in the file total_portfolio_value.txt get_trans_history()- This function prints the data in stock_transaction_history.txt in order of transaction time.

get_graph()- This function reads the total portfolio value and time in seconds from the total_portfolio_value.txt file, stores them in two separate arrays and then plots a graph between them by using MATLAB.

## MAIN_FINAL_PROJECT_MAREESH.CPP

This file contains two objects bank_account and stock_account of classes BankAccount and StockAccount respectively. These objects are used to call the appropriate member functions of the classes according to the input entered by the user.

A major part of the code in this file is dedicated to generating and displaying the appropriate interface to the user, taking user response and performing the appropriate action according to it.

It also has the following functions:-

getFileName()- This function randomly selects one file to mimic the fluctuations in the value of stock prices.

Here, an array of base pointers (*array[]) is also present where the first element of the array points to a StockAccount class object whereas the other element of the array points to a BankAccount class object. This array of base class pointers are used to implement the design pattern(specifically behavioral pattern) template.

The base class Account has a pure virtual function getName(), which is redefined in the derived classes (StockAccount and BankAccount). The base class also has a function print_type which uses getName(). Thus this function when executed with using the getName() function of the base class can be considered as a basic foundation execution. But, when this function accesses the getName() functions for the derived classes, it works in a way similar to adding variations for each of the derived classes over the foundation provided by the base class. This is evident in this file by the use of the lines array[0]->print_type() and array[1]->print_type(), thereby implementing the design pattern-template.