I first started off from an abstract base class, Account, from which the StockAccount and BankAccount classes are derived from. It has basic functionality to set and get the cash balance that's stored in the private cashBalance variable as well as a pure virtual function, changeBalance, that's implemented by the derived classes to save the cash balance in a txt file. Given that the base class offers a template from which the derived classes expand upon, this would be an example of a Template Design Pattern. From there, I moved on to the BankAccount class as it's the simplest one to implement and can be implemented without depending on the other classes. I used a Meyers implementation of a Singleton Design Pattern to ensure only one instance of the class can be used, same for the StockAccount, fitting as only one instance should ever be needed by the program. The BankAccount can adjust and obtain the cash balance from the txt file via its implemented functions. These functions are also used within the deposit and withdraw functions, which save any changes to the bank transaction history and total portfolio value history. When the BankAccount is made, it'll instantly create a txt file called cash_balance.txt and set the balance to $10,000, if one doesn't exist, otherwise, it'll obtain the cash balance from said txt file. It'll also create and initialize the bank_transaction_history.txt file, if it doesn't exist. There's also a function to print the bank transaction history, if needed.

From there, I moved on to making the Stock and DoubleLinkedList classes as they'll be used as the portfolio in the StockAccount class. The Stock class basically acts as the node for the DoubleLinkedList class, storing the stock name, price of the share, as well as the number of shares of said stock, while also storing information for the previous and next nodes. The DoubleLinkedList is essentially the portfolio of the StockAccount, holding sorted Stock nodes, the size of the list, as well as all instances of the total portfolio value and the associated times within vectors. In my case, in order to make working with the DoubleLinkedList easier, I used two sentinel, or empty, nodes at the head and tail, allowing for me to avoid having to deal with cases when you have to delete or add at the head or tail. In my case, I had the stock account's functionality split between the StockAccount class as well as the DoubleLinkedList class. Sorting the portfolio, adding and removing stocks, printing the portfolio, obtaining and saving portfolio as well as total portfolio value information from and to the txt files, getting the stock price, finding the stock, as well as showing the graph of the total portfolio values on a time scale was handled by the DoubleLinkedList class. When the StockAccount class is made, it'll instantly create a txt file called cash_balance.txt and set the balance to $10,000, if one doesn't exist, otherwise, it'll obtain the cash balance from said txt file. It'll also create and initialize the stock_transaction_history.txt, portfolio.txt, and total.txt files, if they don't exist, and obtain the portfolio and total portfolio value history. It'll also handle buying and selling shares, which changes not only the stock transaction history but also the portfolio, bank transaction history, and total portfolio value history. There's also a function to print the stock transaction history, if needed.