Project Technical Description

Brief Description

In this project/program , an abstract base class **‘Account’** has been created with two derived classes **‘StockAccount’** and **‘BankAccount’** connected with the **balance(account_balance term used in my project for the cash balance).**

This implementation can be done by making the balance static but I have led the balances as two separate variables in the two classes and have them synchronized.

Further a file named account_balance has been created for storing the present value of the balance whenever the program exit.

And when the program starts the constructors of both classes make themselves loaded with the cash balance value from this static persistent file.

Further during the first time when the file **“account_balance.txt”** is not there for storing the account balance value , then these values are loaded with the **10000 dollars value** and the file is created in the local folder and then for further exit and start of the program , the balances in the objects of stockaccount and bankaccount classes are being loaded in and reloaded from that file only. So , **staticness** come here from the file stored in the local folder.

Further , the two files **“Result1.txt” and “Result2.txt”** having the same stock symbols with different prices for mimicking the stock market randomness have been used. So any time when the price of any stock has to be searched then randomly the price from any of the two files was used.

Stock portfolio was stored in the **circular doubly linked list**. List template (**showcasing the behavioral design pattern of templating)** was used for creating any type of list like singly , doubly etc. with common interfaces(methods) of insert , delete etc. was used. Here I have created a derived class **“doubly linked list”** and the singly linked list has been shown here for just the reference as that can also be used, if implemented and hereby showcase the design pattern.

Because list pointer is there inside the portfolio class which is used to abstract the implementation of the list . This Is what is the **“ Structural Bridge design Pattern”.**

Further , stock portfolio in the doubly linked list was made to be sorted at all times in the decreasing order of the total value of any particular stock (i.e. number of shares * price per share). That is, after any update on your stock portfolio the doubly linked list was maintained to keep it sorted.

Three design patterns were used . Two have been highlighted above and the third for sorting explained in the class has been used which is a **behavioral strategy pattern** i.e. **sortstat** is the strategy class which make us select the implementation using the base pointer of **sortimpl** which has the derived classes of **bubblesort** and the **selectionsort**. **Sortstat class** make us select the strategy , by default the bubblesort where the actual implementation class object pointer is put into the sortimpl (base class pointer). For any upgrade and degrade , the sortstat class create the new sorting object of different class and put it into the base pointer of sortimpl. And using the polymorphism , we use the same base pointer of the sortimpl class to call any sorting class – bubble or selection sort. I have created selection sort class for show casing only but in my project I have used bubble only.

Further necessary functions are created in the stockaccount and bankaccount class and other classes as shown below for doing the desired functions which are also provided in detail below for the understanding. Detailed class structure diagram which is an important aspect of any application designing and implementation has been created and shown for crystal clear understanding of my thought process.

Program is doing the following tasks perfectly as desired :-

A. Stock related activities

1. **Display the price of a stock**
2. **Display the current portfolio**
3. **Buy shares**
    a. The user will choose an option to buy shares of a stock.
    b. The user will enter the ticker symbol of the stock he/she wants to buy,
    c. Then enter the amount of shares he/she wants to buy and the maximum amount he/she is willing to pay for each share of the stock (the limit).
    d. If the amount for buying is more than his or her current cash balance in bank account, the transaction will fail and will print out the reason.
    e. Each time the user requests to buy shares, the stock pricing information from Results.txt is being consulted(to mimic total randomness of stock market)
    f. If the stock ticker is not found in the text files, the program print that the stock is not available.
    g. If the cost per stock is higher than the amount the user is willing to pay, the transaction will fail and the program will print out the reason the transaction failed.
    h. If the transaction goes through, the cost of the transaction gets deducted from the bank cash balance
    i. Stock that has been purchased should be added to the portfolio.
    j. The program display on the screen the information from the transaction that has taken place.
    k. This transaction also gets added to the **stock_transaction_history.txt** file.
    l. Simultaneously , a withdrawal transaction is added to your bank account transaction history for the buying operation.

4. **Sell Shares**
    a. The user first send a request to sell shares of a stock.
    b. Then the user enter the ticker symbol of the stock he/she wants to sell, the amount of shares he/she wants to sell and the minimum amount he/she wants to sell each share of the stock for.
    c. If the user enters a stock that is not in the portfolio, or if there are insufficient shares of that stock in the portfolio, the program display that the transaction has failed together with the reason.
    d. Each time the user requests to sell shares (and the shares are available), the stock pricing information from Results.txt is used.

e. If the price per stock is lower than the amount the user is willing to sell for, the transaction fails and the program print out the reason for the transaction being failed.
f. If the transaction goes through, the amount obtained from the **“sell transaction“** gets added to the bank cash balance and the shares that have been sold should be subtracted from the portfolio.
g. If the number of shares of the stock is 0, the stock gets removed from the portfolio.
h. As desired the price of the stock that is in “Results.txt” files when updating the portfolio not static values stored inside the program.
i. Further , the transaction is added to the stock_transaction_history.txt text file.(exact names are written below in files used section)
j. Deposit transaction is also added to the bank account transaction history for this selling operation.

5. Graph of historical total portfolio values

a. Here , I have made a macro HISTORY for providing the number of recent total portfolio values you want to see in terms of graph in order to judge the trend of your portfolio.
b. By default, I have used **5** , but can be **increased or decreased also** , by changing the macro above and the x axis value parameter from 5 to that value.
c. Important thing is that whenever the program properly exit through menu only then only the total portfolio values (i.e. the total stock value , total cash balance and the timestamp)are being added to the file.
d. Then the latest of HISTORY MACRO many transactions are taken and are plotted over the graph to give us the trend.

6. View transaction history

a. Displays all the buy , sell transactions in the stock transactions history file.

B. Bank related activities

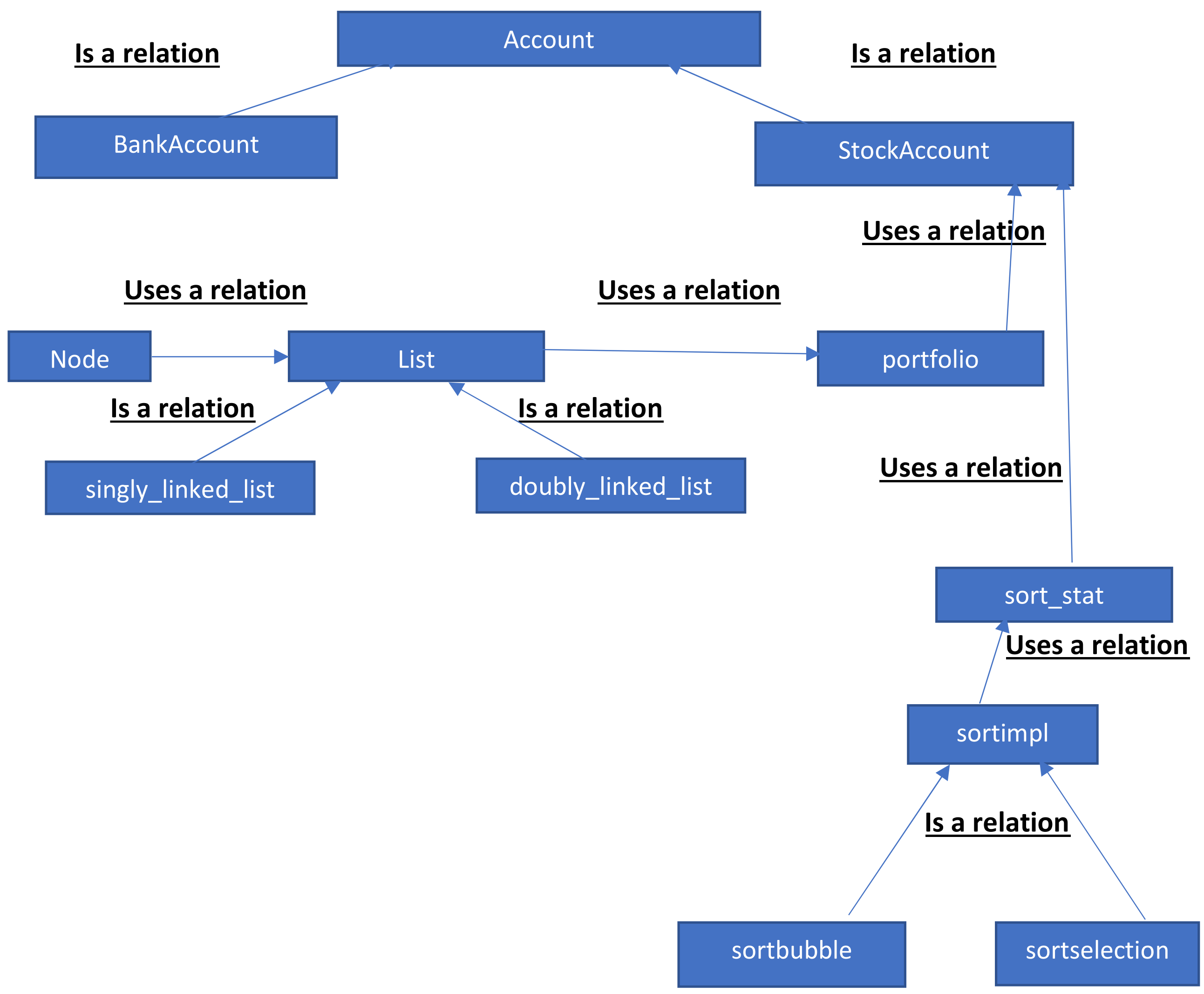
1. View account balance – Displays the account balance.

2. Deposit Money – The user is prompted to mention the amount of money to deposit. This amount should be added to the balance.

3. Withdraw Money – The user selects the amount of money to withdraw. If the balance of the account is not sufficient to withdraw the amount, the error is printed .

4. Print out history - The program print out the history of transactions on the account in order of transaction time.

Class structure



Explanation

In the above diagram as clearly mentioned the following classes are being used:-

1. **Account**
2. **BankAccount**
3. **StockAccount**
4. **Node**
5. **List**
6. **singly_linked_list**
7. **doubly_linked_list**

8. **portfolio**
9. **sortimpl**
10. **sortstat**
11. **sortbubble**
12. **sortselection**

As told in the problem statement, Account is an abstract class which has two derived classes – BankAccount and StockAccount.

Node class provide the data structure for the list class. List class is the abstract class i.e. a template for the various types of lists like singly linked list, doubly linked list etc.
So, List provides a behavioral design pattern wherein the template class here ,the list, presents a template (these are the methods and data) for the derived classes like singly_linked_list and doubly_linked_list class to implement.

As we have used doubly linked class so I have implemented the doubly linked list class for the template of list i.e. for insertion, deletion, searching, sorting etc. of nodes in the list.
Singly_linked_list class has just been made a dummy to show the design pattern using polymorphism and inheritance concepts in the C++/OOPs.

Portfolio class is to create a bridge between the lists and the stock portfolio data structure used inside the StockAccount Class.

StockAccount class needs to maintain the stock portfolio in the doubly linked list as mentioned by the problem statement. Here StockAccount Class has a member variable of data structure of **portfolio class. Portfolio class** has a member variable of the pointer to the List template/Abstract Class . This pointer variable stores the type of list i.e. the implementation of the list template as a singly linked list class and doubly linked list class . Here we created variable of the doubly_linked_list class and store its address to the List pointer variable member of the portfolio class , so as to enable us to let us decide at run time which type of list we want to use in our implementation. So portfolio class behaves as an abstraction class and list is an implementation class where the implementation behavior can be changed at the runtime. **THIS IS THE STRUCTURAL BRIDGE DESIGN Pattern**.(will be more explained further).

Classes and their functions

1. **Account**
   a. **Member Variables**
- double account_balance;
   b. **Member Functions**
- virtual double credit(double) = 0;
- virtual double debit(double) = 0;
- double get_cash_balance_value();

Explanation:

Account is the base template/abstract class for both the BankAccount and StockAccount Class. As an account, the members are made the account_balance which is the term for **the cash balance term** mentioned in the problem statement. Further the two virtual functions are there which are common to any account i.e. the debit and credit which are for the debit (withdrawal) and credit(deposit) to the account. Further the function added is the **get_cash_balance** which every children class will inherit.

2. **BankAccount**
   a. **Member Variables**
   b. **Member Functions**
      - BankAccount()
      - ~BankAccount()
      - void file_update_balance()
      - double debit(double amt)
      - void record_transaction(string event, double amount, double balance)
      - void display_transactions_history()
      - double credit(double amt)
      - void display_balance()
      - double get_balance_value()
      - void update_balance_value(double x)

Explanation

BankAccount is the children class of the account class with constructors and destructors. **The constructor** is for the taking the current static account balance from the file created. When the program will first execute , it will take the value of 10,000 dollars and then create the file for storage of the cash balance and then that file will remain created (in the current folder of the program named "account_balance") for holding the cash balance from which both the StockAccount and the BankAccount class will take the value for further execution of the program because after the first time the **account_balance file** would have been created. Here for this class constructor will be doing this work.

File_update_balance function is to update the static file as mentioned above created , for updating with the current balance after any debit and credit of the account.

Debit and credit functions are the functions as the name mentioned are for the debit and the credit of the amount. And then they will call the file_update_balance function and the record_transaction function for recording the withdrawal and the deposit transaction functions in the transaction file.

display_transactions_history function is for as the name suggest for the displaying the record of transactions . This will read from the static file maintained for maintaining the banking transactions record and let it get displayed on the screen for the users.

update_balance_value function is for updating the account balance information this is necessary for maintaining the coordination between the account balance of the objects of the stock account class and the bank account class , so we have the reason to update the balance of the BankAccount class.

get_balance_value and the display_balance are self explanatory i.e. for displaying and getting the value of the balance value.

3. **StockAccount**
   a. **Member Variables**
      - portfolio stock_prof;
   b. **Member Functions**
      - StockAccount(list * x):stock_prof(x)
      - ~StockAccount();
      - void update_stock_portfolio();
      - void update_balance_value(double x);
      - void file_txn_add(string event, string stock, int num, double value);
      - double credit(double x=0);
      - double fsearch(string searc);
      - double debit(double y = 0);
      - void display();
      - void trans_history();
      -

Explanation

Constructor function is for initializing with the address of the type of list of the parent template list class. Like in our program it is the doubly linked class. And also to take the account balance from the static class built in for this purpose as mentioned in the BankAccount class.

Destructor is for the storing the stock profile in the static file and also to maintain the total stock profile value and the cash balance and the respective timestamp.

Here, I would like to bring to your notice that proper exit through the menu is required for this to occur as the destructor will not be called. So please do the proper exit otherwise this recording of the total stock profile and the cash balance value and the time stamp will not be stored in the file and hence the proper graph will not be produced.

update_stock_portfolio() function is used to store the latest stock profile value in the static file for just maintain the persistence of the stock profile and hence when the program will again execute then this file will be used for filling the doubly linked list i.e. the portfolio data structure again.

update_balance_value function is used for updating the account balance (cash balance) of the StockAccount Class.

file_txn_add is for adding the necessary buying and selling of the stocks transaction in the file for maintain the persistence of the data.

Credit and the debit functions are for the buying and selling of the stocks as they are for implementing the virtual functions that need to be implemented in the children class with default input value. Credit is for buying and debit is for selling of the shares.

Fsearch function is for file searching i.e. for getting the value of the stock by randomly opening the one of the two files and then getting the values out of them.

Display function is for displaying the current account information and the **trans_history** function is for the display of the all buying and selling transactions after reading from the file.

4. **Node**

   a. **Member Variables**

- string stock_name;
- int number_shares;
- node* forward;
- node* back;

   b. **Member Functions**
      - node(string name, int number)

   <u>Explanation</u>

   As per the problem statement we must create the doubly linked list for the storage of the stock profile. So node is a class/data structure with the constructor for storing the required values for each node to be maintained in the stock profile.
   In this program , I have created the circular doubly linked list. So node needs to carry the name of the stock and its number of shares .

5. **List**

   a. **Member Variables**
- node* first;
- node* last;

   b. **Member Functions**
- virtual void insert(node*)=0;
- virtual void sort() = 0;
- virtual double display() = 0;
- virtual void file_add(fstream &) = 0;
- virtual node* search(string) = 0;
- virtual double total() = 0;
- virtual void delete_(node *) = 0;

- virtual ~list(){}

<u>**Explanation**</u>

List is the abstract class / templates in the C++ for the creating a multipurpose lists implementation like in the form of doubly and the singly linked list. So all above function are self explanatory which every class that implements this template needs to implement. **THIS IS THE BEHAVORIAL TEMPLATE DESIGN PATTERN.**

Here the **insert function** is for inserting , **sort function** is for sorting , **display function** is for displaying , **file_add function** is for storing the values in the static file , **search** is for searching the value , **total function** is kind of the reduce function implementation which is very much being used in all data structures nowadays and **delete_** function is for deleting the node.

6. **singly_linked_list**

   a. **Member Variables**
   - node* first;
   - node* last;

   b. **Member Functions**
   - void insert(node*);
   - void sort();
   - double display();
   - void file_add(fstream &);
   - node* search(string);
   - double total();
   - void delete_(node *);
   - ~ singly_linked_list()

<u>**Explanation**</u>

This class has no function described in the program they are just for the demonstration function that by creating the template list class and by using the polymorphism we can implement the behavorial template design pattern and the structural bridge design pattern for just making the implementation of the list abstracted and hence can be changed at the run time.

7. **doubly_linked_list**

   c. **Member Variables**
   - node* first;
   - node* last;

   d. **Member Functions**
   - void insert(node*);

- void sort();
- double display();
- void file_add(fstream &);
- node* search(string);
- double total();
- void delete_(node *);
- double fsearch(string searc);
- ~doubly_linked_list();

Explanation

This class has all functions described in the template class list and implement the doubly linked list and for that implement the various required functions .

8. **portfolio**

   a. **Member Variables**
   - list* listing;

   b. **Member Functions**
   - portfolio()
   - portfolio(list* x)

   Explanation

   This is the abstraction class for the list and named portfolio as it abstracts the actual implementation of the list using the singly , doubly and other type of lists at the run time using the templating design and the polymorphism by creating the pointer of the List class and then just storing the address of the various types of lists to be used and then calling the functions which will be called based on the address of the type of lists in the pointer variable.

9. **Sortimpl**

   Member Functions

   a. virtual void sort(list*) = 0;
   b. double fsearch(string);

10. **sortbubble : public sortimpl**

    Member Functions

    a. void sort(list*);

11. **sortbubble : public sortimpl**

Member Functions

a. void sort(list*);

12. sort_stat

Member Variables

a. sortimpl* m_impl;

Member Functions

a. void sort(list*);
b. sort_stat();
c. void upGrade();
d. void downGrade();

<u>File created and Used</u>

1. **“account_balance**” for storing the current latest account balance and for loading the cash balance for the individual account classes for the further usage after the exit.
2. **“bank_transactions”** for storing the deposit and withdraw transactions happening in the account.
3. **“stock_portfolio”** for storing the latest stock portfolio of the portfolio of the user.
4. **“stock_transactions”** for storing the buying and selling transactions happening in the account.
5. **“total_portfolio”** for storing the total stock portfolio value and the current cash balance and the time stamp during the exit of the program.
6. **“Result_1” and “Result_2”** are being used for getting the stock value by the stock name.

PLEASE: <u>Note only the **“Result_1” and “Result_2”** text files are required for getting the random stock value and all other files will be created in the current folder of the project. So for testing and running please first create the new project and then please add all the files or otherwise create all the above named empty txt files in the current folder of the project.</u>

<u>Design patterns</u>

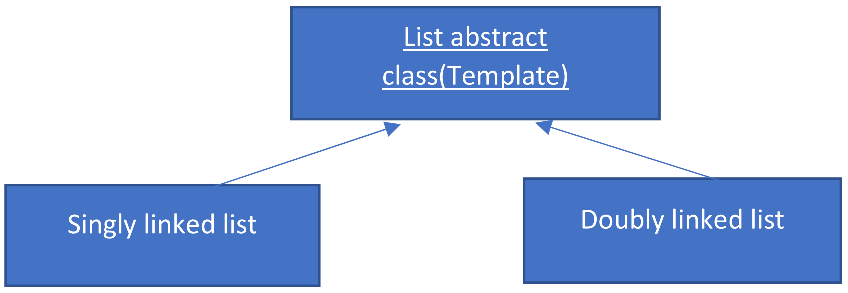
As mentioned above the following design patterns are being used :-

1. **<u>Behavorial Template Design Pattern</u>**

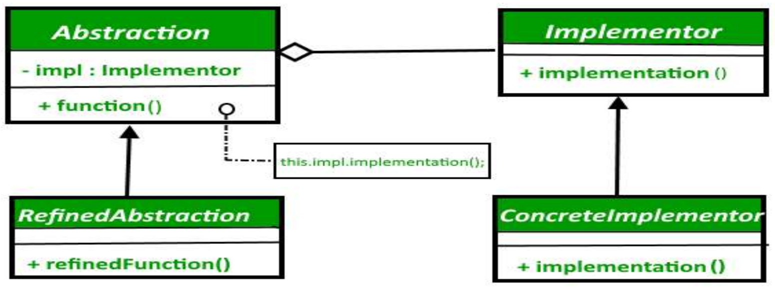
Behavioral patterns are about identifying common communication patterns between objects and realize these patterns. Template pattern defines the skeleton of an algorithm in an operation deferring some steps to sub-classes.

This has been showcased in my project by setting the structure / pattern by the list class and the individual classes like singly_linked_list and doubly_linked_list redefine the functions delegated to them for implementation.

Project Implementation of this design pattern

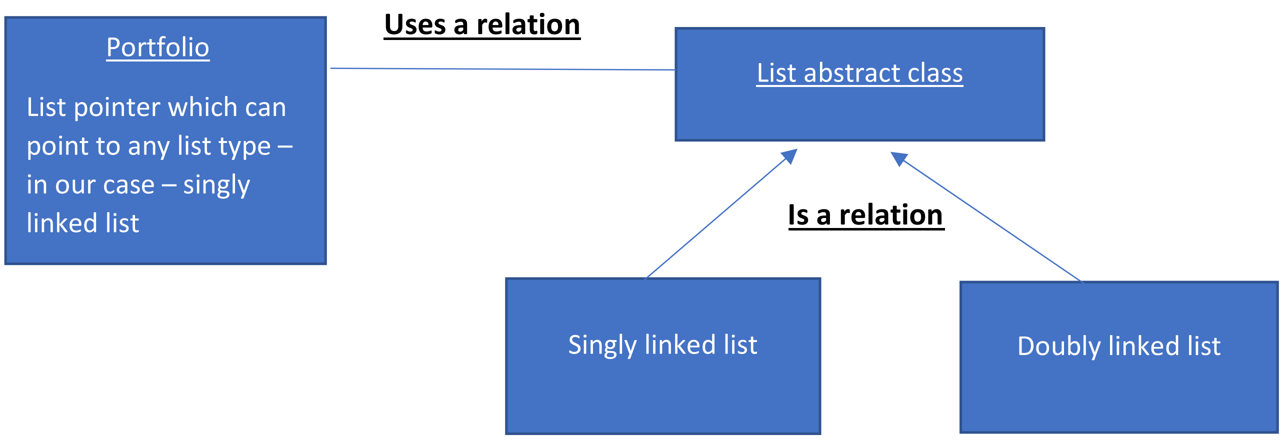


2. **Structural Bridge pattern**

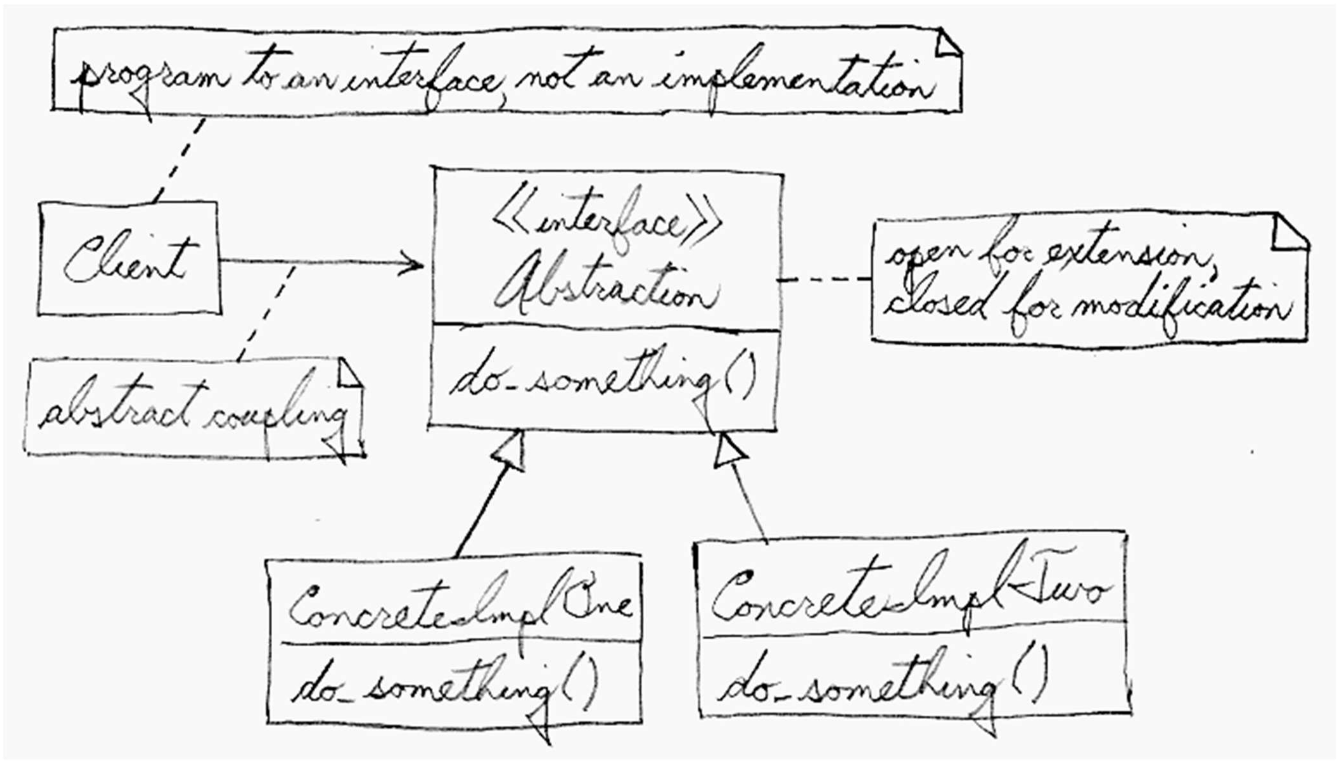


As shown the abstraction and the implementation in the implementor in our class design is the list class and the abstraction of portfolio is having the list pointer which can contain any concrete implementation i.e. of doubly or a singly linked list.

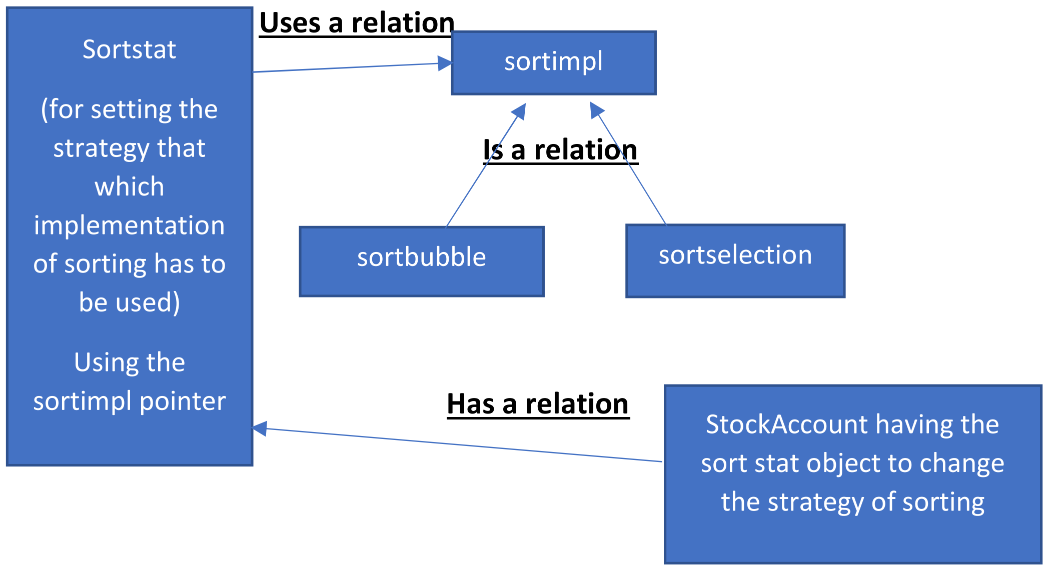
Project Implementation of this design pattern



3. Behavorial strategy pattern



Courtesy : lecture slides


Sortstat
(for setting the strategy that which implementation of sorting has to be used)
Using the sortimpl pointer
Uses a relation
sortimpl
Is a relation
sortbubble
sortselection
Has a relation
StockAccount having the sort stat object to change the strategy of sorting

The above diagram represents the design pattern used. Here the **sortimpl** is the abstract class which is being implemented by the **sortbubble and the sortselection class.**

sortstat is a strategy class which allows us to change the strategy of sorting.

stockaccount class is using this **sortstat** strategy class' object to implement/change the strategy of sorting.