**Class Diagram:**

Agent

Broker

void buy()

void sell()

Command \* address\_of\_request()

string getSE()

request()

void executeorder()

Command (Interface)

Receiver

virtual void execute()

Sellstocks

Buystocks

void execute()

void execute()

StockTrade

void buy(string, long)

void sell(string, long)

**Description of the design:**

I have used **Command pattern** as my design choice. The class Agent accepts the order from the client and it encapsulates the request in an object of type Command \* and passes it to the invoker which is class Broker. The Broker object executes the request object by calling the appropriate execute method of the derived class. I have used run time polymorphism to implement this behaviour of the base class wherein it can execute the derived class’s execute() from a pointer of its own type. I have declared the function execute() in the class Command as virtual to implement this. I have declared two classes Buystocks and SellStocks which are concrete command classes and are used to make the request object. The Agent class uses them to encapsulate the request and these objects are then passed to the invoker.

**Application of the design pattern to this problem and justification:**

The design pattern is the Command pattern. The following classes are used:

class StockTrade: This is the receiver class and the interface displays what stocks were traded and in what quantity.

class Command: This is the abstract class or an “interface” that is used to implement the execute() method. It is used because this class is then inherited by two classes BuyStocks and Sellstocks. They overload the execute() method. The abstract class is designed with a virtual function so as to implement run-time polymorphism and dynamic binding. The request is encapsulated in either the Buystocks object or Sellstocks object and these objects are executed by the invoker which is the class Broker by calling their execute methods within the Broker object’s function executeorder().

class Buystocks: This is a concrete command class and is used when the client wants to place an order. If the order is to buy then this class is used to encapsulate the request. It uses the the interface provided by the Strock Trade system to buy.

class Sellstocks: This behaves similar to the Buystocks class and is used for selling stocks. It uses the the interface provided by the Strock Trade system to sell.

class Agent: The Agent class is used to place the order. The client specifies the name of the stock and the amount to buy/sell and this information is used by the Agent class. The Agent class encapsulates this request in an object of type Buystocks or Sellstocks and passes this object to the invoker which is the Broker class.

class Broker: The Broker class is where the instructions in the request object are executed. The Broker class calls the execute() method of the Buystocks or Sellstocks and the appropriate function is executed.

Thus, as you can see from the description and working above, that we are decoupling the requester (the Agent class) of an action from the object that actually performs the action (Broker class).

A command object of type Buystocks and Sellstocks encapsulates the request to do something on the object (StockTrade).

**Hence, we are separating the object making a request to the object that receives and executes those requests and this is the goal of the Command pattern.**