Brad Saterfiel
MPCS 51050
Summer 2014
Final Project

Executive Summary

The project is meant to show the lifecycle of a stock trade. It is contrived intentionally; some of the components introduced were dumbed down and used only for the execution of the full cycle. The stocks were picked at random as well as the day. The cycle of a trade is far more cumbersome when introducing all of the necessary parts, multiple products, advanced schemes, and compliant risk management. This is far beyond the scope of this project and would not be conceivable in the time span allotted. The basic, yet necessary, parts I introduce here are the stream of quotes from a quote vendor, an interface to consume the quotes, a trade management component checking both strategies and liquidity, a broker to fill and close the trades, and a risk manager to make sure the trades are monitored and closed according to the risk management scheme. Many of the components could have additional strategies, schemes, and/or configurations. Again, that is beyond the scope of this project.

QuoteStream

This is meant to represent the vendor (could be multiple) sending quotes to the program waiting to accept them. The quotes are actual 1 minute data from 7/1/2014 for the stocks CI (Cigna Corp.) and COG (Cabot Oil & Gas Corporation). These were taken from EOD data and split into 1 minute files similarly to previous labs using Prof Shacklette's script. These 1 minute segments are then sent to 2 queues waiting for the quotes. The first is the QuoteStreamBroker where they will be consumed for trade initiation, and the second is the FirmRiskManager where the risk manager will monitor the quotes regardless of what the state of trades is.

Inbound/Outbound Queues/Topics: 1 inbound (from File) / 2 outbound

EIP patterns: Message Channel, Message Translator, Point to Point Channel, Splitter

Patterns:

QuoteConsumer

The QuoteConsumer is where the quote is first processed in house (by the Firm's program). It manages the bulk of the work and houses most of the patterns in use. Here there is a Singleton in use for the stocks available to trade. It's purpose is to impose the origination of trades. These stocks are put into a HashMap. The only, yet certainly extensible, strategy is the Moving Average Crossover of a 5 minute (FMA) and 15 minute (SMA) with a contingency on the cross of greater than .1% difference. It is a contrived, cooked up strategy, but meant to show that use of the Strategy Pattern and where these may be injected in the cycle. The trade abstract class is a Template Pattern. It sets up the basis for either creating a long or short trade. Here the Class Stock also maintains several vectors of useful information. The information is used in deciding when to trade and could further be etched into different strategies. The Observer Pattern is used to write to the console on changes of the stock's trade status. Iterating through the observers (in this case there is only one, but again, it remains extensible) displays the Iterator Pattern. Upon trade decision, the messages are either packaged with a "No Trade" or trade body and passed on to the queue FirmTradeManager.

Inbound/Outbound Queues/Topics: 2 inbound / 1 outbound

EIP patterns: Message Translator, Point to Point Channel, Log

Patterns: Singleton, Strategy, Template, Observer, Iterator

TradeRouter

Here the trade or "No Trade" policy is examined. The trades are packaged with the ExecuteTrade process. The trades are first checked for portfolio liquidity. The Portfolio is a Singleton. There are other methods here, but they will be explained later in the documentation. The "No Trade"s are sent to a dump queue. The trades that pass the liquidity check are then sent further on to the Broker for a fill.

Inbound/Outbound Queues/Topics: 3 inbound / 2 outbound

EIP patterns: Message Router, Message Translator, Content Based Router

Patterns: Singleton, Iterator

Broker

This is the second most contrived class (behind QuoteStream) as it only has two processes that attach an "OK" for a fill and a "Closed" to indicate a fill on the close. This is similar in concept to what would happen in real life, except the message would give the actual price if you were using a market order. Here we assume the fill is at the price of the last trade, be it for entry or exit. Obviously, a firm with a very large position to fill would have an algorithm to manage entry and exit for a large trade so as to not give away their position, prediction, or liquidity consumption/dispersion. The trades are sent to the Risk Manager and Confirmed Trades topics where the Risk Manager and TradeRouter are listening. The Risk Manager will then monitor the trade. The TradeRouter will mark the trade in the Portfolio.

Inbound/Outbound Queues/Topics: 2 inbound / 3 outbound

EIP patterns: Message Translator, Point to Point Channel, Log

Patterns:

Risk Manager

After leaving the QuoteConsumer, this is the lion's share of the backend work. It is more than likely how a firm will live or die. For the project, this class maintains the trade once filled. As was noted, the trade (in this example) is guaranteed to be filled. Here there is a simple exit strategy…either exit trade taking profit at .5% gain or taking loss at .35% of stock price. Once a trade hits those simple requirements, we send a closing trade to the broker. Again, the assumption is that the fill is executed at the last closing price. Once the trade is returned as "Closed" the TradeRouter updates the Portfolio and a further message is consumed by the QuoteConsumer to make note the stock is no longer in a trade.

Inbound/Outbound Queues/Topics: 3 inbound/4 outbound

EIP patterns: Message Translator, Point to Point Channel, Pub/Sub, Message Filter

Patterns: Singleton

Conclusion

I'm very satisfied with the overall project given the time constraints. The documentation also has a rough sketch to aid the written documentation. This data does "make" money, but is highly far-fetched. Camel is powerful and the queues/topics are highly responsive. The project was a fair amount to maintain over the course of several sessions. I appreciate the guidance I received in class and in the TA sessions.

References

- Camel online documentation
- *Camel in Action*, Ibsen & Anstey
- *Design Patterns*, Gamma, Helm, Johnson, & Vlissides
- *Enterprise Integration Patterns*, Hohpe & Woolf
- In-class assignments, lectures & TA Sessions
- http://www.tutorialspoint.com/design_pattern
- http://www.java2s.com/Tutorial/Java
- http://javarevisited.blogspot.com/2011/12/observer-design-pattern-java-example.html