Running the project

The different components of the project and what they can do as of now have been explained in the Documentation.pdf. This document assumes that points mentioned in that document are clear before proceeding with this one. There are many scenarios one may want to test, and how to test may not have been clear from the documentation. This document is to clear that. The following are the scenarios that have been tested, and the method to do the same has been mentioned.

Simple Match Testing. One basic scenario is to check if the matcher runs correctly. i.e., the market parses input orders correctly, matches them according to specified rules, and produces an update stream. To run the virtual market, all that the coder needs to do is create a StockMarketServer object. This object essentially does the task of hosting a server at the address specified in its constructor. Then one needs to run a client class, that can connect to the server and exchange messages. One way of doing so is shown in test.TestClient.java. This class has a main, and a handler (which can be adjusted to print relevant details from the market's update stream) which can read in orders from the console and send them to the market.

Market Maker Testing. There is also a case where one may want to test a market maker. The example of how to do so is shown in test.MarketMakerTest.java. Once again the market is simulated by creating the StockMarketServer object. This hosts the market server. Then a historic stream is started, which replays data from the history into the market. Currently the code for syncing historic timestamp with current timestamp has been commented to allow heavier load of historic data. Once the historic stream is begun, the market maker object is created, and the execute() method of the market maker object is called to begin its function. Code for the testing is as simple as shown below:

```
StockMarketServer s = new StockMarketServer();

HistoricTrader ht = new HistoricTrader(); ht.run();

MarketMaker mm = new MarketMaker(0.1, 1000, 1000, 10101); mm.execute();
```

However it should be noted that the mm.execute() call is a blocking call, and should be made in the end, or through a thread if more than one Market maker is to be added to the market. (It can also be done through a different main(), or by adding a main() to the MarketMaker class, enabling it to be a stand alone process).

Algo Trader testing. Here there is not as much uniformity as there should be. Up till now all executions have been integrated into one, where all components have a method which allows them to start executing. However, for algo traders, they have a separate main, which essentially makes them a separate process. This same thing can be done for market makers as well, and indeed it would be cleaner for the codebase to handle both market makers and algo trader executions similarly. However for now, they are handled differently, and the coding sequence to test algo traders would be: Create the StockMarketServer object, and run the main of the algo trader you want to run. (you can also start the historic stream as done in the previous example).