OO Design & UML

This diagram is an example of how we see the MVP being built, with a fleet of java trade bots behind the scenes, buying and selling set stocks according to a sin curve, causing the pricing of the stocks to fluctuate accordingly due to a Stonk\_Change being passed to a Stonk every time a buy or sell action occurs. The Stonk prices are then all updated at set time intervals, and a snapshot of the previous value saved.

As SpringBoot is still new to all the members of the team it has been more challenging to visualise how the classes above will interface with Spring. An example of the progress made towards this can be seen in the Player\_Bot (PB) and Player\_Update\_System (PUS) classes. We believe that each team should have a team username and password that will be passed to the PUS, and then by checking the registered player bots the PUS can figure out which PB to update. This decreases the possibility for attacks between teams, where nonsensical commands can be sent to another teams bot as a form of sabotage, while also allowing the system to identify incoming instructions. The DO command would tell a PB to interact with a certain number of a certain stock. Negative would be sell, positive would buy. The PUT command would generate a new Player\_Bot and register it with the username and password passed as arguments. The GET command would return the player rankings, with the Bot name (Not the team username!) and their current net worth. This can then be sorted and displayed as a leaderboard by our website.

When any bot buys or sells on the Stonk Market it will call the respective methods (not shown above) to see if A: It can buy (Does it have enough money, and in a later release if there is a buyer waiting) and B: If it has enough money for the number of stocks it wants. If this succeeds, a Receipt will be made, which will both be passed back to the Bot and also added to the transaction history. The Stonk market will then create a new Stonk\_Change based on the value of the transaction, and pass it to the respective Stonk. This will cause the price to change; and increase over time if it was just bought and a decrease over time if it was just sold.

The main bulk of the backend has been modelled with very little Spring based interaction, because there are not many complicated actions we predict the system to have to handle. Any additional GET functions needed can be implemented by suplimentary Spring classes the simply call the already available getter methods, and all complicated decision logic should be left to the bots the teams have programmed themselves. The only interaction they need with the project should be a simple “buy/ sell” command. By focusing on an easily accessible system with inbuilt flexibility (An overridable Update in Trade\_Bot and the freeness of the Stonk\_Change struct) it will be easy to adapt to any additional web based needs, which is a must considering our teams lack of web based experience. This will let us easily rectify mistakes we may make due to inexperience.