Conventions:

- camelCase looks like this
- /**
  *Javadoc looks like this
  * @param parameters
  * @return returnValue
  */
- Type /** and press enter in intellij and it will autocomplete javadoc

How to Calculate Voting Power:

- Received budget based on users voting power
    - Each User can use funds from a global funds pool
    - Suppose a User has initial funds from the pool = $1000 and his voting power = m
- Make transactions
    - If profit
        - Change profitability and re calculate voting power
        - Generate bonus cash based on the voting power
        - New Budget = current cash + profit from the transactions + bonus from the pool
            - 1100 = 1000 + $50 + 50 bonus, voting power = 2.2
    - If lose
        - Change profitability and re calculate voting power
        - New budget = current cash
            - Lost 100
            - Current budget = 900 voting power = 1.8
- When user want to restart the budget
    - Repeat first step
        - Change all asset into cash
        - Get initial funds from the pool = $2000 voting power 2.2.
            - Voting power 1.8, get $950.

Problem:

- Do we have to vote for when selling?
- Ensure the total budget is less than

Voting Power Specification

Definitions:

Group Pool - the total capital in the system.
Voting Power - how much a User's Vote matters relative to other Users' Votes.
Deposit - how much capital did the individual User bring into the system.
Budget - how much capital an individual User can utilize from the Group Pool to make transactions.
Initial Budget - the Budget for a new User.
Net Profit - measures the historical Net profit of the User.
Portfolio: a combination of assets

Voting Power Formula:

Voting Power = F(Net Profit)

Initial Budget Formula:

Initial Budget = F(Deposit, Voting Power)

Budget Formula:

Budget = F(Initial Budget, Net Profit, Voting Power)

Net Profit Formula:

Net Profit = sum(Profits) - sum(Losses)

Entities:

Portfolio extends Identifiable:
Responsibilities:

- Track historical and current portfolio information for a User

Implementation:

- Contains list of transaction history, voting history, profitability, current assets for a User

Asset extends Identifiable

Responsibilities:

- Keep track of information for a stock, cryptocurrency, currency, etc.
- Stores price when bought, and does not change until it is sold

Implementation:

- Price ($ per unit) and volume (how many units) and getters and setters
- Price is updated once when the transaction is executed
- Type of asset: stock, cryptocurrency, currency. (subclass)
- Type includes information on the specific type of asset. For example, stocks may have a symbol, market cap, 24h change, etc.

User extends Identifiable

Responsibilities:

- None

Implementation:

- Name of User
- UUID (Unique ID)
- Admin status (authorities) (subclasses)
- User portfolio

Transaction extends Identifiable

Responsibilities:

- Record information for a transaction in the system

Implementation:

- UUID
- Initiator (User), Asset to sell, Asset to buy (getters)
- Date/Time of execution

Vote extends Identifiable

Responsibilities:

- Record information for a vote

Implementation:

- Boolean for upvote or downvote
- Initiator (User)

CommandParser:
Responsibilities:
- Take an input string and output an executable Command object

Implementation:
- Parse() method: Take a input string, seperate by words
- Check if the words follows the correct format
- If so, use CommandManager to loop through all of the commands and compare their name() strings
- Then make the command

CommandManager:
Responsibilities:
- Provides centralized access for all command subtypes

Implementation:
- Stores instances of each command
- stores all command subtype classes
- get a list of all available commands
- Allow instantiation of any Command

<Abstract> Identifiable:
Responsibilities:
- Provide unified source for identifying objects

Implementation:
- Creates a UUID for the object
- Override equalsTo and hashCode methods so that subtypes are compatible with hashmaps
- Is able to be compared to other Identifiable objects

<Interface> Command:
Responsibilities:
- Interface for abstraction of commands

Implementation:
- execute() method to execute functionality of commands
- help() method to get help string
- name() method to get name for CommandParser to match

Buy implements Command:
Responsibilities:
- Use TransactionManager to initiate a buy order

Implementation:
- implement responsibility within execute() method

Sell implements Command:
Responsibilities:
- Use TransactionManager to initiate a sell order

Implementation:
- implement responsibility within execute() method

Upvote implements Command:
Responsibilities:
- Use VoteManager to add a Upvote Vote to an existing transaction

Implementation:
- implement responsibility within execute() method

Help implements Command:
Responsibilities:
- Use CommandManager to get the help strings of other commands and display it

Implementation:
- implement responsibility within execute() method

Kick implements Command:
Responsibilities:
- Use UserManager to remove an existing User from the system

Implementation:
- implement responsibility within execute() method

Leave implements Command:
Responsibilities:
- Use UserManager to remove the User that called this Command from the system

Implementation:
- implement responsibility within execute() method

CheckPrice implements Command:
Responsibilities:
- Use API to get the price of an Asset

Implementation:
- implement responsibility within execute() method

Downvote implements Command:
Responsibilities:
- Use VoteManager to add an Downvote Vote to an existing transaction

Implementation:
- implement responsibility within execute() method

ViewVote implements Command:
Responsibilities:
- Use VoteManager to view the Votes for an existing Transaction

Implementation:
- Implement responsibility within execute() method
- Option to return Votes for a specific Transaction or all Transactions

ViewTransaction implements Command:
Responsibilities:
- Use TransactionManager to view all the pending Transactions

Implementation:
- Implement responsibility within execute() method
- Option to return Votes for a specific Transaction or all Transactions

Deposit implements Command:
Responsibilities:
- Use TransactionManager to execute deposit money for each User.

Implementation:
- Implement responsibility within execute() method

CreateUser implements Command:
Responsibilities:
- Create a user account using UserManager.

Implementation:
- execute method that executes the functionality of the command
- help method that returns a help string
- name method to return a name for the CommandParser to match for

UserManager:
Responsibilities:
- Track, organize, and manipulate all Users in the system

Implementation:
- Add users, remove users, search for users
- Hashmap<UUID, User>

VoteManager:
Responsibilities:
- Record votes for each existing Transaction

Implementation:
- Create, add, remove, find Votes
- Return which transaction a Vote is for
- Return votes of a transaction
- Hashmap<Transaction, List <Vote>>

AssetManager (Global Portfolio):
Responsibilities:
- Organize and keep track of all Assets currently in the system

Implementation:
- Hashmap<UUID, Asset>
- Add, remove, find Assets
- Prove methods to get information on all Assets, ex. Global Profit, Global 24 hour difference, Asset Diversity

Stock extends Asset:
Responsibilities:
- Provide basic information and api information on the stock asset type.

Implementation:
- Name, symbol, current price, market cap, 24 hour difference, etc
- NASDAQ or NYSE or …
- Store information to pull real time data off yahoo finance or other websites

TranscationManager:
Responsibilities:
- Create, track, manage the Transactions.

Implementation:
- Create, add, delete, find a transaction
- Hashmap<UUID, Transaction>
- Only stores pending Transactions

TranscationHisotry(Pending):
TranscationExecutor:
Responsibilities:
- Execute the Transactions.
- Update the asset price when executing the transaction

Implementation:
- Given a Transaction, get the current price of the asset, execute it.
- Return how much money was made or lost (double type)

VotingPowerAlg:
Responsibilities:
- TBD

Implementation:
- TBD

GUI:
Responsibilities:
- Present a suite of graphs representing the performance of the pooled portfolio in general and the user leaderboard.

Implementation:

- Use a factory design pattern to generate JPanels.

discordConverter:

- Wrap around the CLI class, details tbd.