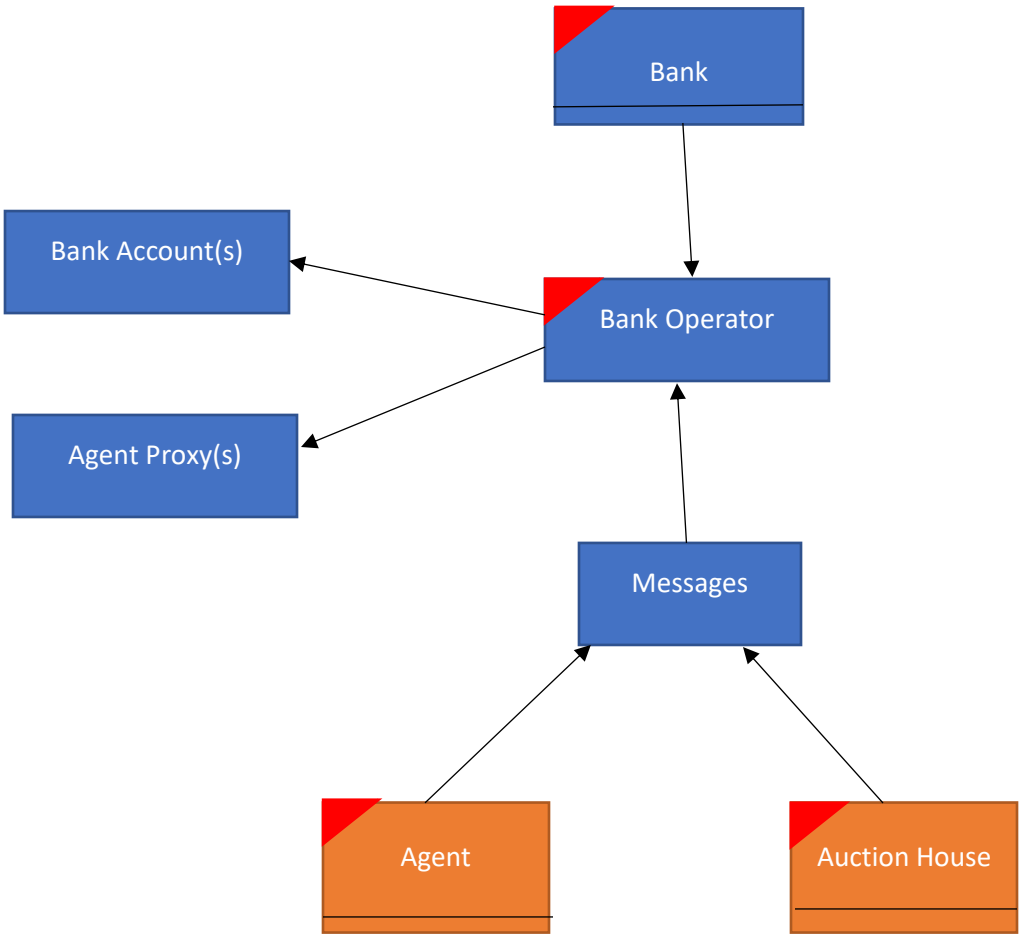
Bank
Bank Account(s)
Bank Operator
Agent Proxy(s)
Messages
Agent
Auction House

Implementation Details:

- Bank: Main Class that starts the whole Bank Server for Agents and Auction Houses to connect. When launched, it will create a thread that will listen to user input to be able to interact and get information about the bank. Afterward, we will continuously listen for new clients trying to connect. Whenever a client connects and we accept the connection, we will create a Bank Operator and make it handle that client. This allows us to process multiple clients at the same time, each in their separate threads. To store the members of the bank, I used a synchronized map (more specifically, a Hash Map) provided by the Collections library that stores their Account ID generated by the Bank Operator as the key and a Bank Account object as the value. This allows me to access the data while having the Map deal with the synchronization details. Auction Houses, denoted as business members in the bank, are also stored in a synchronized map, where the Account ID is the key and the IP addresses of said Auction House is their value. Finally, I keep a list of Agent Proxies used to update all Agents of any changes in the Auction Houses, such as registering or deregistering to the bank.
- BankOperator: The engine of the bank. Given a socket, it will process messages sent through that socket and appropriately respond to them. Whenever a BankOperator is started, we will bind input an output streams and generate a unique random integer up to 5 digits. It does this by using SecureRandom, which is a class used in cryptography to produce non-deterministic random numbers. This, coupled with the fact that we check to see if the id is contained in our members, allows us to get an id that is unique in most cases. Whenever an agent is registered into our bank, we will generate an AgentProxy that will take care of updating agents of changes to our Auction Houses. Whenever an Auction House is registered into our bank, we will update our AgentProxy list with the new information. In both cases, we will create new BankAccount objects and add them to our Bank. Whenever they are deregistered, we will notify the appropriate proxies and remove them from them bank
- BankAccount: Stores all necessary information needed to have a bank account. An ID generated by BankOperator and a balance is given to create it. Provides basic functionality such as changing the value of the balance and adding and removing holds. Holds are stored in an ArrayList and are represented as Integers. Adding and removing holds needs to be synchronized since the bank account does not distinguish between holds and can cause issues if a hold of the same amount is being removed and added at the same time. Getting the balance returns the current available balance considering all holds and does not return the actual balance the account has.
- AgentProxy: Acts as a middleman to send information to the agents. Used only when we need to update multiple agents for a single event, such as when a new Auction House is registered into our Bank or when one is removed from it. Takes an ObjectOutputStream since it will only communicate to the agent and update it.
- Messages: Provides all details to properly communicate between processes. Message takes an Enum determining the type of message it will take and certain variables will be set depending on what the Message Type is.