Assignment 5 Writeup

The goal of this assignment is to simulate secure bank verifications. The user inputs are accepted through command-line arguments, which includes the number of bank clients, the number of verifications, the fraction of invalid messages, and the output file name.

The above user inputs are validated and parsed using a CmdHandler, and are then used to construct a SecureBankVerificationSimulator object.

The job of SecureBankVerificationSimulator is to create the requested number of clients and verifications. The clients are represented by Client objects, and the client requests are represented by Request objects. To process the verifications, the simulator sends the requests to a Bank object, who will verify the requests and return the transaction status. The results of verifications are stored in Result objects. After all verifications have been processed, the results of simulation are save to a csv file using a CsvFileGenerator object, and some "metadata" is also printed to console using a Printer.

Each Client object generated by the simulator contains the client ID, RSA key pair (public and private keys), and provides a method to generate digital signature for a message. In Bank class, the information of each client is stored as a ClientInfo object. One thing to note is that the ClientInfo does not contain private key, but it contains public key as well as deposit limit and withdrawal limit (these two limits are not contained in Client objects). The reason to differentiate Client and ClientInfo is because of the requirement of this assignment that "the bank does not know the private key of clients; the clients do not know their deposit limit / withdrawal limit".

Please see UML diagram in the assignment5 folder (some interfaces are not shown in the diagram for simplicity).

<u>Data collections used:</u>

Map<Integer, IClient> -- Stores the Clients objects in simulator, the key in this map is the client ID, which is unique for each client, and the value is the IClient object. The reason to use a map here is because the simulator needs to access the clients by their ID when generating requests. The most efficient data structure is a map (O(1) time complexity). This can also avoid duplicate IDs during generation of clients.

Map<Integer, IClientInfo> -- Stores the ClientInfo in the Bank class. Same reason as above.

List<IRequest> -- Stores the requests generated by the simulator. A List is used here because there is no specific requirement for a container of requests. As long as the data structure is iterable, the simulator will be able to iterate the requests when sending them to the bank one by one.