Design Patterns

- <u>The factory method pattern</u>

All of objects creation were through factory methods written in factory classes in com.atypon.factory package such as:

- `BlockchainFactory`, for creating a `Blockchain`.
- `BlockFactory`, for creating a `Block`.
- `ClientFactory`, for creating a `Client` and a `ClientSocket`.
- `KeyFactory`, for creating `Public & Private Key` pairs as well as unique IDs for `Transactions`.
- `TransactionFactory`, for creating `Transactions` and `MinedTransactions`.
- `UserFactory`, for creating a `User`.

or inside the class using static methods if the class was a GUI Interface in `com.atypon.gui` package such as:

- `runLogin(…)`, which instantiates and runs the login window.
- `runWindow(…)`, which instantiates and runs the main window.
- `runNewTransaction (…)`, which instantiates and runs the new transaction window.
- `runShowTable (…)`, which instantiates and runs the show table window.

- <u>The chain-of-responsibility pattern</u>

The system was build from bottom to top considering the single responsibility principle, where Transaction was built to hold transaction info, and `MinedTransaction` was built on top of it to hold The `Transcation` and the mining info, The block was built to hold info about a `Blockable` data that is proven of work, the Blockchain is a list of blocks and the User is an API that exposes the blockchain.

- <u>Façade pattern</u>

The design behind the User API is to use the blockchain in an encapsulated environment and expose only the needed functionality to the user, which makes the User/Blockchain a Façade pattern.

- <u>The producer-consumer pattern</u>

Used in the `transactionPool` by using the `BlockingQueue` data structure which uses the producer consumer design pattern, where some user “produces” a transaction and broadcast it to everyone, and the miner will “consume” a transaction and mine it into a valid block then add it to the blockchain and broadcast the change.