Clean Code Principles

- C1: Inappropriate Information

I made sure to include only the important comments, comments about the functionality of the function are included in the documentation of the function, detailed comments are added if needed to clarify how the code is actually doing what.

- C2: Obsolete Comment

Comments that aren’t useful anymore were removed.

- C3: Redundant Comment

Comments are written only when needed to be written, only to explain the main progress of the function or a specific complicated line of code.

- C4: Poorly Written Comment

I made sure I use proper English language to well explain the intension of the code.

- C5: Commented-Out Code

Any commented out code were removed.

- E1: Build Requires More Than One Step

I built the project using Intellij’s build artifact, which is a one click build.

- E2: Tests Require More Than One Step

I wrote tests for all the classes so that testing can be done in one click.

- F1: Too Many Arguments

Most functions don’t have many arguments and are well optimized.

- F2: Output Arguments

Arguments in my functions were all used as inputs.

- F3: Flag Arguments

I avoided using boolean and flag arguments in the functions, except for the setters for boolean fields.

- F4: Dead Function

Dead functions were deleted, all functions are used in my code.

- G1: Multiple Languages in One Source File

All my code is written in Java, so this is not a problem.

- G2: Obvious Behavior Is Unimplemented

The code runs as you would expect it to be, testing proves that.

- <u>G3: Incorrect Behavior at the Boundaries</u>

I tested out the corner cases for the functions and made sure they work in all cases.

- <u>G4: Overridden Safeties</u>

All warnings and suggestions by the compiler were taken into account except for a few that doesn’t fit the project requirements.

- <u>G5: Duplication</u>

In some of the factory methods, the same block of code was used so I used some methods to call the others to make sure I don’t repeat the code, same goes for other places where I used overloading.

- <u>G6: Code at Wrong Level of Abstraction</u>

In Transaction and MinedTransaction, I made sure all the functionality that could be moved to the higher abstraction are in Transaction class.

- <u>G7: Base Classes Depending on Their Derivatives</u>

Transaction is completely separate of MinedTransaction and doesn’t depend on it in any way.

- <u>G8: Too Much Information</u>

I used the simplest GUI I could come up with to run the application.

- <u>G9: Dead Code</u>

I made sure to add try-catch statements and if statements only when necessary.

- <u>G10: Vertical Separation</u>

I tried to arrange the functions and fields inside the classes, so they are easy to track, and as for the local variables, they are always declared in the smallest scope right at their first usage.

- <u>G11: Inconsistency</u>

There is no inconsistency in the behavior or the design, I tried to make the GUI look familiar and the tabs to look the same (All of Show users, Show blockchain and show transaction pool uses the same table layout)

- <u>G12: Clutter</u>

I occasionally kept cleaning up the code from useless pieces of code.

<u>G13: Artificial Coupling</u>

The only things that are coupling are the `User` and the `Client` which the `Client` is actually a part of the `User` as it handles it’s communications.

G16: Obscured Intent

I wrote the code to be expressive so it would be easy to understand the intention of it, by using meaningful names, and splitting a single line complicated code to multiple lines.

G17: Misplaced Responsibility

The code was placed were it would feel natural.

G18: Inappropriate Static

I used static methods were the data wasn't related to a specific object, like the function 'veifyChain' inside class `User`, at some point I needed a function to verify a new blockchain with its sequence of transactions so I couldn't put it in the Blockchain class as it is more generic than just Transactions, and since the blockchain is new, then it's independent from the current user, so I changed the method to be static to server this purpose as well.

G19: Use Explanatory Variables

Explained in G16.

G20: Function Names Should Say What They Do

Function names were carefully chosen so they would do what you would expect from them by reading just their names.

G21: Understand the Algorithm

I wrote the code understanding exactly what should happen and how to handle the tasks at hand.

G24: Follow Standard Conventions

My code follows the standard conventions of Java as explained in the report for "Java Effective Code".

G25: Replace Magic Numbers with Named Constants

All numbers used were changed to constants such as port number for the server and its ip address, number of threads, all the constants regarding the calculations for the transactions.

G26: Be Precise

Decisions regarding types, naming, design, flow of code… etc. were all well though and planned so they fit the best practices.

G28: Encapsulate Conditionals

Conditions are clear to the reader and there is no ambiguity in them.

G29: Avoid Negative Conditionals

Explained in G28.

G30: Functions Should Do One Thing

Functions are written to do only one task.

G33: Encapsulate Boundary Conditions

Explained in G16.

G36: Avoid Transitive Navigation

Transitive navigation was minimized by adding functions to B that A can call which interacts with C.

T1: Insufficient Tests

I tried to write tests for as many scenarios as possible.

T3: Don’t Skip Trivial Tests

Even trivial tests were added to ensure code doesn’t break while building up the system.

T5: Test Boundary Conditions

Explained in T1.

T9: Tests Should Be Fast

The only slow test is the one that runs multiple users and mines up to 120 transactions, otherwise they’re super fast.