SOLID Principles

The Single Responsibility Principle (SPP)

To achieve the SRP, I make sure that every class is responsible for its own actor, for example:

In the server application:

- `FileWriter`: this class saves the file to the storage.
- `FileSocketReader`: this class reads the file from the socket.
- `StringSocketWriter`: this class writes a String value to socket.
- `ClientRunnable`: this class that handles a single connection with a client.
- `ServerThread`: this class which contains the main functionality of the server.

In the client application:

- `MainView`: this class act as View that prints to console
- `AppMainController`: this class acts as an interface between Model and View components to process all the business logic and incoming requests.
- `AppDataManager`: this class manages the whole data on the app.
- `AppFileHelper`: this class used to simplify work with the files.
- `FileReader`: this class reads a file from the storage.
- `FileSocketWriter`: this class writes a stream of bytes to the socket.
- `StringSocketReader`: this class reads a String value from the socket.

The Open-Closed Principle (OCP)

It's not easy to apply this principle to the project due to the requirements.

The Liskov Substitution Principle (LCP)

This principle applies to the project since it allows you to design:

- Any kind of socket reader extends the functionality by writing your own `SocketReader` class and implementing `SocketReader` interface without any unexpected behaviour.
- Any kind of socket writer extends the functionality by writing your own `SocketWriter` class and implementing `SocketWriter` interface without any unexpected behaviour.
- Any kind of files writer by extending the functionality by writing your own `StreamWriter` class and implementing `StreamWriter` interface without any unexpected behaviour.
- Any kind of files reader by extending the functionality by writing your own `StreamReader` class and implementing `StreamReader` interface without any unexpected behaviour.

The Interface Segregation Principle (ISP)

It's not easy to apply this principle to the project due to the requirements.

The Dependency Inversion Principle (DIP)

In the client application:

- The `Gson` object was injected to the `AppFileHelper` by the `Main`.
- The `FileHelper` object was injected to the `AppDataManager` by the `Main`.
- The `DataManager` object was injected to the `AppMainController` by the `Main`.
- The `MainController` object was injected to the `MainView` by the `Main`.