1. **Describe what problem you’re solving?**

The problem we are trying to solve is log processing, where log files have several types of log entries. There are three categories of logs - APM Logs, Application Logs, and Request Logs which contain all types of important metrics and information required to monitor and diagnose applications behaviour and performance. The APM logs contain performance-related metrics, including the usage of the CPU, memory, and disk. Each log entry gives information regarding a specific metric and the value, and this is important in performance alerting and monitoring.The Application Logs contains logs related to its operational state, including error, warning, and informational messages.The Request Logs capture the requests made via HTTP to the application, such as the request method, URL, response status, and response times for supporting the analysis of user interaction and performance of the server response.

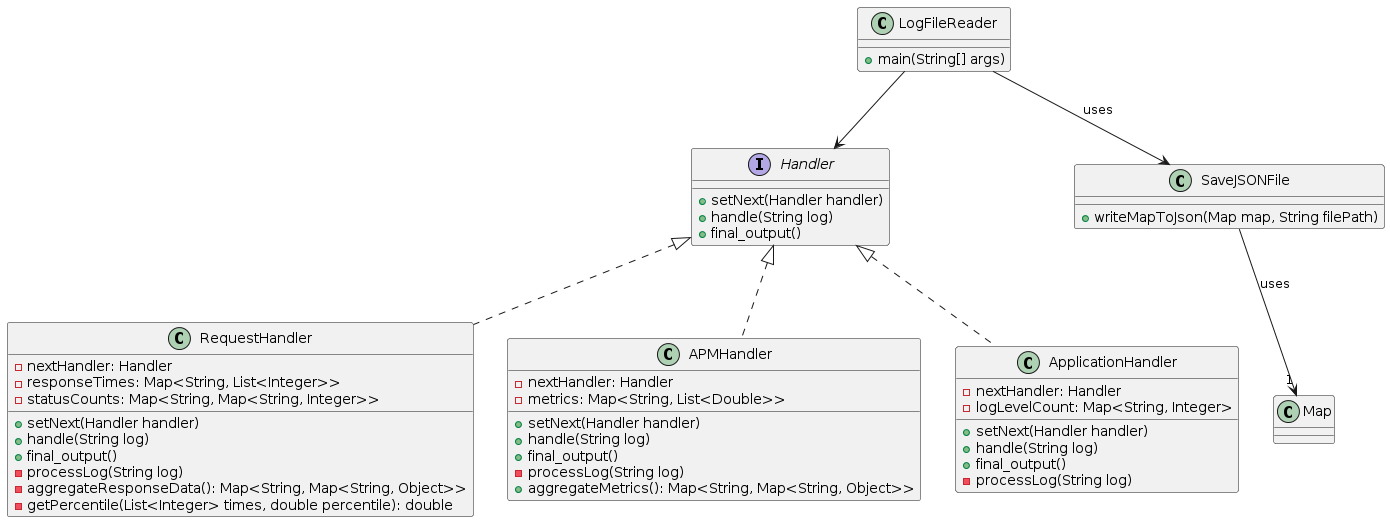
1. **What design pattern(s) will be used to solve this?**

The design pattern I am using here is chains of responsibility. This is easy to streamline the processing and parsing of diverse log entries. This pattern is well-suited for this application because it allows a request to pass through a chain of handlers, each capable of processing a specific type of log. This modular approach makes the system highly extensible and maintainable, as new log types can be easily integrated by adding new handlers without modifying the existing code. Additionally, we can separate the concerns by isolating the processing logic for different log types into different handler classes. This design not only simplifies the management and scalability of the application but also enhances its ability to adapt as new requirements and log formats emerge, making it robust against changes and expansions in log analysis needs.

1. **Describe the consequences of using this/these patterns ?**

Using the Chain of Responsibility pattern in the log file parsing application provides several advantages. It provides a high degree of modularity, allowing each type of log to be handled by a separate handler, which simplifies the code structure and makes it easier to maintain and update. This pattern also enhances the scalability of the application by facilitating the easy addition of new handlers as new log types are introduced, without the need for altering existing code. However, a potential drawback is the performance impact in scenarios where a log entry must pass through multiple handlers before being processed, which could increase processing time, especially in systems with a large number of log types. Additionally, if not managed carefully, the chain can grow complex, making debugging more challenging as the application scales.

1. **Create a class diagram - showing your classes and the Chosen design pattern ?**

****