Individual Project - Log Parser and Aggregator CLI Tool

Name: Rushabh Runwal
SJSU ID# 018176834

Question #1: Describe what problem you're solving.
The problem we are solving is to efficiently process and analyze log files generated by applications, which typically contain a mix of performance metrics, system events, and user requests. These logs are valuable for understanding application health, debugging issues, and monitoring performance, but their raw format makes them difficult to interpret. By categorizing these logs into specific types like APM logs, application logs, and request logs, along with performing aggregations (like calculating minimum, median, average, and counts). The application simplifies the raw data into actionable insights. The output is then written into respective JSON files, making it easy to consume and integrate with other tools or systems. The solution is designed to be scalable and adaptable, supporting new log types and formats in the future.

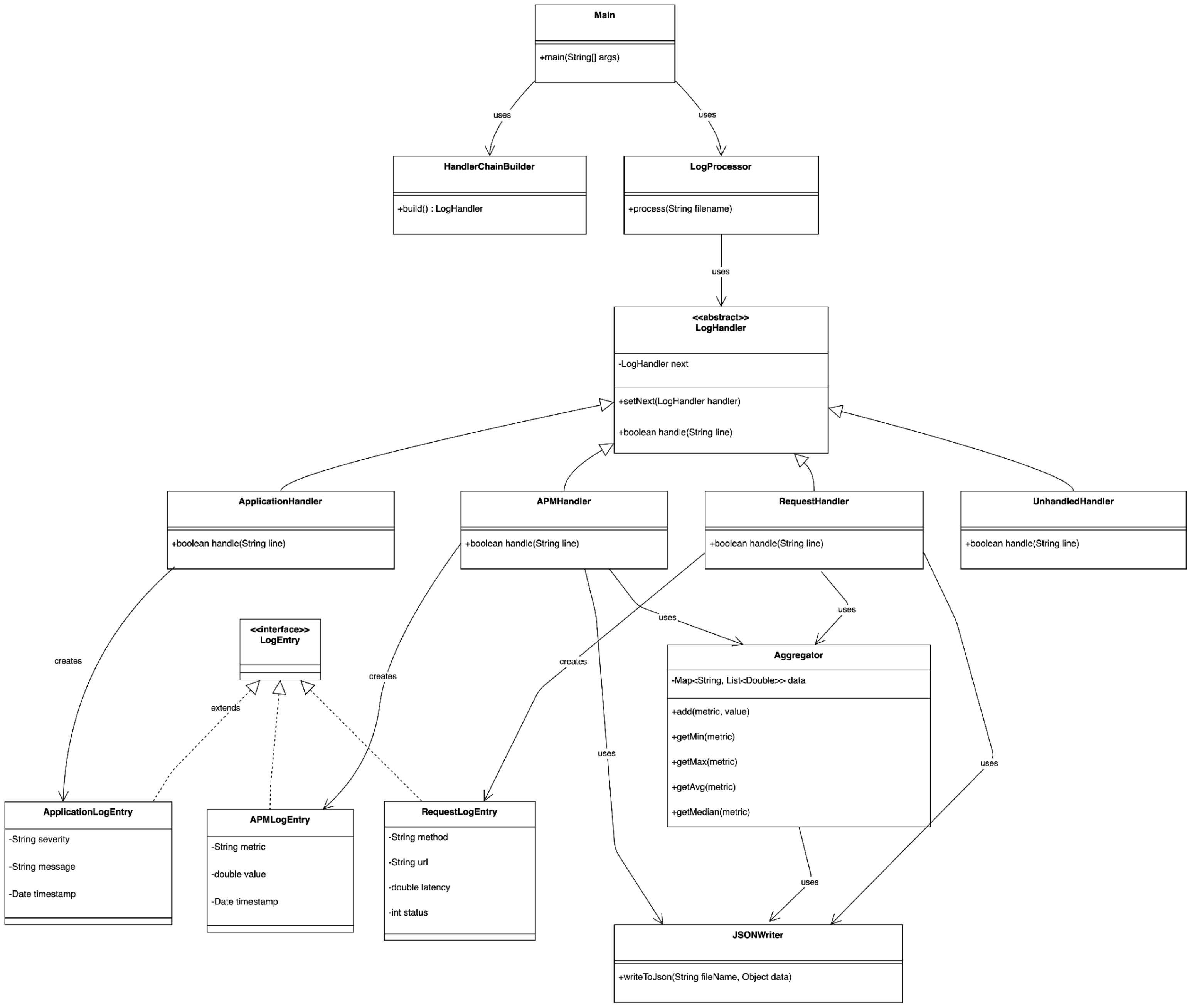
Question #2: What design pattern(s) will be used to solve this?
The **Chain of Responsibility (CoR) Design Pattern** is used to solve this requirement. It helps to process different types of logs (APM, Application, and Request) in the most simple and organized way. Each handler is responsible for handling one type of log, like handling metrics, severity levels, or response times. If a handler can't process a log, it passes it to the next one in the chain. This design makes it easy to:

1. Handle logs dynamically based on their type.
2. Add new log types in the future without changing the existing code.
3. Keep each handler focused on one task, making the system easier to maintain.
4. Safely handle invalid logs by passing them through the chain or flagging them as unhandled.
5. Adapt to new requirements or log formats as the system evolves.
6. Overall, it ensures the log processing is efficient, easy to manage, and ready for future needs.

Question #3: Describe the consequences of using this/these pattern(s).

● **Logs May Be Missed**:
If a log doesn't match any handler, it might remain unprocessed unless specifically flagged.
● **Difficult to Trace**:
Following the path of a log through a long chain of handlers can be challenging, complicating debugging.
● **Performance Impact**:
When the chain grows longer, logs may take more time to be processed as they pass through multiple handlers.

Question #4: Class Diagram:


Main
+main(String[] args)
uses
uses
HandlerChainBuilder
+build() : LogHandler
LogProcessor
+process(String filename)
uses
<<abstract>>
LogHandler
-LogHandler next
+setNext(LogHandler handler)
+boolean handle(String line)
ApplicationHandler
+boolean handle(String line)
APMHandler
+boolean handle(String line)
RequestHandler
+boolean handle(String line)
UnhandledHandler
+boolean handle(String line)
creates
<<interface>>
LogEntry
extends
creates
creates
uses
uses
Aggregator
-Map<String, List<Double>> data
+add(metric, value)
+getMin(metric)
+getMax(metric)
+getAvg(metric)
+getMedian(metric)
uses
uses
ApplicationLogEntry
-String severity
-String message
-Date timestamp
APMLogEntry
-String metric
-double value
-Date timestamp
RequestLogEntry
-String method
-String url
-double latency
-int status
uses
JSONWriter
+writeToJson(String fileName, Object data)