Problems Solved:

The application solves the **analysis and aggregation of log data** into structured statistical summaries, ensuring efficiency and usability. Algorithms are implemented to calculate key metrics like min, max, median etc. for numerical data. It summarizes the result in a file which stores JSON data which can be used as a means for further extension in other services.

Design Patterns Used and respective consequences:

1. Chain of Responsibility Pattern:

- Implementing COR enables us to handle each log type in its required way. For each log type an individual log parser is implemented which has its own logic on how to parse the logs.
- This is done using a LogParser class which is a parent class. Each parser extends it. The chain of command is established by setting a nextParser for each parser. We identify which parser to select by recursively using keywords from a log line and until a parser processes a log line the execution doesn't flow through.
- The key advantage of using this approach is that it enables extensibility to add parsers in the future and decouples the logic to respective log parser class.

2. Builder Pattern (via ObjectNode in Jackson):

- For structured output we use JsonNode to create data in JSON format.
- Using Jackson library's ObjectMapper and ObjectNode to incrementally build JSON objects with statistical metrics.
- Key advantage of code maintainability and extensibility is achieved.

Overall Consequences

The combined use of these patterns results in:

1. **Highly Extensible Design**:
    - Adding new log parsers or metrics is straightforward without impacting existing functionality.
2. **Performance Overhead**:
    - Passing logs through the chain and constructing hierarchical JSON structures may introduce minor processing delays.
3. **Readable and Maintainable Code**:
    - Clear separation of responsibilities (e.g., log processing, statistics calculation, and JSON output).

Class Diagram