1: Benefits of my design:

For the given task, there were two main modules: one for loading the data from a file, and the other for processing the data. I have implemented both of them in two separate classes to make the code more readable and maintainable for future purposes.

Two classes (*PatternHandler*, *PatternFileHandler*) have been created for processing and loading the data. I believe that this design allows for easy extension to accommodate any new type of Pattern. All that's required is the creation of a separate class according to the desired new pattern, along with the addition of a new method in *PatternFileHandler* to return the pattern.

2: Improvement Potential:

I have done my best to design the program, but I am open to feedback and potential improvements if there are any.

3: Assumptions and Trade-offs:

Assumptions:

- **File System Assumption:** The program assumes that the file path provided as a arguments are valid and accessible within the file system.
- **File Format Assumption:** The program assumes that the loaded file is correctly formatted in quartet tuple format, separated by commas ','.

Trade-offs:

- **Class Coupling vs. Encapsulation:** Both classes are tightly coupled as *PatternHandler* is directly dependent on *PatternFileHandler* for file I/O operations. While this coupling simplifies the code structure, it reduces encapsulation and makes it harder to unit test *PatternHandler* independently.

4: Time Complexities of Quries:

- **Retrieve the pattern call by id:** O(n)
- **List all pattern calls by name:** O(n)
- **List all pattern calls by path:** O(n)
- **List all pattern calls by skipped flag:** O(n)
- **List all pattern calls by not skipped flag:** O(n)

5: I did not find any difficulties.