# A brief Report on the Design Choices and Lessons Learnt in Implementing FCFS and SJF Scheduling Algorithms in Java

This report gives a general idea of the design choices and inferences that were made during the implementation of First-Come, First-Served (FCFS) and Shortest-Job-First (SJF) scheduling algorithms in Java. Design of the program is based on basic elements of modularity, reusability and Java standard library features for effective use. Through scrutinizing the underlying design ideas and what we have learned, we tried to pick the essential components that resulted in the success of the scheduling algorithms.

## Design Choices

**Process Class:** The most important feature of this program is inside the Process class which holds the attributes of a process, such as its ID, arrival time, burst time, waiting time and turnaround time. In this class, the entire process is clearly and effectively illustrated, allowing students to examine and make use of the process data.

**Scheduling Algorithms:** The scheduling algorithms (FCFS and SJF) are realized in KennethRichards class as separate methods. This design decision enables us to have a well-defined separation of concerns thus the code becomes modular and easy to maintain. It also reduces complexity during the adding of new scheduling algorithms in the future.

**File Reading:** The program also gets process details from an external file, thus achieving dynamics and ease of further input data adjustments. This decision also comes with highlighting the separation of concerns, as the functionality of the program does not depend on the format of the input data directly.

**Use of Java Collections and Streams:** The implementation uses Java collection libraries (such as arrays) and streams for processing the process data. This method illustrates a contemporary and efficient utilization of the Java standard library for short and readable codes.

**Lessons Learnt**

**Modularity and Reusability:** Program modularity through the principle of separation of concerns has resulted in a program that is easier to understand, maintain, and extend. Every element of the program has a distinct function, and any modifications in one part will not affect the rest.

**Effective Use of Java Collections and Streams:** The usage of Java's collections and streams for data processing and calculation reflected the essence of such features. It also underlined the necessity of knowing and applying the existent standard library of Java in order to write clean and succinct code.

**Challenges in Implementation:** The other challenge was accurately figuring out how to implement the scheduling algorithms, especially in relation to computing the waiting period and turnaround time. The problem was resolved by the scrutiny of the algorithms' logical sense combined with a diligent data processing.

**Testing and Validation:** The other challenge was the correctness of the program, especially in terms of the correct calculation of the average waiting time and turnaround time. This was resolved by including various inputs and validation checks to make sure that the program works as expected.

**Learning from Existing Projects:** The review of projects on GitHub related to scheduling algorithms e.g., the Elevator-Scheduling-Simulator and CPU-scheduling-Algorithms provided insight into the real-world applications and challenges of scheduling algorithms. These insights were very useful in the development and implementation of the program’s design.

In summary, the design considerations and the lessons learnt during the implementation of the FCFS and SJF scheduling algorithms in Java stress the importance of having a modular design, language features used appropriately and thorough testing. It uses a modular design and aims at using the Java standard library where possible, which leads to the program with clarity, maintainability, and also efficiency. Using this information, we can develop further development strategies and continue refining software design and also implementation techniques.