1. Introduction

The objective of this assignment was to design and implement a CPU Process Scheduling Simulator to compare the behavior and performance of different scheduling policies. The simulator models the execution of processes under First-Come, First-Served (FCFS), Shortest Job First (SJF), Priority Scheduling, and Round Robin (RR) algorithms. The application visualizes the scheduling timeline using a Gantt chart and calculates key performance metrics to facilitate a direct comparison of algorithmic efficiency and fairness.

2. Design

The simulator is implemented in **Java** using the **Swing** library for the Graphical User Interface (GUI). The design is modular and object-oriented:

- **Data Structures:**
    - Job Class: Represents a Process Control Block (PCB), storing the Process ID, Arrival Time, Burst Time, Priority, and calculated metrics (Finish, Wait, Turnaround).
    - Queue<Job>: Used in the Round Robin algorithm to manage the ready queue.
    - List<Job>: Used to store the pool of all processes.
- **Key Functions:**
    - parseInputData(): Reads and validates raw text input, converting it into Job objects.
    - startSimulation(): The central controller that clones the job list (to preserve original data) and dispatches it to the selected algorithm.
    - **Scheduling Logic:**
        - **FCFS:** Sorts jobs by arrival time and executes them sequentially.
        - **SJF & Priority:** Utilizes a "ready pool" approach. At any given time t, the simulator checks which processes have arrived (arrival <= t) and selects the optimal one based on Burst Time (SJF) or Priority Level (Priority).
        - **Round Robin:** Maintains a simulation clock and a queue. Processes execute for min(remaining_time, time_quantum). If a process is not finished, it is re-added to the back of the queue. The logic strictly handles new arrivals, ensuring they enter the queue before a preempted process re-enters.
- **Visualization:** A custom VisualizationPanel class uses Java 2D Graphics to draw the Gantt chart dynamically, scaling the bars based on total execution time.

3. Results and Analysis

Sample Input Used:

P1 (0, 8, 3),
P2 (1, 4, 1),
P3 (2, 9, 4),
P4 (3, 5, 2)

Summary Table (Time Quantum =4 for RR):

Algorithm	**Avg Turnaround Time**	**Avg Waiting Time**	**CPU Utilization**
FCFS	15.25	8.75	100.0%
SJF (Non-Preemptive)	14.25	7.75	100.0%
Priority (Non-Preemptive)	14.25	7.75	100.0%
Round Robin (TQ=4)	18.25	11.575	100.0%

4. Discussion

A. Best Algorithm Performance

For this specific input, SJF (Shortest Job First) and Priority Scheduling performed the best, both yielding an Average Waiting Time of 7.75.

- **SJF Advantage:** SJF is mathematically proven to minimize average waiting time because it executes the shortest tasks first. By clearing P2 (Burst 4) and P4 (Burst 5) quickly before the longer P3 (Burst 9), it reduced the "convoy effect," lowering the wait time for the system overall.
- *Note:* In this specific dataset, Priority Scheduling mimicked SJF because the processes with the shortest burst times (P2, P4) also happened to have the highest priorities (1 and 2), resulting in an identical schedule.

B. Round Robin: Performance vs. Fairness

Round Robin involves a trade-off between throughput/turnaround time and fairness/responsiveness.

- **Trade-off:** RR generally has a higher average waiting time than SJF (as seen in the results: 13.50 vs 7.75) because long processes are constantly preempted, delaying their completion. However, it is "fairer" because no process waits indefinitely; every process gets a slice of the CPU.
- **Time Quantum Effect:**
  - **Small Quantum:** Increases responsiveness but increases overhead (context switching).
  - **Large Quantum:** Reduces overhead but makes the algorithm behave like FCFS, potentially causing the convoy effect.
  - In this simulation (TQ=4), the quantum was smaller than most burst times, causing frequent preemption, which increased the turnaround time significantly compared to the non-preemptive approaches.

C. Starvation

- **Definition:** Starvation occurs when a runnable process is overlooked indefinitely by the scheduler because other processes are consistently preferred over it.
- **Vulnerable Algorithms: Priority Scheduling** and **SJF** are susceptible to starvation. In Priority scheduling, a continuous stream of high-priority processes will prevent a low-priority process from ever running.
- **Demonstration:** Using the input below or in the files (./starvation.txt), if we run **Priority Scheduling**, the process P_Low will likely starve (or wait until the very end) if new high-priority jobs keep arriving.

```
P_Low,  0, 100, 10

P_High1, 1, 5, 1

P_High2, 5, 5, 1

P_High3, 10, 5, 1
```

D. Impact of I/O Bursts

In a real-world system, processes alternate between CPU bursts and I/O bursts.

- **SJF vs. RR:** If processes are I/O bound, **Round Robin** is generally superior to non-preemptive SJF.
- **Reasoning:** In non-preemptive SJF, if a process enters a CPU burst, it holds the CPU until it finishes or voluntarily yields (usually for I/O). If a process has a short CPU burst but blocks frequently for I/O, SJF handles it well. However, if the scheduling is *strictly* based on CPU burst estimation without preemption, a process that computes for a long time before doing I/O can block shorter, interactive processes.
- **RR Advantage:** RR ensures that when a process blocks for I/O (or uses up its quantum), another process runs immediately. This overlaps the I/O of one process with the computation of another, maximizing system throughput and device utilization.