Write a program to simulate CPU Scheduling Algorithms: SJF (PREEMPTIVE) in java.

SJF (PREEMPTIVE) :

```
package P1;

import java.util.*;

class Process {
    int pid, arrivalTime, burstTime, remainingTime, completionTime, waitingTime,
turnaroundTime;
    boolean isCompleted;

    public Process(int pid, int arrivalTime, int burstTime) {
        this.pid = pid;
        this.arrivalTime = arrivalTime;
        this.burstTime = burstTime;
        this.remainingTime = burstTime;
        this.isCompleted = false;
    }
}

public class SJFPreemptive {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of processes: ");
        int n = sc.nextInt();

        Process[] processes = new Process[n];
        for (int i = 0; i < n; i++) {
            System.out.print("Enter arrival time for process " + (i + 1) + ": ");
            int arrivalTime = sc.nextInt();
            System.out.print("Enter burst time for process " + (i + 1) + ": ");
            int burstTime = sc.nextInt();
            processes[i] = new Process(i + 1, arrivalTime, burstTime);
        }

        Arrays.sort(processes, Comparator.comparingInt(p -> p.arrivalTime));

        int currentTime = 0, completed = 0;
        double totalWaitingTime = 0, totalTurnaroundTime = 0;


System.out.println("\nPID\tArrival\tBurst\tCompletion\tWaiting\tTurnaround");

        while (completed < n) {
            Process current = null;
            for (Process p : processes) {
                if (!p.isCompleted && p.arrivalTime <= currentTime) {
                    if (current == null || p.remainingTime <
current.remainingTime) {
                        current = p;
```

```
                    }
                }
            }

            if (current != null) {
                current.remainingTime--;
                currentTime++;

                if (current.remainingTime == 0) {
                    current.completionTime = currentTime;
                    current.turnaroundTime = current.completionTime -
current.arrivalTime;
                    current.waitingTime = current.turnaroundTime -
current.burstTime;
                    current.isCompleted = true;
                    completed++;
                    totalWaitingTime += current.waitingTime;
                    totalTurnaroundTime += current.turnaroundTime;

                    System.out.printf("%d\t%d\t%d\t%d\t\t%d\t%d\n", current.pid,
current.arrivalTime, current.burstTime,
                            current.completionTime, current.waitingTime,
current.turnaroundTime);
                }
            } else {
                currentTime++;
            }
        }

        System.out.printf("\nAverage Waiting Time: %.2f", totalWaitingTime / n);
        System.out.printf("\nAverage Turnaround Time: %.2f", totalTurnaroundTime /
n);

        sc.close();
    }
}
```

OUTPUT:

```
Enter number of processes: 4
Enter arrival time for process 1: 0
Enter burst time for process 1: 8
Enter arrival time for process 2: 1
Enter burst time for process 2: 4
Enter arrival time for process 3: 2
Enter burst time for process 3: 9
Enter arrival time for process 4: 3
Enter burst time for process 4: 5

PID    Arrival Burst Completion  Waiting Turnaround
2      1         4      5         0        4
4      3         5       10        2        7
1      0         8      17        9        17
3      2         9      26        15       24

Average Waiting Time: 6.50
Average Turnaround Time: 13.00
```

Explanation

- **Process Class**: Represents each process with attributes like `pid`, `arrivalTime`, `burstTime`, `priority`, `waitingTime`, `turnaroundTime`, and `completionTime`.
- **Input**: The program prompts the user to enter the number of processes, and for each process, its arrival time, burst time, and priority.
- **Sorting**: Processes are sorted based on their arrival times, and in case of a tie, by their priority values (lower priority value indicates higher priority).
- **Scheduling**:
    - The program iterates through the processes, selecting the one with the highest priority (lowest priority value) that has arrived and is not yet completed.
    - It calculates the completion time, waiting time, and turnaround time for each process.

- **Averages**: The program calculates and displays the average waiting time and turnaround time for all processes.