Name : Prasad Lolage
Roll No.:3202015
Assignment No.: 3 SPOS

```
import java.util.*;

public class Main {     public static void
main(String[] args) {         Scanner sc =
new Scanner(System.in);
        List<Process> processes = new ArrayList<>();

        System.out.print("Enter number of processes: ");
        int n = sc.nextInt();

        for (int i = 0; i < n; i++) {
            System.out.println("Enter details for Process " + (i + 1) + ":");
            System.out.print("Arrival Time: ");
int at = sc.nextInt();
            System.out.print("Burst Time: ");
            int bt = sc.nextInt();
            System.out.print("Priority (Lower = Higher Priority): ");
int pr = sc.nextInt();

            processes.add(new Process(i + 1, at, bt, pr));
        }

        List<Process> copyForPriority = deepCopy(processes);
        List<Process> copyForRR = deepCopy(processes);

        PriorityNonPreemptive.schedule(copyForPriority);

        System.out.print("\nEnter Time Quantum for Round Robin: ");
int quantum = sc.nextInt();

        RoundRobinPreemptive.schedule(copyForRR, quantum);

        sc.close();
    }

    private static List<Process> deepCopy(List<Process> original) {
        List<Process> copy = new ArrayList<>();
        for (Process p : original) {
            copy.add(new Process(p.pid, p.arrivalTime, p.burstTime, p.priority));
```

```
        }
        return copy;
    }}
import java.util.*;

public class PriorityNonPreemptive {
    public static void schedule(List<Process> processes) {
int time = 0;
        List<Process> completed = new ArrayList<>();
        List<Process> readyQueue = new ArrayList<>();

        processes.sort(Comparator.comparingInt(p -> p.arrivalTime));
int n = processes.size();        int index = 0;

        while (completed.size() < n) {
            while (index < n && processes.get(index).arrivalTime <= time) {
readyQueue.add(processes.get(index++));
            }

            if (readyQueue.isEmpty()) {
                time++;
                continue;
            }

            Process next = Collections.min(readyQueue, Comparator.comparingInt(p ->
p.priority));            readyQueue.remove(next);

            time += next.burstTime;            next.completionTime = time;
next.turnaroundTime = next.completionTime - next.arrivalTime;
next.waitingTime = next.turnaroundTime - next.burstTime;

            completed.add(next);
        }

        printResults(completed);
    }

    private static void printResults(List<Process> processes) {
System.out.println("\nPriority Scheduling (Non-Preemptive):");
System.out.println("PID\tAT\tBT\tPriority\tCT\tTAT\tWT");        for
(Process p : processes) {
            System.out.printf("%d\t%d\t%d\t%d\t\t%d\t%d\t%d\n",
p.pid, p.arrivalTime, p.burstTime, p.priority,
                    p.completionTime, p.turnaroundTime, p.waitingTime);
```

```
      }
   }
}public class Process {
   int pid;     int
arrivalTime;     int
burstTime;     int
remainingTime;     int
priority;     int
completionTime;     int
waitingTime;     int
turnaroundTime;
   public Process(int pid, int arrivalTime, int burstTime, int priority) {
this.pid = pid;         this.arrivalTime = arrivalTime;
this.burstTime = burstTime;         this.remainingTime = burstTime;
this.priority = priority;
   }
}
import java.util.*;

public class RoundRobinPreemptive {
   public static void schedule(List<Process> processes, int quantum) {
      Queue<Process> queue = new LinkedList<>();
      int time = 0;         int n
= processes.size();
      int completed = 0;

      processes.sort(Comparator.comparingInt(p -> p.arrivalTime));
int index = 0;

      while (completed < n) {
         while (index < n && processes.get(index).arrivalTime <= time) {
queue.add(processes.get(index++));
         }

         if (queue.isEmpty()) {
            time++;
continue;
         }

         Process current = queue.poll();

         int executeTime = Math.min(quantum, current.remainingTime);
current.remainingTime -= executeTime;
         time += executeTime;
```

```
        while (index < n && processes.get(index).arrivalTime <= time) {
queue.add(processes.get(index++));
        }

        if (current.remainingTime == 0) {               current.completionTime = time;
current.turnaroundTime = current.completionTime - current.arrivalTime;
current.waitingTime = current.turnaroundTime - current.burstTime;
completed++;
        } else {
           queue.add(current); // Put back in queue
        }
      }

      printResults(processes);
   }

   private static void printResults(List<Process> processes) {
System.out.println("\nRound Robin Scheduling (Preemptive):");
      System.out.println("PID\tAT\tBT\tCT\tTAT\tWT");
for (Process p : processes) {
        System.out.printf("%d\t%d\t%d\t%d\t%d\t%d\n",
p.pid, p.arrivalTime, p.burstTime,
              p.completionTime, p.turnaroundTime, p.waitingTime);
      }
   }
}
```

Output:

```
PS E:\SOPS3> javac *.java
PS E:\SOPS3> java Main Enter
number of processes: 4 Enter
details for Process 1:
Arrival Time: 1
Burst Time: 5
Priority (Lower = Higher Priority): 1
Enter details for Process 2:
Arrival Time: 2
Burst Time: 5
Priority (Lower = Higher Priority): 2
Enter details for Process 3:
Arrival Time: 3
Burst Time: 9
```

Priority (Lower = Higher Priority): 4 Enter
details for Process 4:
Arrival Time: 4
Burst Time: 8
Priority (Lower = Higher Priority): 5

Priority Scheduling (Non-Preemptive):

PID	AT	BT	Priority	CT	TAT	WT
1	1	5	1	6	5	0
2	2	5	2	11	9	4
3	3	9	4	20	17	8
4	4	8	5	28	24	16

Enter Time Quantum for Round Robin: 2

Round Robin Scheduling (Preemptive):

PID	AT	BT	CT	TAT	WT
1	1	5	16	15	10
2	2	5	19	17	12
3	3	9	28	25	16
4	4	8	27	23	15

PS E:\SOPS3>