Spos 3

```
import java.util.*;

public class Main {

    static class Process {
        int pid, at, bt, rt, pr;
        int ct, tat, wt;

        Process(int pid, int at, int bt, int pr) {
            this.pid = pid;
            this.at = at;
            this.bt = bt;
            this.rt = bt;
            this.pr = pr;
        }

        Process copy() {
            return new Process(pid, at, bt, pr);
        }
    }

    static void print(List<Process> p) {
        System.out.println("PID\tAT\tBT\tCT\tTAT\tWT");
        for (Process x : p) {

System.out.println(x.pid+"\t"+x.at+"\t"+x.bt+"\t"+x.ct+"\t"+x.tat+"\t"+x.wt);
        }
    }

    // ======================== FCFS ========================
    static void fcfs(List<Process> list) {
        list.sort((a, b) -> a.at - b.at);

        int time = 0;
        for (Process p : list) {
            if (time < p.at) time = p.at;
            time += p.bt;
            p.ct = time;
            p.tat = p.ct - p.at;
            p.wt = p.tat - p.bt;
        }

        System.out.println("\nFCFS:");
        print(list);
    }

    // ======================== SJF PREEMPTIVE ========================
    static void sjf(List<Process> list) {
```

```
        int n = list.size();
        int time = 0, done = 0;

        while (done < n) {
            Process best = null;

            for (Process p : list) {
                if (p.at <= time && p.rt > 0) {
                    if (best == null || p.rt < best.rt)
                        best = p;
                }
            }

            if (best == null) {
                time++;
                continue;
            }

            best.rt--;
            time++;

            if (best.rt == 0) {
                done++;
                best.ct = time;
                best.tat = best.ct - best.at;
                best.wt = best.tat - best.bt;
            }
        }

        System.out.println("\nSJF Preemptive:");
        print(list);
    }

    // ======================== PRIORITY NON-PREEMPTIVE
========================
    static void priorityNP(List<Process> list) {
        list.sort((a, b) -> a.at - b.at);

        int n = list.size(), done = 0, time = 0;

        while (done < n) {
            Process best = null;

            for (Process p : list) {
                if (p.at <= time && p.ct == 0) {
                    if (best == null || p.pr < best.pr)
                        best = p;
                }
            }

            if (best == null) {
```

```
            time++;
            continue;
        }

        time += best.bt;
        best.ct = time;
        best.tat = best.ct - best.at;
        best.wt = best.tat - best.bt;
        done++;
    }

    System.out.println("\nPriority Non-Preemptive:");
    print(list);
}

// ======================= ROUND ROBIN =======================
static void rr(List<Process> list, int q) {
    Queue<Process> qn = new LinkedList<>();
    list.sort((a, b) -> a.at - b.at);

    int time = 0, done = 0, i = 0, n = list.size();

    while (done < n) {

        while (i < n && list.get(i).at <= time)
            qn.add(list.get(i++));

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

        Process p = qn.poll();
        int run = Math.min(q, p.rt);

        p.rt -= run;
        time += run;

        while (i < n && list.get(i).at <= time)
            qn.add(list.get(i++));

        if (p.rt == 0) {
            done++;
            p.ct = time;
            p.tat = p.ct - p.at;
            p.wt = p.tat - p.bt;
        } else {
            qn.add(p);
        }
    }
```

```
        System.out.println("\nRound Robin:");
        print(list);
    }

    // ======================= MAIN ======================
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

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

        List<Process> input = new ArrayList<>();

        for (int i = 1; i <= n; i++) {
            System.out.println("Process " + i);
            System.out.print("AT: "); int at = sc.nextInt();
            System.out.print("BT: "); int bt = sc.nextInt();
            System.out.print("Priority: "); int pr = sc.nextInt();

            input.add(new Process(i, at, bt, pr));
        }

        // make copies for all algorithms
        List<Process> l1 = new ArrayList<>();
        List<Process> l2 = new ArrayList<>();
        List<Process> l3 = new ArrayList<>();
        List<Process> l4 = new ArrayList<>();

        for (Process p : input) {
            l1.add(p.copy());
            l2.add(p.copy());
            l3.add(p.copy());
            l4.add(p.copy());
        }

        fcfs(l1);
        sjf(l2);
        priorityNP(l3);

        System.out.print("\nEnter Quantum: ");
        int q = sc.nextInt();
        rr(l4, q);
    }
}
```

spos 4

```
import java.util.*;

public class Main {

    // LRU Page Replacement
    static void lru(int[] pages, int frames) {

        System.out.println("\nLRU Page Replacement:");

        int[] memory = new int[frames];
        Arrays.fill(memory, -1);

        int[] lastUsed = new int[frames];
        int pageFaults = 0;
        int time = 0;

        for (int page : pages) {
            time++;
            boolean found = false;

            // check if present
            for (int i = 0; i < frames; i++) {
                if (memory[i] == page) {
                    found = true;
                    lastUsed[i] = time;
                    break;
                }
            }

            // if not found -> page fault
            if (!found) {
                int lruIndex = 0;
                int minTime = lastUsed[0];

                // find least recently used frame
                for (int i = 1; i < frames; i++) {
                    if (lastUsed[i] < minTime) {
                        minTime = lastUsed[i];
                        lruIndex = i;
                    }
                }

                memory[lruIndex] = page;
                lastUsed[lruIndex] = time;
                pageFaults++;

                System.out.println("Page " + page + " caused a fault.");
            }
```

```
    }

    System.out.println("Total Page Faults = " + pageFaults);
}


// Optimal Page Replacement
static void optimal(int[] pages, int frames) {

    System.out.println("\nOptimal Page Replacement:");

    int[] memory = new int[frames];
    Arrays.fill(memory, -1);

    int pageFaults = 0;

    for (int i = 0; i < pages.length; i++) {

        int page = pages[i];
        boolean found = false;

        // check if in memory
        for (int j = 0; j < frames; j++) {
            if (memory[j] == page) {
                found = true;
                break;
            }
        }

        if (!found) {
            int indexToReplace = -1;
            int farthestUse = -1;

            // find optimal frame to replace
            for (int j = 0; j < frames; j++) {

                int k;
                for (k = i + 1; k < pages.length; k++) {
                    if (pages[k] == memory[j]) break;
                }

                if (k > farthestUse) {
                    farthestUse = k;
                    indexToReplace = j;
                }
            }

            memory[indexToReplace] = page;
            pageFaults++;

            System.out.println("Page " + page + " caused a fault.");
```

```
        }
    }

    System.out.println("Total Page Faults = " + pageFaults);
}


// FIFO Page Replacement
static void fifo(int[] pages, int frames) {

    System.out.println("\nFIFO Page Replacement:");

    int[] memory = new int[frames];
    Arrays.fill(memory, -1);

    int pageFaults = 0;
    int pointer = 0;   // points to the next frame to replace

    for (int page : pages) {

        boolean found = false;

        // check if present
        for (int i = 0; i < frames; i++) {
            if (memory[i] == page) {
                found = true;
                break;
            }
        }

        if (!found) {
            memory[pointer] = page;
            pointer = (pointer + 1) % frames;
            pageFaults++;

            System.out.println("Page " + page + " caused a fault.");
        }
    }

    System.out.println("Total Page Faults = " + pageFaults);
}


public static void main(String[] args) {

    int[] pages = {1, 2, 3, 4, 2, 1, 5, 2, 1, 6, 7, 8, 1};
    int frames = 3;

    System.out.println("Page Reference String: " + Arrays.toString(pages));
    System.out.println("Number of Frames: " + frames);
```

```
        fifo(pages, frames);
        lru(pages, frames);
        optimal(pages, frames);
    }
}
```