import java.util.Scanner;

class Process {

int pid; // Process ID

int arrivalTime; // Arrival time

int burstTime; // Burst time

int remainingTime; // Remaining time (used for Round Robin)

int completionTime; // Completion time

int waitingTime; // Waiting time

int turnAroundTime; // Turnaround time

public Process(int pid, int arrivalTime, int burstTime) {

this.pid = pid;

this.arrivalTime = arrivalTime;

this.burstTime = burstTime;

this.remainingTime = burstTime; // For Round Robin

}

}

public class FCFS\_ROUNDROBIN {

// FCFS Algorithm

public static void fcfsScheduling(Process[] processes) {

int n = processes.length;

// Sort by Arrival Time

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (processes[j].arrivalTime > processes[j + 1].arrivalTime) {

Process temp = processes[j];

processes[j] = processes[j + 1];

processes[j + 1] = temp;

}

}

}

int currentTime = 0;

for (Process p : processes) {

if (currentTime < p.arrivalTime) {

currentTime = p.arrivalTime; // Wait for the process to arrive

}

p.completionTime = currentTime + p.burstTime;

currentTime = p.completionTime;

p.turnAroundTime = p.completionTime - p.arrivalTime;

p.waitingTime = p.turnAroundTime - p.burstTime;

}

System.out.println("\nFCFS Scheduling:");

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

for (Process p : processes) {

System.out.println(p.pid + "\t" + p.arrivalTime + "\t" + p.burstTime + "\t" +

p.completionTime + "\t\t" + p.turnAroundTime + "\t\t" + p.waitingTime);

}

}

// Round Robin Algorithm

public static void roundRobinScheduling(Process[] processes, int timeQuantum) {

int n = processes.length;

int currentTime = 0;

boolean allProcessesCompleted;

do {

allProcessesCompleted = true;

for (Process p : processes) {

if (p.remainingTime > 0) {

allProcessesCompleted = false; // At least one process is not completed

if (p.remainingTime > timeQuantum) {

currentTime += timeQuantum;

p.remainingTime -= timeQuantum;

} else {

currentTime += p.remainingTime;

p.remainingTime = 0;

p.completionTime = currentTime;

p.turnAroundTime = p.completionTime - p.arrivalTime;

p.waitingTime = p.turnAroundTime - p.burstTime;

}

}

}

} while (!allProcessesCompleted);

System.out.println("\nRound Robin Scheduling (Time Quantum = " + timeQuantum + "):");

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

for (Process p : processes) {

System.out.println(p.pid + "\t" + p.arrivalTime + "\t" + p.burstTime + "\t" +

p.completionTime + "\t\t" + p.turnAroundTime + "\t\t" + p.waitingTime);

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Input for number of processes

System.out.print("Enter number of processes: ");

int n = sc.nextInt();

// Array of Process objects

Process[] processes = new Process[n];

// Input for process details

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);

}

// Input for scheduling choice

System.out.println("\nChoose Scheduling Algorithm:");

System.out.println("1. FCFS");

System.out.println("2. Round Robin");

System.out.print("Enter your Choice: "); // Fixed missing semicolon

int choice = sc.nextInt();

// Switch-case to handle algorithm choice

switch (choice) {

case 1:

fcfsScheduling(processes);

break;

case 2:

// Input for time quantum

System.out.print("Enter time quantum for Round Robin: ");

int timeQuantum = sc.nextInt();

roundRobinScheduling(processes, timeQuantum);

break;

default:

System.out.println("Invalid choice! Please enter 1 or 2.");

}

sc.close();

}

}