Round RIbbion

```
import java.util.*;

class Process {
   int id, at, bt, rt, ct, tat, wt;
}

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

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

      System.out.print("Enter the time quantum: ");
      int tq = sc.nextInt();

      Process[] p = new Process[50];
      for (int i = 0; i < n; i++) {
         p[i] = new Process();
         p[i].id = i + 1;
         System.out.print("Enter arrival time of Process " + (i + 1) + ": ");
         p[i].at = sc.nextInt();
         System.out.print("Enter burst time of Process " + (i + 1) + ": ");
         p[i].bt = sc.nextInt();
         p[i].rt = p[i].bt;
         p[i].ct = p[i].tat = p[i].wt = 0;
      }

      // Sort by Arrival Time (keep stable order)
```

```
for (int i = 0; i < n - 1; i++) {
   int m = i;
   for (int j = i + 1; j < n; j++) {
      if (p[j].at < p[m].at)
         m = j;
   }
   if (m != i) {
      Process temp = p[i];
      p[i] = p[m];
      p[m] = temp;
   }
}

Queue<Integer> q = new LinkedList<>();
boolean[] marked = new boolean[50];
int cur = 0, done = 0;
double sumWT = 0, sumTAT = 0;

q.add(0);
marked[0] = true;

System.out.println("\n===== Round Robin Scheduling Simulation =====");

while (done < n) {
   if (q.isEmpty()) {
      // CPU idle → find next available process
      for (int i = 0; i < n; i++) {
         if (p[i].rt > 0) {
            q.add(i);
            marked[i] = true;
            break;
```

```
        }
    }
}

int i = q.poll();

if (cur < p[i].at)
    cur = p[i].at; // wait until process arrives

int run = (p[i].rt > tq) ? tq : p[i].rt;
p[i].rt -= run;
cur += run;

// Add newly arrived processes up to current time
for (int k = 0; k < n; k++) {
    if (!marked[k] && p[k].at <= cur && p[k].rt > 0) {
        q.add(k);
        marked[k] = true;
    }
}

if (p[i].rt > 0) {
    q.add(i); // not finished yet → requeue
} else {
    p[i].ct = cur;
    p[i].tat = p[i].ct - p[i].at;
    p[i].wt = p[i].tat - p[i].bt;
    sumWT += p[i].wt;
    sumTAT += p[i].tat;
    done++;
}
```

```
        }

        // Display results
        System.out.println("\n================================================");
        System.out.println("PID\tAT\tBT\tCT\tTAT\tWT");
        System.out.println("================================================");
        for (int i = 0; i < n; i++) {
            System.out.println("P" + p[i].id + "\t" + p[i].at + "\t" + p[i].bt + "\t" +
                    p[i].ct + "\t" + p[i].tat + "\t" + p[i].wt);
        }
        System.out.println("================================================");
        System.out.printf("Average Turnaround Time: %.2f\n", (sumTAT / n));
        System.out.printf("Average Waiting Time:    %.2f\n", (sumWT / n));

        sc.close();
    }
}
```