Avrostningsuppgift

U 1.

```
public class Job {
    private String name;
    private int time;

    /** Skapar ett jobb med namnet name som tar tiden time att utföra. */
    public Job(String name, int time) {
        this.name = name;
        time = time;
    }

    /** Returnerar jobbets tidsåtgång. */
    public int getTime() {
        return time;
    }

    /** Returnerar en sträng som representerar jobbet på formen
        namn (tidsåtgång). */
    public String toString() {
        return name  + " (" + time + ")";
    }
}

public class Machine {
    private int nbr;
    private ArrayList<Job> jobs;
    private int totalTime;

    /** Skapar maskin nr nbr. */
    public Machine(int nbr) {
        this.nbr = nbr;
        jobs = new ArrayList<Job>();
        totalTime = 0;
    }

    /** Tar reda på maskinens nr. */
    public int getNbr() {
        return nbr;
    }


    /** Tilldelar maskinen jobbet j. */
    public void assignJob(Job j) {
        jobs.add(j);
        totalTime += j.getTime();
    }

    /** Tar bort alla jobb från maskinen. */
    public void clearJobs() {
        jobs.clear();
        totalTime = 0;
    }

    /** Tar bort och returnerar nästa jobb som maskinen ska utföra.
         Returnerar null om maskinen inte har några jobb. */
    public Job getNextJob() {
        if (jobs.isEmpty()) {
            return null;
        }
```

```
            totalTime -= jobs.get(0).getTime();
            return jobs.remove(0);
        }


        /** Tar reda på den totala tiden för maskinens jobb. */
        public int getTotalTime() {
            return totalTime;
        }

        /** Returnerar en sträng som innehåller maskinens nr samt maskinens
            schemalagda jobb inom [] med kommatecken mellan. */
        public String toString() {
            StringBuilder b = new StringBuilder();
            b.append(nbr);
            b.append(' ');
            b.append('[');
            for (Job j : jobs) {
                b.append(j);
                b.append(", ");
            }
            if (jobs.size() != 0) {
                b.deleteCharAt(b.length() - 1);
                b.deleteCharAt(b.length() - 1);
            }
            b.append(']');
            return b.toString();
        }
    }

    public class Scheduler {
        private Machine[] machines;

        /** Skapar ett schemaläggare för maskinerna i vektorn machines. */
        public Scheduler(Machine[] machines) {
            this.machines = machines;
        }

        /* Returnerar den maskin som har minst att göra. */
        private Machine machineWithLeastToDo() {
            int min = Integer.MAX_VALUE;
            int minPos = -1;
            for (int i = 0; i < machines.length; i++) {
                int totalTime = machines[i].getTotalTime();
                if (totalTime < min) {
                    min = totalTime;
                    minPos = i;
                }
            }
            return machines[minPos];
        }

        /** Fördelar jobben i listan jobs på maskinerna.
            Jobben är sortrade är sorterad efter avtagande tidsåtgång. */
        public void makeSchedule(ArrayList<Job> jobs) {
            for(int i = 0; i < machines.length; i++) {
                machines[i].clearJobs();
            }
            for (Job j : jobs) {
                Machine m = machineWithLeastToDo();
                m.assignJob(j);
            }
```

```
        }

        /** Skriver ut maskinernas scheman. */
        public void printSchedule() {
            for (int i = 0; i < machines.length; i++) {
                System.out.println(machines[i]);
            }
        }
    }
```