CS1020 Sit-in Lab 03 B - Company Employee Management

Semester 2 AY2013/2014

Requirement

Use Java Stack and/or LinkedList to help you in solving this sit-in lab. **Not using Stack or LinkedList will result in correctness marks being halved !**

Problem Description - Part A

In this problem you are asked to write a program to simulate the employee management in a company. In employee management, the company can **hire** a new employee or **fire** some employee(s). In hiring a new employee, the employee can belong to different ranks, with rank 1 being the lowest rank and rank 5 being the highest rank.

The **hire** operation is as follows:

1. hire <name> <rank>
   (a) *name* is the name of the employee being hired and it is a unique single word name.
   (b) *rank* is the employee's rank, an integer between 1 and 5.

In firing employee(s), the company will specify the number of employees to be fired and proceeding from **lowest rank to highest rank** start firing from **most recently hired to least recently hired** at each rank.

The **fire** operation is as follows:

1. fire <number>
   (a) *number* is the number of employees to fire, an integer ≥ 1 and $\leq$ total number of employees in the company.

Input

The 1st line in the input is an integer **N** (**N** ≥ 1), indicating the number of operations. The next N lines are the N operations given in **chronological order**. An example input is given below:

```
6              <-- 6 operations
hire John 1    <-- John is hired at rank 1
hire Mark 4    <-- Mark is hired at rank 4
hire Betty 3   <-- Betty is hired at rank 3
hire Abel 4    <-- Abel is hired at rank 4
fire 2         <-- fire 2 employees. In this case they are John and Betty
hire Jack 2    <-- Jack is hired at rank 2
```

Output

For each operation, print out the operation on a line, followed by the employees in the company. For the employees, print them from **lowest rank** to the **highest rank**, and within each rank from **most recently hired** to **least recently hired**. Print out the name of the employee followed by his/her rank separated by a single WHITE SPACE. If there are no employees left in the company print @. Output for each operation should be followed by an empty line. Example output for the given input above is shown below:

```
hire John 1
John 1

hire Mark 4
John 1
Mark 4

hire Betty 3
John 1
Betty 3
Mark 4

hire Abel 4
John 1
Betty 3
Abel 4
Mark 4

fire 2
Abel 4
Mark 4

hire Jack 2
Jack 2
Abel 4
Mark 4
```

Skeleton Program

Your program is to be named **EmpManagement.java** (do not change this name). A skeleton program is provided, but you can change it any way or add new class(es) and method(s) as you deem fit. **Write all your code in EmpManagement.java, do not create new files!** The skeleton program is as follows:

```
import java.util.*;

/* Service class representing an employee */
class Employee {

}

/* Client class to simulate the employee management system */
public class EmpManagement {
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
        }
}
```

Testing your program

The following **sample** input and output files are in your plab account:

```
EmpManagementTC1.in, EmpManagementTC2.in, EmpManagementTC3.in, EmpManagementTC4.in
EmpManagementTC1.out, EmpManagementTC2.out, EmpManagementTC3.out, EmpManagementTC4.out
```

EmpManagementTC1.in and EmpManagementTC2.in are input test cases for **part A**, EmpManagementTC3.in and EmpManagementTC4.in are input test cases for **part B**. EmpManagementTC1.out, EmpManagementTC2.out, EmpManagementTC3.out and EmpManagementTC4.out are the expected output for the respective test cases.
After you have compiled your program, to test it with say EmpManagementTC1.in, you type:

```
java EmpManagement < EmpManagementTC1.in
```

Please complete part A before starting on part B !

Problem Description - Part B

In addition to the hire and fire operation, the company can also **promote** employees to a higher rank. In a promotion, the employee will become the **most recent** employee at his new rank.

The **promote** operation is as follows:

1. promote <name> <rank>
   (a) *name* is the name of the employee to be promoted. Employee is guaranteed to exist.
   (b) *rank* is his new rank. This is guaranteed to be higher than his old rank.

Input

Input is the same as before except now there is an additional promote operation. An example input with the additional operations is given below:

```
6              <-- 6 operations
hire John 1    <-- John is hired at rank 1
hire Mark 4    <-- Mark is hired at rank 4
hire Betty 3   <-- Betty is hired at rank 3
promote John 5 <-- John is promoted to rank 5
hire Abel 4    <-- Abel is hired at rank 4
fire 1         <-- fire 1 employee. In this case it is Betty
```

Output

Same as before. Example output for the given input above is shown below:

```
hire John 1
John 1

hire Mark 4
John 1
Mark 4

hire Betty 3
John 1
Betty 3
Mark 4

promote John 5
Betty 3
Mark 4
John 5

hire Abel 4
Betty 3
Abel 4
Mark 4
John 5

fire 1
Abel 4
Mark 4
John 5
```

Grading Scheme

1. Program correctness = 70 marks, Design = 20 marks, Programming = 10 marks
2. No marks awarded if the program does not compile.
3. Marks will be deducted if **student particulars and program description** are not filled up in the top portion of the source code.
4. There are 10 test cases. Each test case is worth 7 marks. There will be 7 test cases for part A and 3 for part B.
5. Correctness marks will be halved if Stack and/or LinkedList is not used.
6. Things to look out for under design
   (a) correct usage of programming constructs (eg use correct type for variables)
   (b) Not overly complicated logic
   (c) No redundant logic
7. Things to look out for under programming style
   (a) Meaningful comments (including pre- and post-condition description if necessary)
   (b) Proper indentation
   (c) Meaningful identifiers