## OOP Exam Preparation – Capitalism

After long and hard interviews you have now resulted into the Chief Software Architect position in a growing company which provides accounting solutions. Their biggest project is a solution for companies to manage their employees, managers and salaries. You are here to design the application domain and the initial core logic for their first alpha version.

The customer companies have some kind of predefined employee positions – CEO, Managers, Regular Employees, Cleaning Ladies, Salesmen, Chief Telephone Officers and Accountants. The Cleaning Lady and Chief Telephone Officer receive 2% less than the regular salary for the others in the team. The Salesman receives 1.5% more than the regular salary for the others in the team.

The CEO is the manager of all employees, but because they have a hard time managing each employee individually, then each Manager manages the employees in its department. The departments might be very big, so the customers want to manage their sub-departments in within the department. A department manager manages the employees in the department. If it has sub-departments, they also have managers, so they receive a delegated work.

For the first version of the software, managers will only pay salaries to their subordinates, but the software might grow and extend to hire and fire employees and so on. There’s a strict scheme in paying salaries. The CEO receives fixed salary. Each of its direct subordinates receives 15% of its salary. For each sub-department, the employees in it receive 1% less. (Production -> GameProduction -> IndieGameProduction; The CEO receives 1000. The manager of Production receives 150. Its subordinates including the GameProduction manager receive 14% of it (140). The employees of GameProduction except the manager receive 1% less -> 13% (130) including the IndieGameProduction manager. The IndieGameProduction employees except the manager receive 1% less – 12% (120))

The software will be commanded from the standard input. The customer will create its company(ies), assign a CEO, then create departments. Denote if the departments are sub-departments and which department they belong to, if any. Assign managers to the departments or any other of the predefined positions.

When the customer wants to pay salaries, one just sends command for paying salaries and denotes which company one needs to pay salaries to.

## Problem 1. Class Hierarchy

Depending on the business case that is provided above, design the class hierarchy with all properties and compositions.

80 score

## Problem 2. Validation

Decorate the newly created templates with all the validations needed.

20 score

## Problem 3. Project Infrastructure

Create the event loop that dispatches the commands which might involve object creation and in-memory persistence.

40 score

## Problem 4. Command Execution

From the given input create the domain logic for the desired output using the best practices in Object-Oriented Programming and Object-Oriented Design, including highest level of abstraction, strong cohesion and loose coupling.

60 score

## Problem 5. Correct Results in the Judge System

You are given an automated Judge system where you can test your solution: <https://judge.softuni.bg/Contests/Practice/Index/144>. There are ten tests which measure correctness. Each test is worth 10 score. Submit your project as an archive file.

100 score

## Commands

The following commands the customer will send from the standard input:

* **create-company {companyName} {CEO-name} {salary}** - Creates company with the given name, assigns it a CEO. If the CEO does not exist, creates it. If the company already exists in the application then it outputs “Company <companyName> already exists”.
* **create-department {companyName} {departmentName} {managerName} *[mainDepartmentName]*** – Creates a new department in the given company. If the company does not exist, the application outputs “Company <companyName> does not exist”. If the department exists in the given company the application outputs “Department <departmentName> already exists in <companyName>” and does not add it. If the manager does not exist in the company, the application prints “There is no employee called <firstName> <lastName> in company <companyName>”. If the optional parameter ***[mainDepartmentName]*** is given, the command creates the department as a sub-department. If there is no such department in the company, the application prints “There is no department <mainDepartmentName> in <companyName>”. If the manager is not a direct employee in the specified department, the application outputs “There is no employee called <firstName> <lastName> in department <departmentName>”. If the person exists but is not a Manager, the application outputs “<firstName> <lastName> is not a manager (real position: <position>)”.
* **create-employee {firstName} {lastName} {position} {companyName} *[departmentName]*** –Creates an employee with the given first and last name in the given company department. If the company does not exist, the application outputs “Company <companyName> does not exist”. If the ***[departmentName]*** parameter is provided, assigns the employee to the specified department, and makes it a direct employee of the company otherwise. If the employee exists in the company outputs “Employee <firstName> <lastName> already exists in <companyName> (in department <departmentName>)”. If the person is not assigned to a department (and is a direct employee of the company), outputs “Employee <firstName> <lastName> already exists in <companyName> (no department)”.
* **pay-salaries {companyName} -** Pays salaries for the given company in the manner given above. Outputs in the manner given below.
* **show-employees {companyName}** - Prints the company hierarchy. The company name is printed enclosed in parentheses on the first line, then the CEO on the next line. Each subordinate (if any) is printed on the next line indented 4 spaces to the right more than their manager. Each department name is printed in order of appearance one line above its manager indented as many spaces as the manager name.

## Constraints

Consider the customer wants everything to work just fine no matter what did they input, which means commands should not have any side effects, no data types should overflow and so on.

The program is considered started once its opened and stopped when the customer stops the process by any way the operation system allows it. The program can also stop when the customer types “end”.

## Examples

### Zero Test #1

|  |  |
| --- | --- |
| **Input** | **Output** |
| **create-company C Dragan Draganin 1000**  **create-employee Koicho Stoichov Regular C**  **create-employee Nikolay Stoichov Manager C**  **create-department C Root1 Nikolay Stoichov**  **create-employee Pesho Stoichov Regular C Root1**  **create-employee Ivanka Stoichova Manager C Root1**  **create-department C Root1-1 Ivanka Stoichova Root1**  **create-employee Draganka Stoichova CleaningLady C Root1-1**  **pay-salaries C**  **show-employees C**  **pay-salaries C**  **pay-salaries C**  **show-employees C**  **end** | **C (1707.40)**  **Root1 (407.40)**  **Root1-1 (127.40)**  **(C)**  **Dragan Draganin (1000.00)**  **Koicho Stoichov (150.00)**  **Nikolay Stoichov (150.00)**  **(Root1)**  **Pesho Stoichov (140.00)**  **Ivanka Stoichova (140.00)**  **(Root1-1)**  **Draganka Stoichova (127.40)**  **C (1707.40)**  **Root1 (407.40)**  **Root1-1 (127.40)**  **C (1707.40)**  **Root1 (407.40)**  **Root1-1 (127.40)**  **(C)**  **Dragan Draganin (3000.00)**  **Koicho Stoichov (450.00)**  **Nikolay Stoichov (450.00)**  **(Root1)**  **Pesho Stoichov (420.00)**  **Ivanka Stoichova (420.00)**  **(Root1-1)**  **Draganka Stoichova (382.20)** |