## 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 feel hard times to manage 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.

## Task 1

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

## Task 2

Decorate the newly created templates with all the validations needed.

## Task 3

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

## Task 4

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.

## 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
* **create-****department {companyName} {****departmentName} *[******mainDepartmentName]*** - Creates department in a given company. If the department exists, do not do it. If the optional parameter ***[mainDepartmentName]*** is given, creates the department as its sub-department
* **create-****employee {firstName} {lastName} {****position} {companyName} {departmentName} -** Creates an employee with the given first-last name pair in the given company’s department. If the employee exists in the company, just assign it to the department
* **pay-salaries {companyName} -** Pays salaries for the given company in the manner given above
* **show-employees {companyName}** - Prints the company hierarchy. Each subordinate of a manager is printed 4 spaces indented of it. The first name, last name **and the money received so far** should be printed enclosed with brackets “( )” cut to the second digit after the decimal separator

## 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.

## Examples

### Zero Test #1

|  |  |
| --- | --- |
| **Input** | **Output** |
| **create-company C Dragan Draganin 1000**  **create-department C Root1**  **create-department C Root2**  **create-department C Sub1-1 Root1**  **create-department C Sub1-1-1 Sub1-1**  **create-department C Sub1-2 Root1**  **create-department C Sub2-1 Root2**  **create-employee Pesho Peshov** **Manager C Root1**  **create-employee Drago Chaia** **CleaningLady C Root1**  **create-employee Alex Slivarova** **ChiefTelephoneOfficer C Root1**  **create-employee Koicho Stoichov** **Regular C Root1**  **create-employee Pishman Shef Manager C Sub1-1**  **create-employee Shef Manchef Regular C Sub1-1**  **create-employee Deveta Dupka Manager C Sub1-1-1**  **create-employee Tam Ninakov Salesman C Sub1-1-1**  **pay-salaries C**  **pay-salaries C**  **show-employees C** | **Dragan Draganin (2000.00)**  **Pesho Peshov (300.00)**  **Drago Chaia (240.00)**  **Alex Slivarova (240.00)**  **Koicho Stoichov (280.00)**  **Pishman Shef (280.00)**  **Shef Manchef (260.00)**  **Deveta Dupka (260.00)**  **Tam Ninakov (270.00)** |