Lab 1

In this assignment, you will create a company management system. You will implement that system gradually in two parts. In the first part, you will implement basic classes (without inheritance) with basic functionality. In the second part, you will employ inheritance with more complex functionality.

Part 1: Introduction to **Employee** and **Manager** with **Company** Class

In the first part of the assignment, you will create Employee, Manager, and Company classes as described below. In this part, the add_employee method in the `Company` class will only add the employees to a list within the company.

1. **Employee**

- Attributes:

    - *name*: Name of the employee.

    - *employee_id*: A unique identifier for each employee.

    - *position*: The employee's job title or role.

    - *salary*: The employee's salary.

- Methods:

    - *get_details*: Display employee details.

    - *salary_raise(percent)*: Increases the employee’s salary by the specified percentage.

2. **Manager**

- Attributes:

    - *name*: Name of the manager.

    - *employee_id*: A unique identifier for the manager.

- *position*: The manager's job title or role.

- *salary*: The manager's salary.

- *subordinates*: A list of employees (subordinates) reporting to the manager.

- Methods:

- *add_subordinate(employee)*: Add a subordinate to the manager's team.

- *remove_subordinate(employee)*: Remove a subordinate from the manager's team.

- *get_subordinates*: List the subordinates reporting to the manager.

- *get_details*: Display manager details and the list of his/her subordinates.

- *salary_raise(percent)*: Increases the manager's salary by the specified percentage.

3. **Company**

- Attributes:

- *name*: Name of the company.

- *employees*: A list to store all employees in the company.

- Methods:

- *add_employee(employee)*: Add an employee to the company's list.

- *remove_employee(employee)*: remove an employee to the company's list.

- *list_employees*: List all employees in the company.

Tasks:

1. Implement the Employee, Manager, and Company classes as described above. Suggest at least two attributes to add to each class.

2. Add at least three more classes relevant to the problem where each class has at least two attributes and one method. Think about other entities in a company that can be modeled as classes (For example a Task that a manager can assign to an employee). Or, think of some attributes of the Employee class that can be modeled as a class (For example an Address). Don’t forget to update the Employee, the Manager, or the Company classes as needed to accommodate the relations to your new classes.

3. Create instances of a company, three employees, and two managers and organize them in a hierarchical structure as follows: Add the managers to the company. Add two employees under the first manager and the third employee under the second manager.

4. Show employee and manager details using the *get_details* method.

5. List all employees within the company using the *list_employees* method.

6. Demonstrate the use of your implementation (including the extra classes that you created) by creating multiple scenarios similar to the (3).

Part 2: Enhancing with Inheritance

In the second part of the assignment, you will continue working with the Employee and Manager classes, now using inheritance to model the organizational hierarchy. You will also add a general manager to the company class. You will also enhance the add_employee method in the Company class to add the employee under their respective manager. Additionally, a new method, *calculate_bonus*, will be implemented differently in the Employee and Manager classes to account for their unique roles within the company. (We will assume the company has three levels only, general manager->managers->employees)

Updated Class Hierarchy:

1. Employee and Manager (Inheritance-Based Hierarchy, as described in Part 1):

As you have (hopefully) noticed while implementing both Employee and Manager, there is a lot of annoying code duplication. The manager is also an employee in the company, and they share similar attributes. Inheritance is exactly the solution to issues like this. Think about what the two classes have in common and improve your design using inheritance to avoid as much redundancy as you can.

Also, add the method *calculate_bonus* that calculates the bonus of the employee as 5% of the salary and for the manager as 5% of the salary + 1% of the total salaries of their direct subordinates.

2. Update the Company class

- Attributes:
    - *name*: Name of the company.
    - *general_manager*: A reference to the highest-level manager (CEO) of the company.

- Methods:

- *add_employee(employee, manager_id)*: Add an employee to the company's under their respective manager. (You need to search for the manager in the company hierarchy and add the employee under the respective manager), (if the manager_id is the general manager, you need to make sure that the added employee is actually a manager).

- *remove_employee(employee)*: Remove an employee from the company (If the employee that is removed is a manager, transfer their subordinates to another manager, and if there are no more managers, remove the employees).

- list_employees: List all employees in the company (general manager, managers, and employees).

- calculate_company_salary: Calculate the total salary expenditure of the entire company.

Tasks:

1. Implement the Employee and Manager classes as described above (with inheritance).

2. Create instances of employees and managers in a company under the general manager.

3. Perform different scenarios for adding and removing employees (regular employees or managers).

4. Print a list of all the employees in the company.

5. Print the total amount of the salaries that the company pays.