CIS 345 – Business Information Systems Development - II

Assignment: Employees and Managers

Learning Outcomes

1.1. Implement derived classes using inheritance
1.2. Utilize the protected modifier and base constructors
1.3. Implement polymorphism

Program Overview

In this program, you will create a data list of Employees. Each Employee has a name & and ID.

Managers must also be part of this data list. A manager is an employee as well. However, in addition to managers having all the characteristics of employees, managers also manage employees as subordinates. Therefore, a Manager has a list of Employees they manage as well as a count of how many subordinates they have.

Your application should ask the user for details to maintain the list of employees. For each employee, the program should ask the user for the name and whether or not the employee is a manager.

If the user specifies that the employee is a Manager, the program should ask the user how many subordinates they manage as well as their names. It should create a Manager object, and within the Manager object, create the subordinate Employees. The program should ask the user for the name of the subordinate employees and store their names in the subordinate Employee objects.

If the user specifies the employee is not a Manager, the program should create an Employee object.

All employees (including Managers), should be added to the data list of Employees. Finally, the program should print out a list of all employees. However, if an Employee is a Manager, it should also print out a list their subordinate Employees.

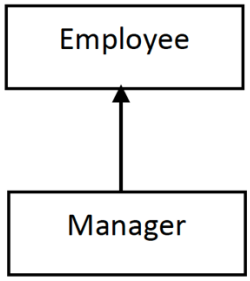
Sample Output

```
            Employee Management System

Adding Employees...

Enter name: Jeff
Enter id: 1
Is the employee a manager? (Y/N) n
Do you want to enter more? y

Enter name: Olivia
Enter id: 2
Is the employee a manager? (Y/N) y
How many subordinates? 2
Enter subordinate name: Jessica
Enter subordinate id: 45
Enter subordinate name:
Enter subordinate id:
Do you want to enter more? y

Enter name: Peter
Enter id: 3
Is the employee a manager? (Y/N) n
Do you want to enter more? n

Printing Employee List
0001: Jeff
0002: Olivia
    Olivia's Employees
    0045: Jessica
    9999: Unknown
0003: Peter

Process finished with exit code 0
```

Instructions

Create a new python module named **[ASUrite]_main.py**. Code the following within.
Classes: Employee, Manager (Name all your classes and filenames appropriately).

- Implement an inheritance relationship between classes Employee and Manger based on the following inheritance structure:



Implement each of the following classes using the UML class diagram for details on the class structure and following instructions for details on the logic to be implemented.

Don't forget to always put your name, class, etc. on Line 1 of ALL modules.

EMPLOYEE CLASS

Employee
- name: str - eid: str + <<property>> name : str + <<property>> eid : str
+ init (name : str, eid : str) + str ():str

Purpose: Employee class abstracts the notion of an Employee. Use name mangling on variables.

Properties: Implement properties as seen in the UML Class Diagram.
Both properties must implement both the property and setter decorators.

name: property returns capitalized name and setter checks the name isalpha() and if true store the name in the object otherwise store 'Unknown'

eid: property returns eid zero filled up to 4 spaces and setter will assign '9999' if length is zero, otherwise store the eid.

Methods: str () formats '<eid>: <name>' so all employee always print both id and name.

MANAGER CLASS

Manager
+ subordinates: list
+ init (name : str, eid : str) + print_subordinates(): None + add_subordinate(): None

Purpose: A Manager class abstracts the notion of a Manager. A **Manager** is also an Employee. Therefore, the Manager class should inherit from the Employee class.

Fields: Maintain fields for variables which you need to access from other methods! You definitely need an list of Employees and a count. Add any more that you need!

__init__(): Implement the initializer for Manager. Be sure to call the **Employee** or super initializer!

Methods

add_subordinate() method

General Method Purpose: This method should add one employee to the subordinates list. So you should ask the user details about one employee, create a new Employee and add it to the list.

print_subordinates() method

General Method Purpose: This method should loop through the subordinates list and print out a list of all employees' names and ids.

main() Function

Purpose: Implement a main function (or similar), which starts up your program. Call main() at the bottom of your module.

Local variables: Maintain fields for variables which you need to access from other functions/methods! You definitely need a list of Employees. Add more variables that you might need!

General Function Purpose: Maintain this function as your main program logic which runs the program. This means that this function should make sure you have an Employee list, show the welcome **screen**, prompt the user to enter employees as long as the user keeps saying yes, and finally displays a list of all the employees.

When user is done adding employees, print a header stating you are 'Printing Employee List'. Then proceed to print all employees.

If an Employee is a Manager, it should also print out a list of all its subordinates by calling the relevant method from the Manager class. *Remember – you will need to determine if the instance is a Manager first! This is similar to how we went through the list of Students in class!*

add_employee() Function

General Function Purpose: This function should return one employee to **main** to be added to the list. Therefore, you should ask the user details about one employee as well as whether the employee is a manager or not. Depending on the answer, you should create either an Employee object or a Manager object.

If the user specifies that the employee is a manager, it should ask the user how many subordinates are to be added. You should call the relevant method from the Manager class, however many times as needed.

Make sure you return the employee (be it Manager or Employee) to **main** to be added to the list of employees you are maintaining in the **main** function!

General Grading Criteria

1. Assignments will be scored out of 30 points.
2. Assignments will be on source code AND output.
3. Submit your **[ASUrite]_main.py** only.

Grading Criteria	Points
Class Employee is implemented properly including properties	1
Class Manager is implemented properly. Class Manager manages the list of subordinates, e.g. adding them, printing subordinate list, etc.	1
main Function manages a list of Employees. There should be no separate list of Managers. Employees who are managers are managed using isinstance	0.7
Function is implemented for Adding Employees.	0.5
Program input/out is done as expected according to sample program screenshot.	0.5
• Variable names are descriptive • variable names conform to PEP8 standards • Functions/Methods use snake_case per PEP8 • Program has relevant comments documenting the code • File names and project name are accurate. • Class file has name, class, assignment number, and class time written on Line 1 **of all modules**	0.3