ASU W. P. Carey School of Business
Arizona State University

CIS 345 – Business Information Systems Development - II

Assignment 1: Inheritance and Polymorphism

Due on Blackboard: **Friday, September 01, 2017, by 11:59 PM** Arizona Time

Learning Outcomes

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

Program Overview

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

Managers must also be part of this database. 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 he/she manages as well as a count of how many subordinates he/she has.

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.

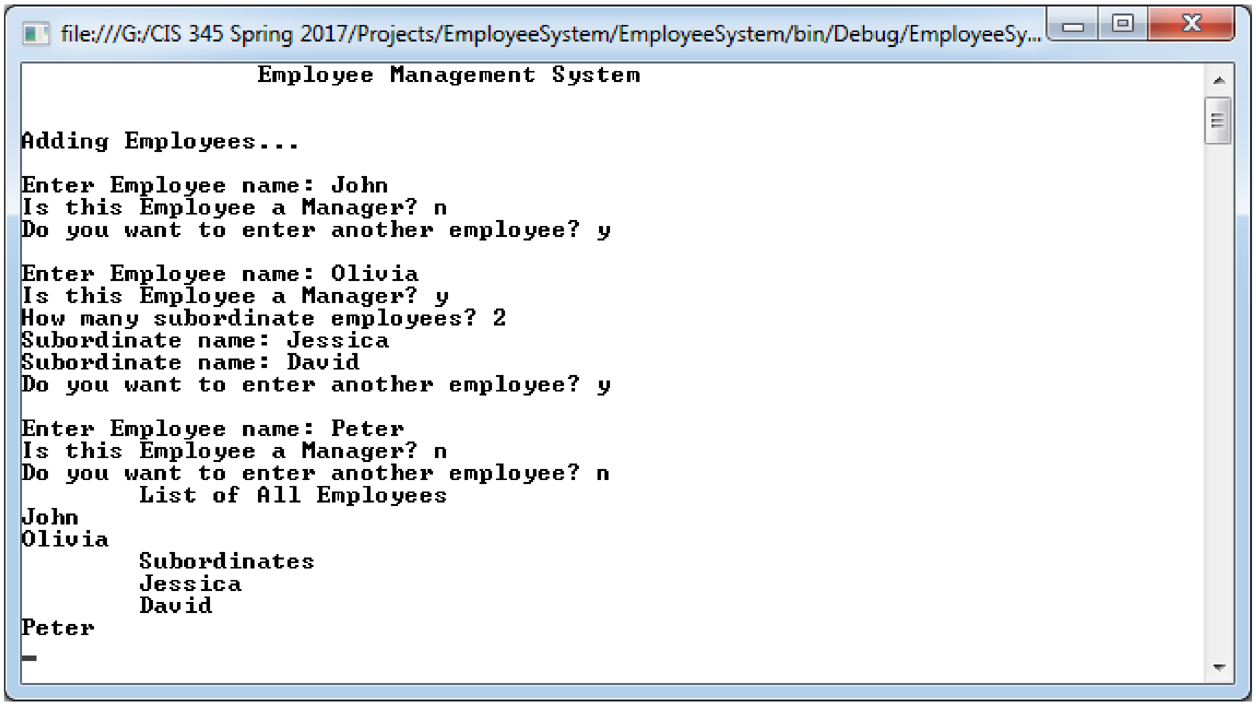
If the user specifies that the employee is a Manager, the program should ask the user how many subordinates he/she manages as well as their names. It should create a Manager object, and within the Manager object, create the subordinate Employees. The program should ask the user for the name of the subordinate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

If the user specifies a Manager, the program should create an Employee object.

The new employee, either Manager or regular Employee, should be added to the database 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 his/her subordinates Employees.

Sample Output

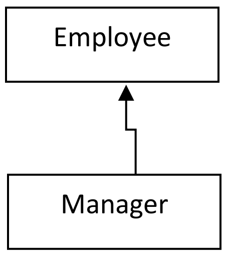

file:///G:/CIS 345 Spring 2017/Projects/EmployeeSystem/EmployeeSystem/bin/Debug/EmployeeSy...

```
                Employee Management System


Adding Employees...

Enter Employee name: John
Is this Employee a Manager? n
Do you want to enter another employee? y

Enter Employee name: Olivia
Is this Employee a Manager? y
How many subordinate employees? 2
Subordinate name: Jessica
Subordinate name: David
Do you want to enter another employee? y

Enter Employee name: Peter
Is this Employee a Manager? n
Do you want to enter another employee? n
        List of All Employees
John
Olivia
        Subordinates
        Jessica
        David
Peter
_
```

Instructions

Project Name: A1
Classes: EmployeeSystem, Employee, Manager (Name all your classes and filenames appropriately).

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



- Use auto generation of code where you can!

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

Put your name, class, etc. on Line 1 of ALL classes.

If you want to deviate and implement a slightly different solution, you can do that!

EMPLOYEE CLASS

Employee
name: string # id: string + <<property>> Name : string + <<property>> ID : string
+ <<constructor>> Employee() + <<constructor>> Employee(name: string)

Purpose: The Employee class abstracts the notion of an Employee.

Properties: Implement properties as seen in the UML Class Diagram.

Constructors: Implement constructors as seen in the UML Class Diagram.

MANAGER CLASS

Manager
- subordinates: Employee[] - numberOfSubordinates: int
+ <<constructor>> Manager() + <<constructor>> Manager(name: string) + PrintSubordinateList(): void + AddSubordinate(): Employee

Purpose: A Manager class abstracts the notion of a Manager. An apple 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 array of Employees and a count. Add any more that you need!

Constructors: Implement constructors for manager. For the overloaded constructor, be sure to redirect to the overloaded constructor of ~~Student!~~

Methods

AddSubordinate() method

General Method Purpose: This method should add one employee to the subordinates array. So you should ask the user details about one employee, create a new Employee and add it to the array. Remember to increment the count of subordinates!

PrintSubordinateList() method

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

EMPLOYEESYSTEM CLASS

EmployeeSystem
- listofEmployees : Employee[] - countOfEmployees: int
- RunProgram() : void - AddEmployee() : void - PrintEmployeeList() : void + Main (args: string[]) : void

Purpose: This is the main program class, where the program starts. Implement a RunProgram method (or similar), which starts up your program. Call this method from your Main method.

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

Methods:

RunProgram() method

General Method Purpose: Maintain this method as your main method which runs the program. This means that this method should make sure you have an Employee array, start up the welcome street, prompt the user to enter employees as long as the user keeps saying yes, and finally displays a list of all the employees.

AddEmployee() method

General Method Purpose: This method should add one employee to the array. 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 and you should call the relevant method from the Manager class, however many times as needed.

Make sure you add the employee (be it Manager or Employee) to the listOfEmployees and increment the count! *This uses upcasting!*

DisplayEmployeeList() method

General Method Purpose: This method should loop through the listofEmployees and print out the name of 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 use downcasting for this!*

General Grading Criteria

1. Assignments will be scored out of 30 points.
2. Assignments will be on source code AND output.

Grading Criteria	Points
Class Employee is implemented properly	7
Class Manager is implemented properly. Class Manager manages the list of subordinates, e.g. adding them, printing subordinate list, etc.	10
Class EmployeeSystem manages an array of Employees. There should be no array of Managers. Employees who are managers are managed using upcasting and downcasting (polymorphism) Methods are implemented for Adding Employees and Printing Employee list. Program input/out is done as expected according to sample program screenshopt.	10
• Variable names are descriptive • Camel Casing is used for variable names • Pascal Casing is used for public Methods, Properties • 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 classes**	3