I’m working on the Open-ended Project Progress.  
The project I’m developing has the purpose to solve one of the problems we have in the company I work for.  
Problem: managing company’s employee information, including their name, job title, salary, and vacation days; calculate employee bonuses based on their performance.  
  
I want to create a program that will create a list of employees, add new employees, remove employees, and calculate bonuses. The program will have the following 8 classes:

1. Program.cs: This is the main entry point for the program. It initializes the program and handles any necessary setup before starting the program loop. It also contains the Main method, which is the starting point of the program.
2. Employee.cs: This is the base class for all employee information. It defines the basic properties and methods that all employees have, such as name, job title, salary, and vacation days.
3. Manager.cs: This is a subclass of Employee that adds manager-specific information and behavior, such as managing employee performance and delegating tasks to employees.
4. HRAdmin.cs: This is another subclass of Employee that adds HR administrator-specific information and behavior, such as managing employee benefits and handling employee complaints.
5. EmployeeList.cs: This class manages a list of employees. It allows the program to add new employees, remove employees, and access information about individual employees or the entire list of employees.
6. BonusCalculator.cs: This class calculates employee bonuses based on their performance. It takes into account factors such as productivity, attendance, and overall job performance.
7. InputHandler.cs: This class handles user input, such as reading input from the console and validating user input.
8. OutputHandler.cs: This class displays program output to the console, such as displaying employee information or program messages.

The principle of abstraction will be demonstrated by having separate classes for Employee, Manager, and HRAdmin; each with their own responsibilities and methods for managing employee data.  
The principle of encapsulation will be demonstrated by using private fields for employee information and public methods for accessing and modifying that information.  
The principle of inheritance will be demonstrated by having Manager and HRAdmin classes inherit from the Employee class and sharing common methods and properties.  
The principle of polymorphism will be demonstrated by having the Manager class override the CalculateBonus method from the Employee class to provide a custom implementation.  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
Progress:  
So far, I have finished the project.  
1) Employee  
The class has properties for the employee's name, job title, salary, and vacation days. The constructor initializes these properties when a new employee is created. The class also has three methods: GetInfo() returns a string with the employee's information, CalculateVacationDays() calculates the number of vacation days an employee has based on their salary, and CalculateBonus() calculates the bonus amount an employee is entitled to based on their salary. All three methods are marked as virtual so that they can be overridden in derived classes.

2) Manager  
The class inherits from the Employee base class, so it includes all of the properties and methods defined in the base class. In addition, it has two properties: Performance, which represents the manager's performance rating as a percentage, and Tasks, which is a list of tasks assigned to the manager. The constructor initializes these properties in addition to the base properties. The class overrides the GetInfo() and CalculateBonus() methods from the base class to include the manager's performance rating and a performance bonus based on their rating, respectively. The class also includes two new methods: AssignTasks() to assign tasks to employees and EvaluatePerformance() to evaluate the performance of an employee.

3) HRAdmin  
The class inherits from the Employee class and has an additional property called Benefits, which is a list of strings. The class has two methods: AddBenefits and HandleComplaint. The AddBenefits method takes an Employee object and a list of strings representing benefits. The method updates the employee's job title and adds the benefits to the HRAdmin's Benefits list. The HandleComplaint method takes an Employee object and a string representing the complaint. The method simply writes a message to the console indicating that the HR Admin is handling the complaint.

4) EmployeeList  
The class has a private property Employees of type List<Employee> to store all the employees. The class provides four methods: AddEmployee to add an employee to the list, RemoveEmployee to remove an employee from the list, GetEmployee to get an employee by name, and GetAllEmployees to get all the employees in the list.

5) BonusCalculator  
The class has two properties - Productivity and Attendance - which are used to calculate the bonus for non-manager employees. The CalculateBonus() method takes an Employee object as input, and checks if it is a Manager object or not. If it is a Manager object, the bonus is calculated based on the manager's performance. If it is a non-manager employee, the bonus is calculated based on the employee's productivity and attendance, using the CalculateBonus() method from the Employee base class.

6) InputHandler  
The GetEmployeeData() method prompts the user for employee information and validates the input before returning a new Employee object with the entered data. The GetCommand() method prompts the user for a command and returns the lowercase input as a string for the program to interpret. The available commands are "add", "remove", "get", "getall", and "exit".

7) OutputHandler  
The DisplayEmployeeInfo method takes an Employee object as a parameter and displays the employee's information on the console using the GetInfo method defined in the Employee class. The DisplayMessage method takes a string message as a parameter and simply displays it on the console.

8) Program  
This Program.cs file creates an employee list and enters a loop to handle the user's input. The user is presented with a menu that allows them to add an employee, remove an employee, view an employee's information, calculate employee bonuses, or exit the program.  
When the user selects an option, the appropriate code is executed. For example, when the user selects the "Add Employee" option, the program uses the InputHandler class to get the employee data from the user, creates a new Employee object, and adds it to the employee list.  
Similarly, when the user selects the "Remove Employee" option, the program prompts the user for the employee name, gets the Employee object from the employee list, and removes it from the list.  
The program also uses the BonusCalculator class to calculate employee bonuses, based on the employee's role and performance.