Project Overview:

For our project we made a simple Employee Management system that allows for a variety of features to help organize and keep track of employees in the system. We have implemented 3 design Patterns in the software to help make it more expandable down the road, for pattern we used composite, observer, and decorator patterns.

The application can perform a few different tasks to help keep track of employees. Firstly, you can add new employees to existing mangers. You can also add an existing employee to a carpool list as well and then you are able to retrieve the list of employees of a manager or a carpool. We also added a way to update an employee's information such as their address or the pay. With the employee's pay we can figure out use that to perform some operations to easily find their hourly, weekly, monthly and bonus pay and then finally you can remove an employee from Managers subordinate list if they were fired, which also removes them from any carpools they are also in.

Composite Pattern:

Created two classes(Carpool Class & managers class)   implementing the Employee class.

so we have a kind of tree shape , root is Employee class , and we   have 2 branches(leafs) :

1. The carpool class is implementing   the employees class so   we can update employees address and choose the carpool to drive the employee.

2. The Managers class is implementing the Employee class so we can add employees, remove employees.

Expandability :

1. As The company grows we are free to add more branches to our tree(composit Pattern), for example we can add the supervisors class,   sales people class, outsourcing class....etc.

2. we can also expand the carpool instead of having only 3 drivers , we can add more drivers because when we have more employees we will need to expand the carpool and drivers numbers.

in short words, we can expand both(carpools & managers) without effecting the client code(open/closed principle).

for more expandability , we can make more leafs for each branch of the tree, for example managers class can be expand to TWO   leafs or more:

a. Human resources managers

b. Sales managers

where both a & b implements the managers class.

same concept of expandability can be applied also for Carpool class.

As a conclusion, using a composite pattern gives us the **ability to expand our tree in 2 dimensions** (we can add more Branches) and we can also make the tree goes deep and add more classes (leaf's) to the lowest level of the tree.

Observer Pattern:

The observer pattern was used in this program, since the Employee class could be updated by the user from the UI, and we wanted feedback to be given on whether that update was done. Since observers are used to inform other classes of changes in classes, we decided it would be best in this case.

We have two types of observers, one for each of the editable fields: salary and address. Both implement the Observer interface. They send an informative message including the employee's name, grabbed by the observer, when the update takes place.

Expandability:

This can be expanded to allow for different observers for new fields that can be updated, and different buttons. It can be expanded to iterate through all observers upon update to check for different changes. This was not included however since we changed the observer to fit the smaller scope of the project but can be expanded with more UI elements and class attributes.

Decorator Pattern:

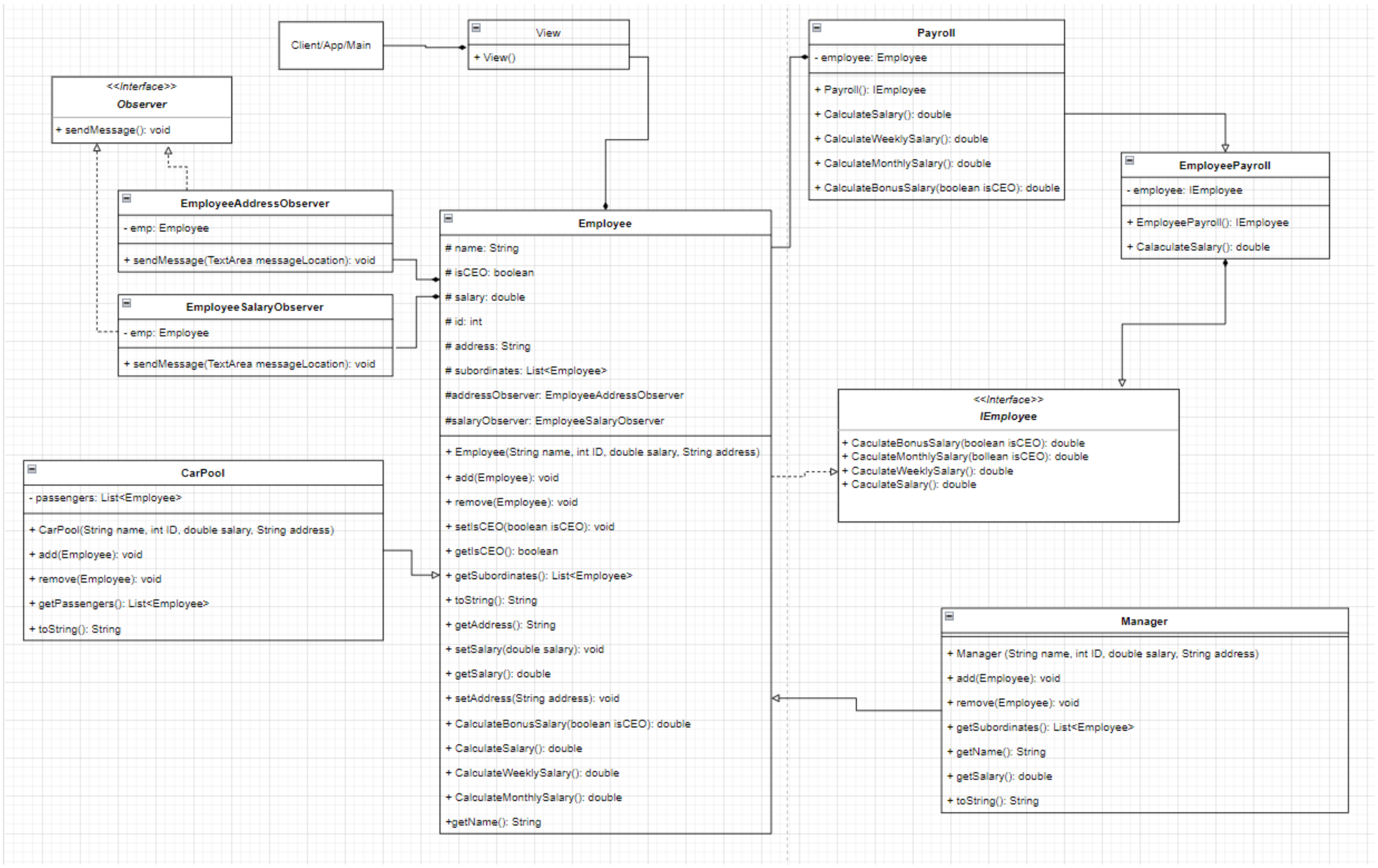
The decorator pattern was chosen to avoid breaking the single responsibility principle. Since the employee class already handles the implementation as far as an employee is concerned, adding another functionality such as the payroll will break this principle since the employee class will be responsible for both the employees and the payroll.

The payroll class relies on the shared IEmployee interface and calculates all payroll-related activities for an employee. This includes the weekly salary, monthly salary and bonus. It also identifies if the user is a CEO or not when calculating the bonus and uses that to determine how much bonus the employee is entitled to.

Expandability:

This can be further expanded to include more payroll activities such as increase salary, payout salary, calculate pension or retirement fund etc.

UML Diagram



Client/App/Main
View
+ View()
Payroll
- employee: Employee
+ Payroll(): IEmployee
+ CalculateSalary(): double
+ CalculateWeeklySalary(): double
+ CalculateMonthlySalary(): double
+ CalculateBonusSalary(boolean isCEO): double
<<Interface>>
Observer
+ sendMessage(): void
EmployeePayroll
- employee: IEmployee
+ EmployeePayroll(): IEmployee
+ CalaculateSalary(): double
EmployeeAddressObserver
- emp: Employee
+ sendMessage(TextArea messageLocation): void
Employee
# name: String
# isCEO: boolean
# salary: double
# id: int
# address: String
# subordinates: List<Employee>
#addressObserver: EmployeeAddressObserver
#salaryObserver: EmployeeSalaryObserver
+ Employee(String name, int ID, double salary, String address)
+ add(Employee): void
+ remove(Employee): void
+ setIsCEO(boolean isCEO): void
+ getIsCEO(): boolean
+ getSubordinates(): List<Employee>
+ toString(): String
+ getAddress(): String
+ setSalary(double salary): void
+ getSalary(): double
+ setAddress(String address): void
+ CalculateBonusSalary(boolean isCEO): double
+ CalculateSalary(): double
+ CalculateWeeklySalary(): double
+ CalculateMonthlySalary(): double
+getName(): String
EmployeeSalaryObserver
- emp: Employee
+ sendMessage(TextArea messageLocation): void
<<Interface>>
IEmployee
+ CaculateBonusSalary(boolean isCEO): double
+ CaculateMonthlySalary(bollean isCEO): double
+ CaculateWeeklySalary(): double
+ CaculateSalary(): double
CarPool
- passengers: List<Employee>
+ CarPool(String name, int ID, double salary, String address)
+ add(Employee): void
+ remove(Employee): void
+ getPassengers(): List<Employee>
+ toString(): String
Manager
+ Manager (String name, int ID, double salary, String address)
+ add(Employee): void
+ remove(Employee): void
+ getSubordinates(): List<Employee>
+ getName(): String
+ getSalary(): double
+ toString(): String