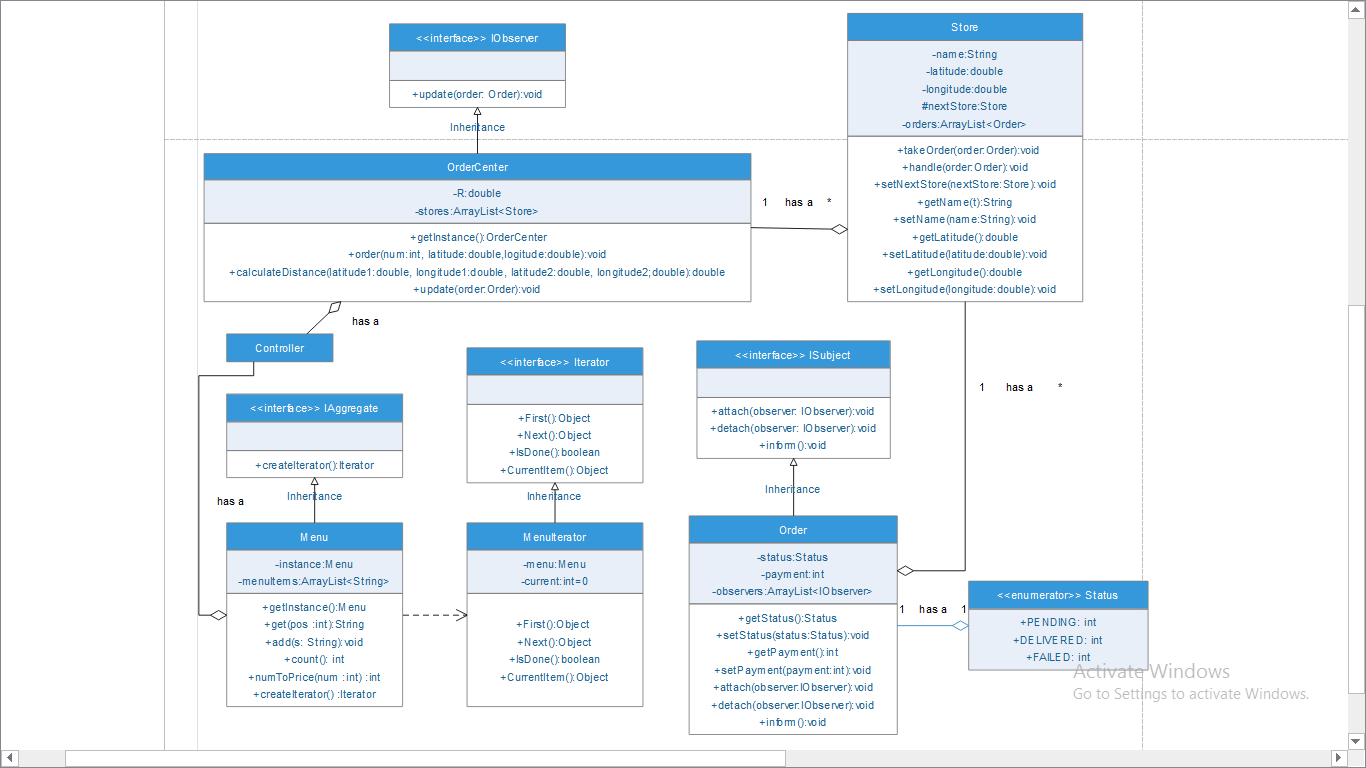
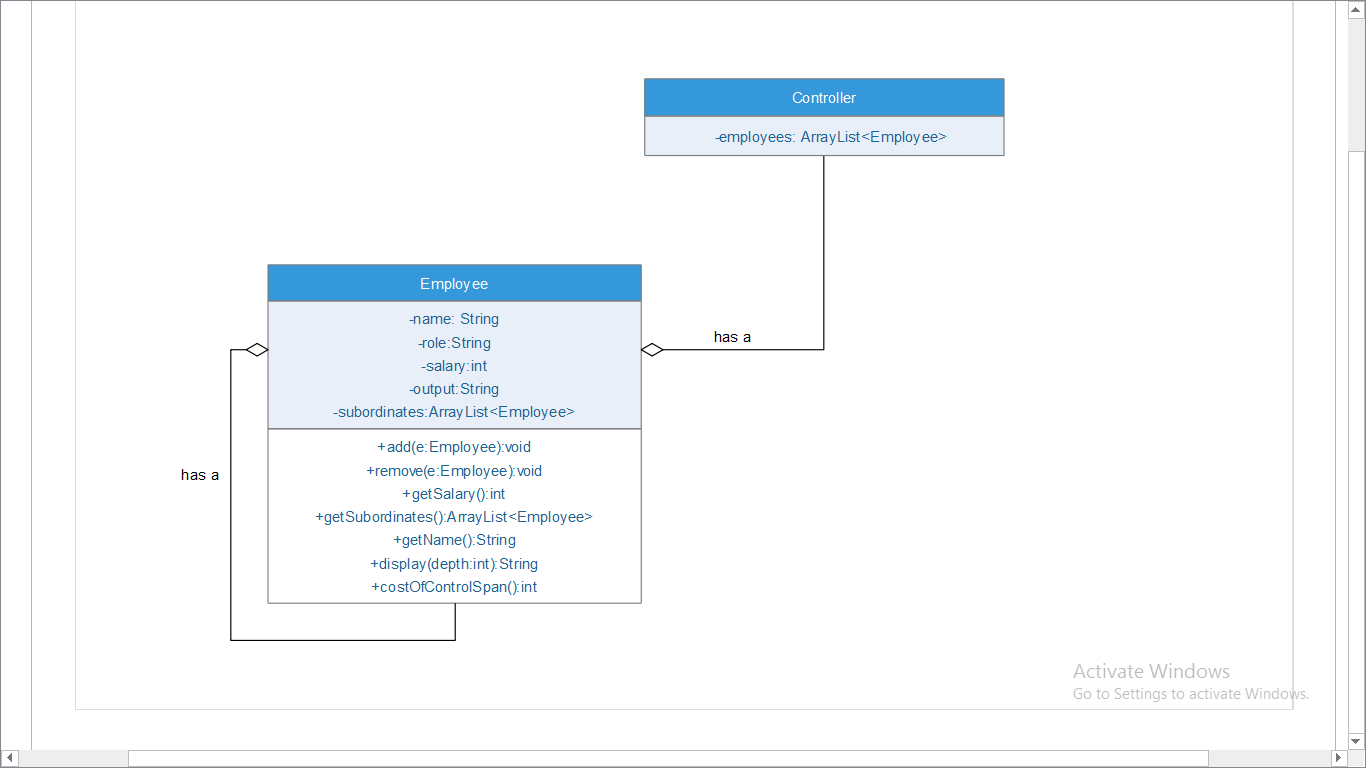
**Class Diagram of question 4**



**Class Diagram of Question 3**



**Question 1**

The pattern we used is Template Pattern. The reason is Template pattern has an abstract class exposes defined way*s*/template*s* to execute its methods. Its subclasses can override the method implementation as per need but the invocation is to be in the same way as defined by an abstract class.

The invocations we use in each subclass are in the same way. We created BestSwimmers abstract class with sort method to be final so that it cannot be overridden. SortByAge and SortBySex are the concrete classes that extend BestSwimmers class and override the method. To sort the swimmers’ list based on age group, we first filter it and then sort it. The same goes for sorting the swimmers’ list based on Sex. So, we can use the same invocation of the methods in each case, the reason why we used template pattern

**Question 2**

The pattern we used is decorator pattern. The reason is Decorator pattern allows a user to add new functionality to an existing object without altering its structures. The pattern creates a decorator class which wraps the original class and provides additional functionality keeping class methods signature intact.

In our case, there is a customer view that list out product items. And there is Executive View that does the same thing as customer view but adds a new functionality that is listing out number of units shipped for each product. So we can decorate the customer view by adding number of units shipped for each product. There is also additional functionality on customer view that sorts the product items if requested by the user. So, we can decorate the customer view when sorting is requested by the users.

**Question 3**

Question 3 asks us to create a tree structure of the organization. By definition, Composite pattern composes objects in term of a tree structure to represent part as well as whole hierarchy. So we used composite pattern.

**Question 4**

The real world problem, we tried to solve is a single Order System for many branches of the same restaurant. We used the following patterns:

1. Singleton because we want only one order center so we create one instance of Order Center class
2. Iterator, to display all the elements of the Menu class
3. Observer, The Order center is the one that communicates with the customer, so the Order Center observes each order. When the order changes its state the Order Center is notified
4. Chain of Responsibility, the Order Center has to give an order to a store. So to choose from the different branches, the Order Center uses the location of the branches. It arranges them in a chain from the closest to farthest. If the first store can take the order, it takes it otherwise it passes it to the next store