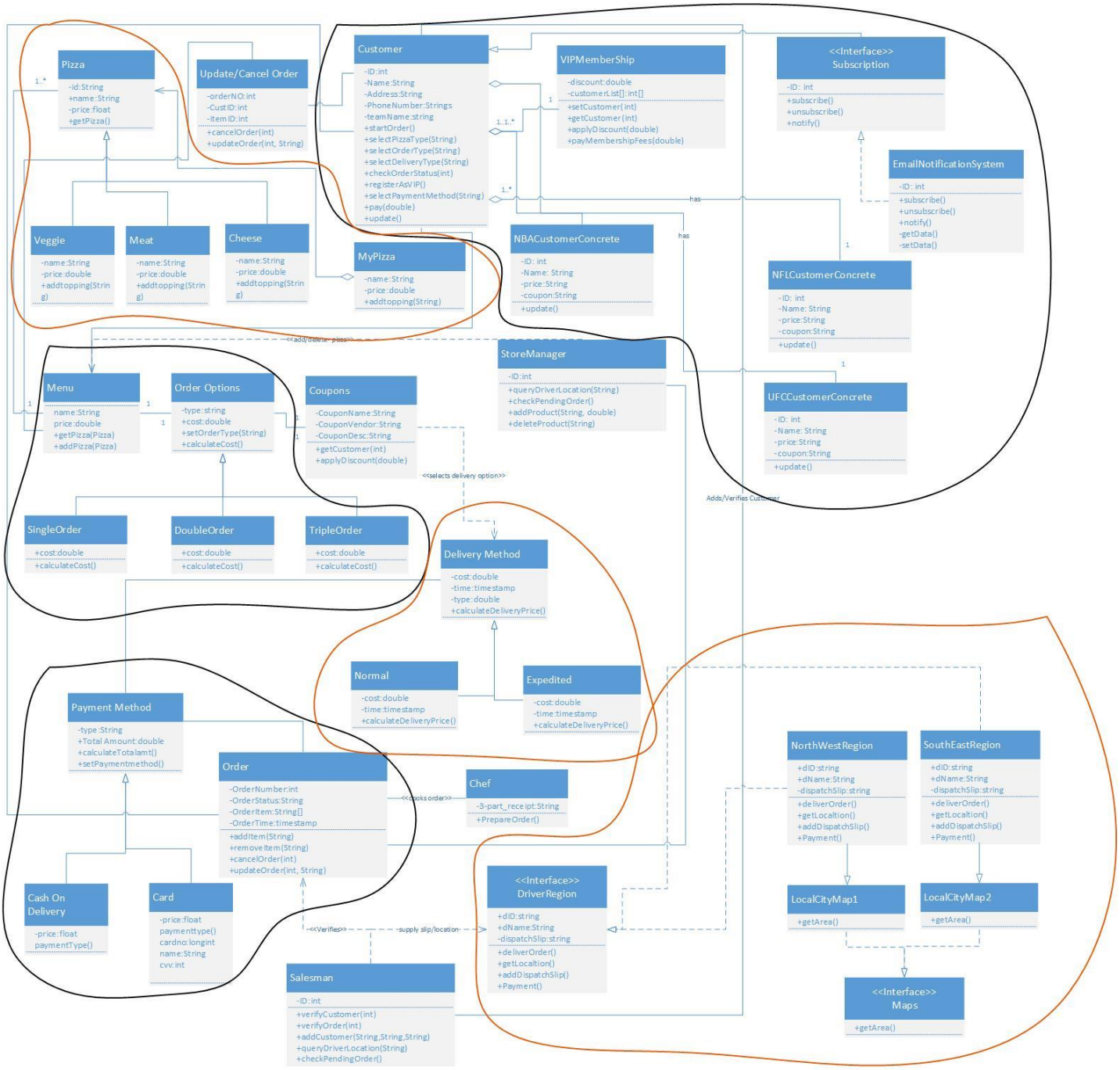
UML - NIKHIL ARORA – A20339503

ASSIGNMENT NO: 4

COMPLETE UML DESIGN MODEL WITHOUT EXPANDING ANY CLASSES

- **Customer**
- **Salesman**
- **Store Manager**
- **Menu**
- **Pizza**
  - **Meat**
  - **Cheese**
  - **Veggie**
- **Order Options**
  - **Single Order**
  - **Double Order**
  - **Triple Order**
- **Coupons**
- **Delivery Method**
  - **Normal**
  - **Expedited**
- **Payment Method**
- **Update / Cancel Order**
- **Order Confirmation**
- **Order Status**
- **Pizza Inventory**
- **Chef**
- **NorthWestDriverFactory**
- **SouthEastDriverFactory**
- **Print**
- **Verification**
- **VIP Membership**
- **Subscription**
- **EmailNotificationSystem**
- **NFLCustomerConcrete**
- **NBACustomerConcrete**
- **UFCCustomerConcrete**
- **MyPizza**

Complete UML Design Model / Class Diagram



BLACK color highlights denotes the design patterns implemented in previous assignment

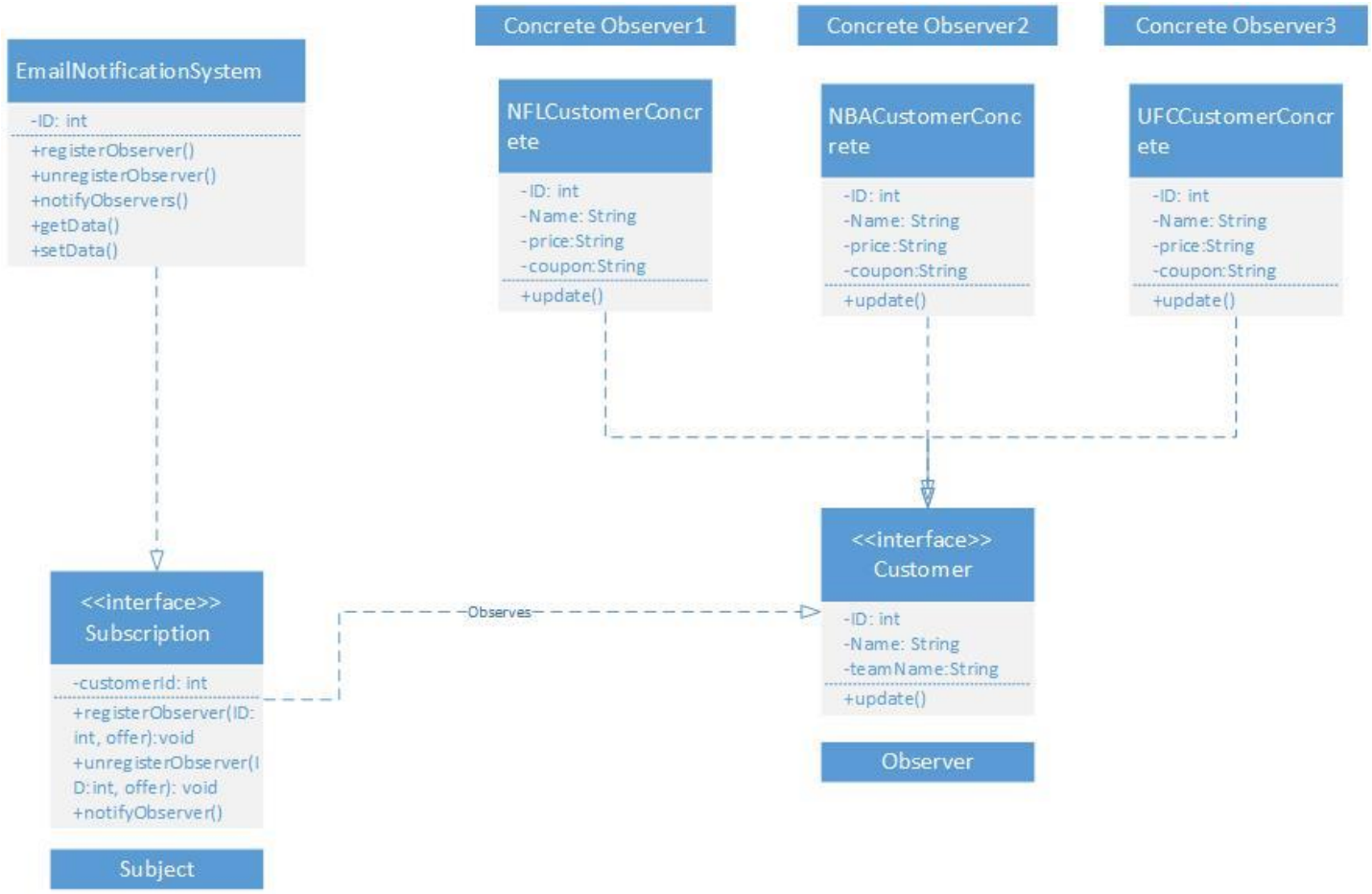
RED color highlights denotes the design patterns implemented in current assignment

List of design patterns used in previous assignment.

- Observer Pattern: Used on E-mail Subscription module
- Strategy Pattern: Used on Order Option module.
- Factory Pattern: Used on Payment module.

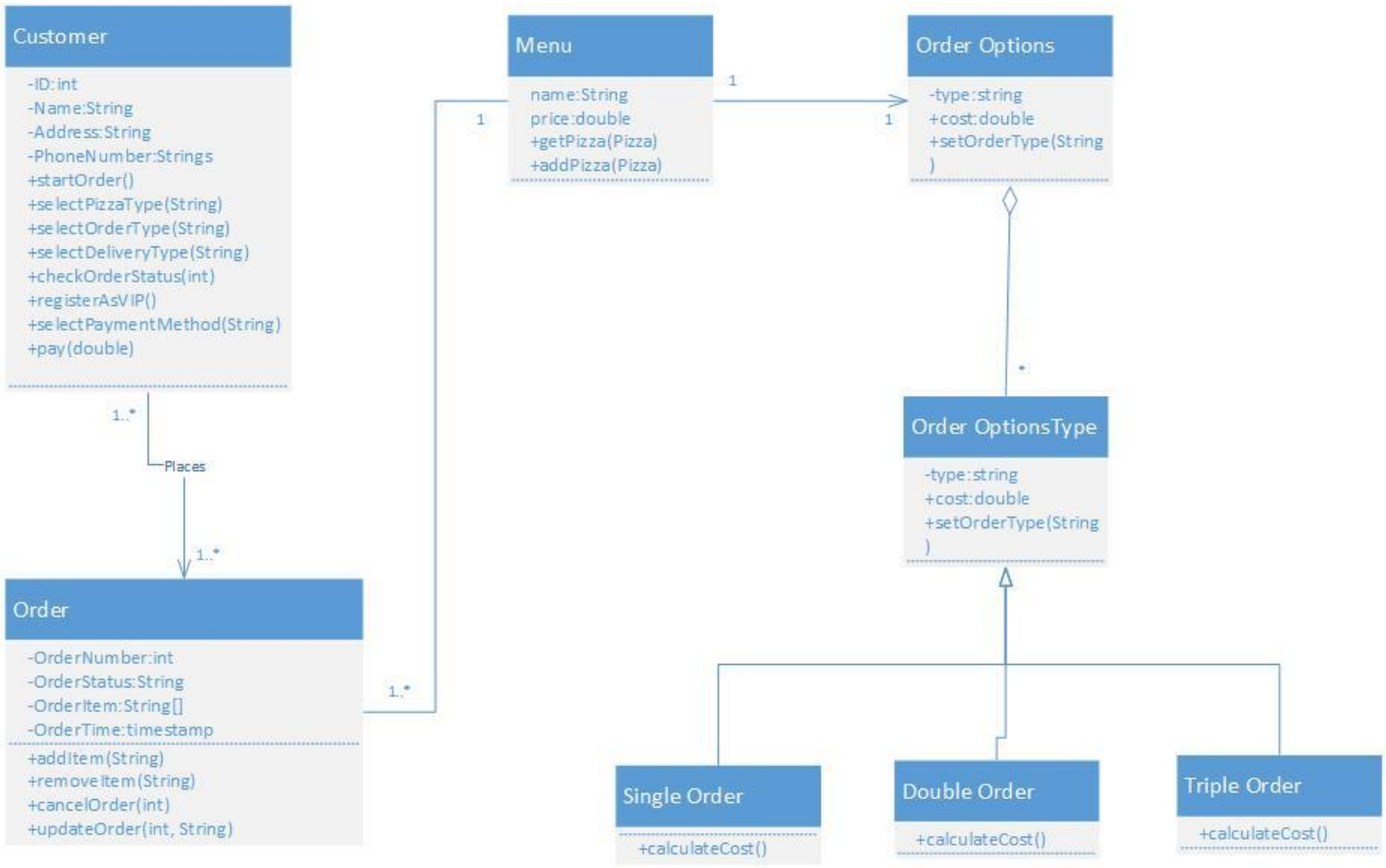
Design Pattern Usage defined below.

1. Observer Pattern:



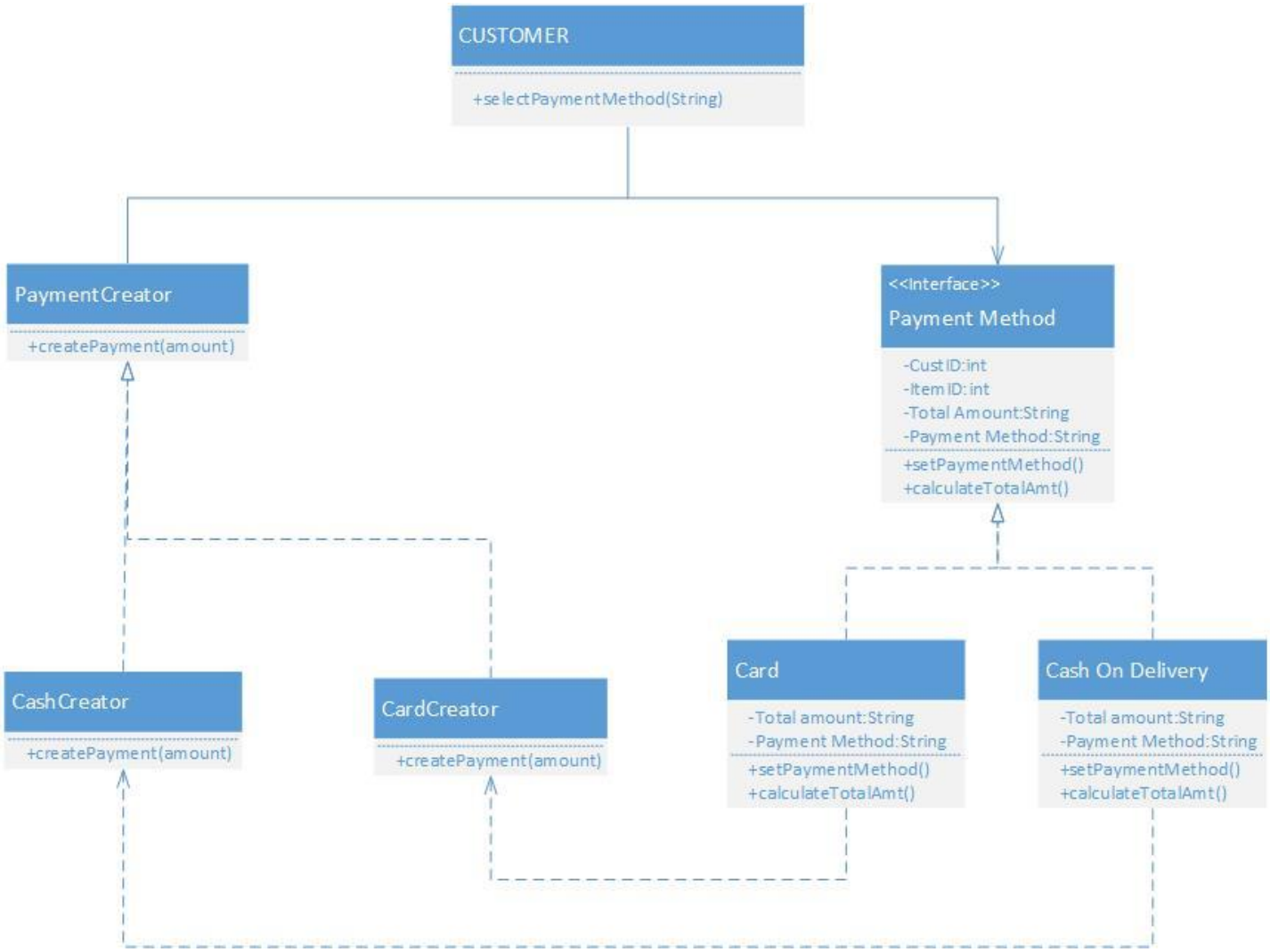
In subscription module observer pattern is used. Here above certain amount, user can subscribe / unsubscribe for email notifications for free pizza delivery. In our case Customer is the observer who use the interface subscription to use that offer. To subscribe / unsubscribe for email notification is being provided be the subscription interface.

2. Strategy Pattern:



Here Customer can choose any of the three kinds of pizza options which help customer to multiple the amount of order based on their choice. I have used strategy pattern to show the proper strategy for choosing the order options which includes single /double/triple the order as per customer need.

3. Factory Pattern:

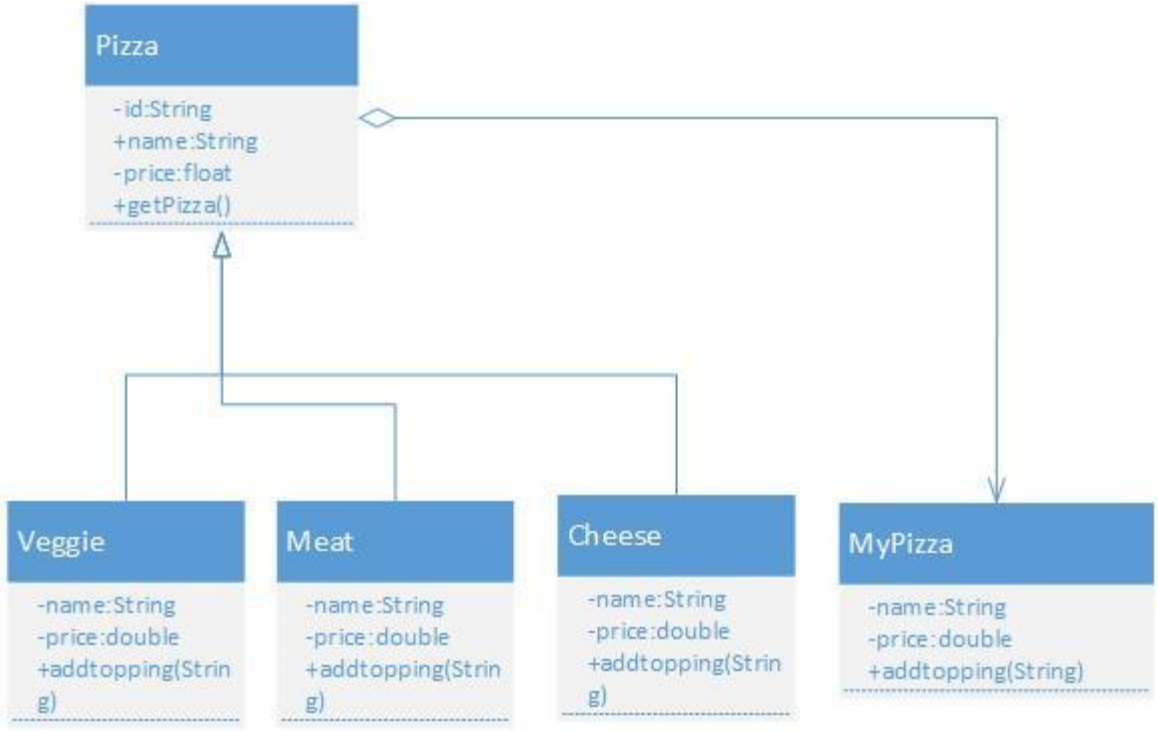


On Payment module I have implemented factory pattern. Here we have different methods to pay for the order for example payment by card or cash on delivery etc. each payment type has its own creator class connected to them. All the methods are provided by the Payment method which is super class acting as an interface for them. Customer uses the desired method to pay for the order.

NOTE: Here I have expanded all relevant classes related to each design patterns.

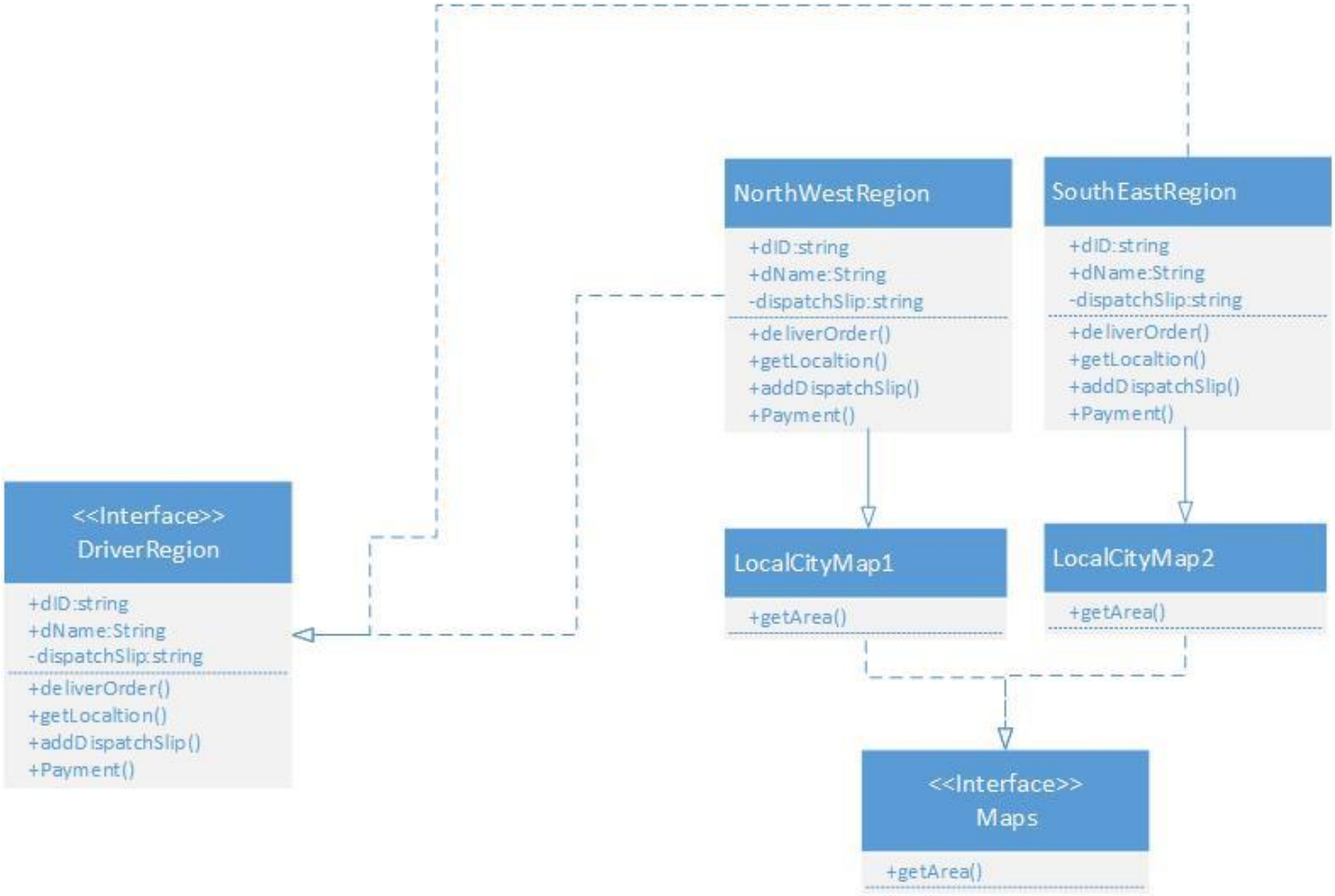
List of design patterns used in current assignment.

1. Composite Design Pattern



This type of pattern comes under structural pattern. This pattern composes objects into tree structures to represent whole-part hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly. This pattern describes Menus that contain menu items, each of which could be a menu. Here Pizza can be any of these, it can be veggie pizza or meat pizza or cheese pizza, and all three subclasses can also act as a menu as we have different types of veggie, meats and cheese.

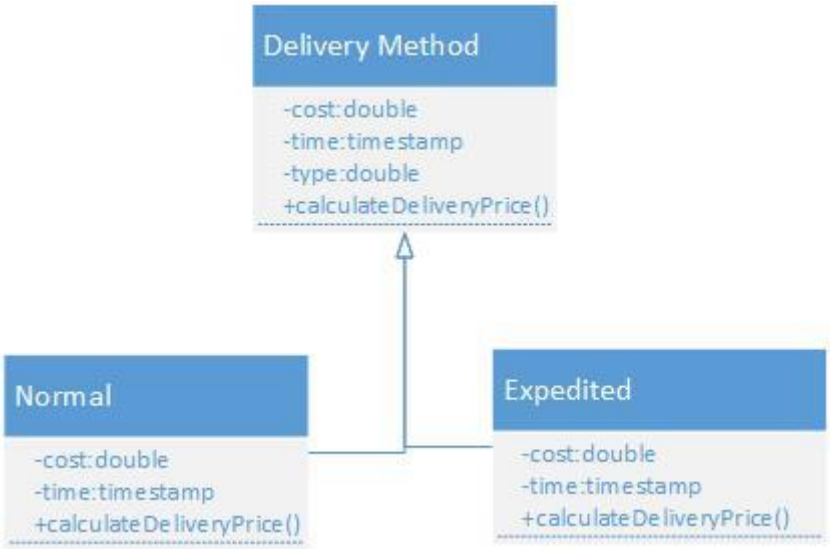
2. Abstract Factory Design Pattern



Abstract Factory patterns work around a super-factory which creates other factories. This factory is also called as factory of factories. This pattern is also considered as creational pattern. In my example, I partitioned Driver class to two levels to partition drivers into two different locations which are NorthWestRegion and SouthEastRegion. The interface DriverRegion contains methods which are implemented in classes SouthEastRegion and NorthWestRegion.

There is another interface which calls function getArea() implemented in two classes as LocalCityMaps1 and LocalCityMaps2, which will provide driver a map for particular area. By implementing abstract pattern leads us with added advantage so that whenever the business grows or we want to cover another area we just need to add region which will not interrupt our current business.

3. Template Design Pattern



This pattern defines the skeleton of an algorithm in an operation, deferring some steps to client subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure. In our case, implementation of calculateDeliveryPrice() is the method declared in base class which is getting defined in the child classes. All the code which is reusable is defined in the base class and child classes either use the code or can override or customize the existing code or can define their own code. Each of the existing classes declares an "is-a" relationship to the new abstract base class. Factory Method is a specialization of Template Method.