Task 1 : Implementation Descriptions

A consortium of shops in a large city has established an agreement with local independent van and taxi drivers to deliver products from city shops to customer destinations. I needed to implement a notification system to send notifications about delivery requests to drivers. When a store gets a product delivery orders should be created at store and a delivery request should be broadcasted to all drivers.

Since we want to **broadcast**, I used **Observer Pattern** to implement the application. Observer pattern lets us:

- Make a delivery system which can broadcast message to different drivers
- Makes our code modular and abstract by keeping implementations of shop, delivery system and drivers separate
- Provides flexibility to edit shop order structure, add more drivers, etc i.e. Updating and extending the functionalities without touching the original code
- Thus provides high cohesion less coupling and no duplicity

General Working

Three main components of the application:

1. Order : There should be an order
2. Delivery System : It is responsible to add, remove and broadcast messages to drivers.
3. Drivers : Who are aimed to receive the notification

To implement:

1. Order

Class " StoreOrderData" is used to create the order. It have following:

storeId : to maintain unique Store ID in db

storeName : to save the store name which is taking order and requesting delivery

storeAddress : pick up address

customerAddress : drop address

rate : could be interpreted as how much driver will be paid for the delivery or the value of order

Similarly, more details could be added to this class as per requirement.

2. Delivery System

Interface "StoreOrderDeliverySubject" is the current required structure of delivery system. This is implemented by "StoreOrderDeliverySystem" which actually perform the + - and notifying to drivers.

In future if we want different structure delivery system we can create a new interface thus old system also remains preserved if required.

3. Drivers

Interface "StoreOrderDeliveryObserver" and "StoreOrderDetails" are suppose to increase flexibility to extend the system. They fulfil the current requirement. "update" method is used for notification by observable and "orderDetailMessage" shows the message to driver. There is a general driver class (Driver), and for now 2 different drivers Bike and Car. More kind of driver could be added similar to bike and car.

Functions execution order / Use:

Create delivery system
StoreOrderDeliverySystem deliverySystem = new StoreOrderDeliverySystem();

Create drivers (suppose 2 drivers are created)
StoreOrderDeliveryObserver driver = new Driver();
StoreOrderDeliveryObserver cabDriver = new CabDriver();

Register Observable/Drivers
deliverySystem.registerObserver(driver);
deliverySystem.registerObserver(cabDriver);

Create store's order format
StoreOrderData storeOrder = new StoreOrderData(1, "StoreName", "Ara Street", "Gib Street", 10);

Place order through deliver system
deliverySystem.setOrder(storeOrder);

This will call notify() method which will call update() and order details message() method and let drivers know about the order.

Other:
Removing Observers
deliverySystem.removeObserver(driver);

2.2 Task 2 - UML Class Diagram.