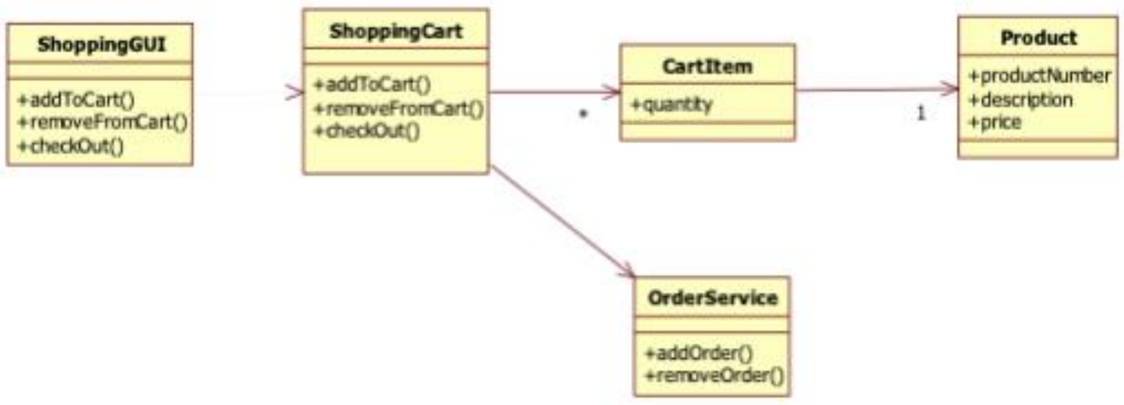
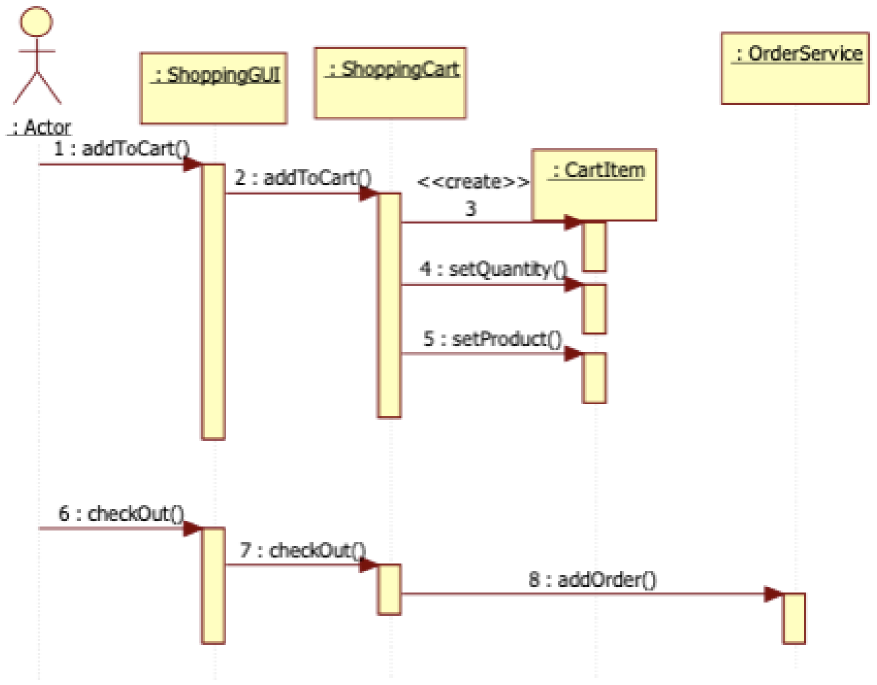
Question 1 [30 points] {40 minutes}

We designed a shoppingcart for a webshop in the following way:



The ShoppingCart contains the following 3 methods:

addToCart(Product product, int quantity);
removeFromCart(Product product, int quantity);
checkOut();



When we call checkOut() on the ShoppingCart, then the ShoppingCart calls the addOrder(ShoppingCart cart) method on the OrderService and add itself as parameter to the addOrder method. This way the OrderService receives the whole ShoppingCart object

a. Redraw the given class diagram such that the ShoppingGUI also contains buttons to undo or redo the previous actions. Your design should make it possible to undo or redo the following 3 methods of the shoppingcart: addToCart(), removeFromCart() and checkout().

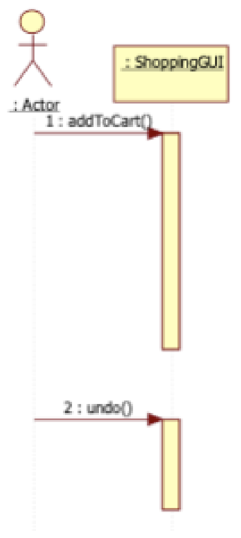
   The undo action for ***addToCart(Product product, int quantity)*** method should remove 'quantity' of these products from the shoppingcart

   The undo action for ***removeFromCart(Product product, int quantity)*** method should add 'quantity' of these products to the shoppingcart

   The undo action for ***checkout()*** method calls the removeOrder() method of the OrderService

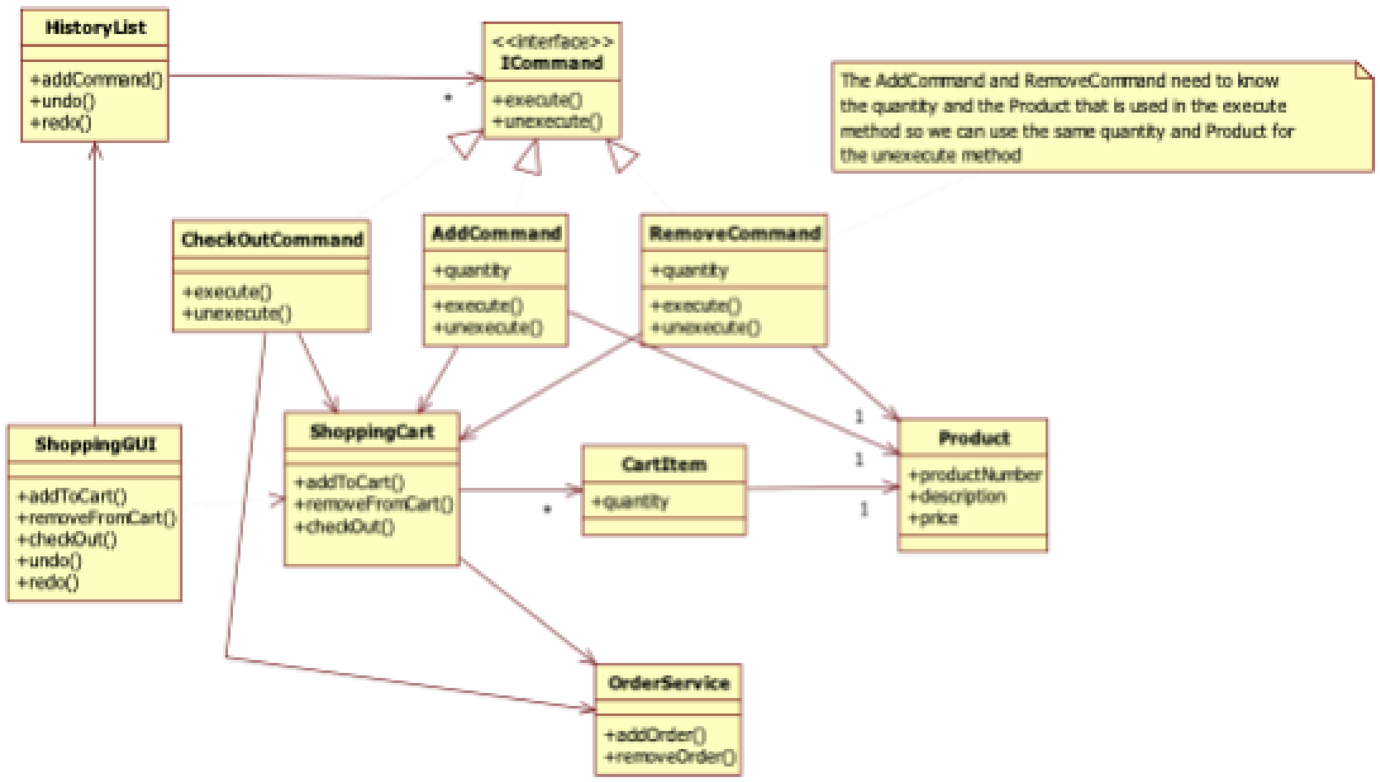
b. Redraw the sequence diagram such that the actor first calls addToCart on the ShoppingGUI and then executes the undo action.

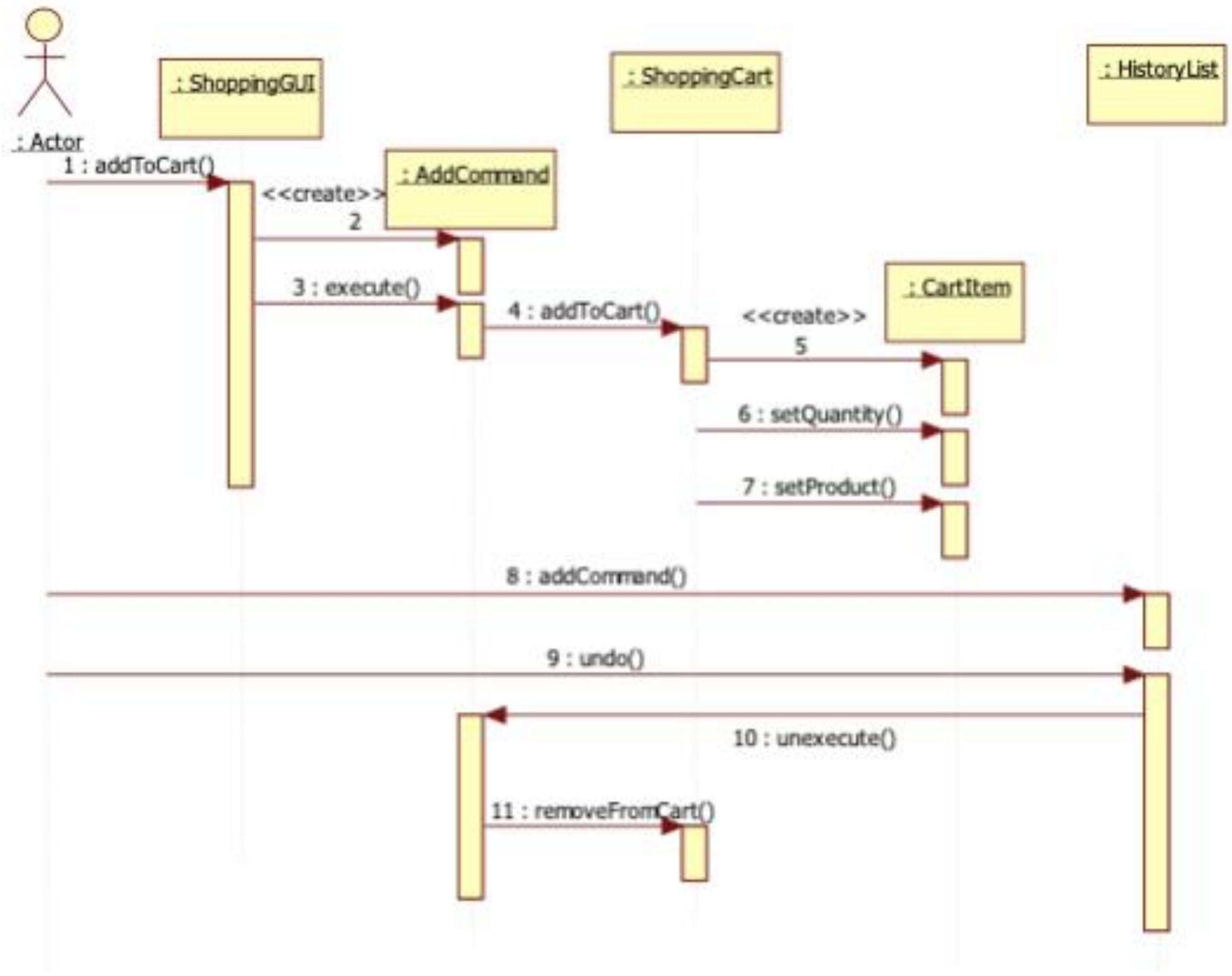
   Make sure you add all necessary UML elements to communicate the important parts of your design.



This sequence diagram is just a partial diagram. Your diagram should show all the details of how your design works.

RESULT 1


HistoryList
+addCommand()
+undo()
+redo()
<<interface>>
ICommand
+execute()
+unexecute()
*
The AddCommand and RemoveCommand need to know the quantity and the Product that is used in the execute method so we can use the same quantity and Product for the unexecute method
CheckOutCommand
+execute()
+unexecute()
AddCommand
+quantity
+execute()
+unexecute()
RemoveCommand
+quantity
+execute()
+unexecute()
ShoppingGUI
+addToCart()
+removeFromCart()
+checkOut()
+undo()
+redo()
ShoppingCart
+addToCart()
+removeFromCart()
+checkOut()
CartItem
+quantity
*
1
1
1
Product
+productNumber
+description
+price
OrderService
+addOrder()
+removeOrder()


: Actor
: ShoppingGUI
: ShoppingCart
: HistoryList
: AddCommand
: CartItem
1 : addToCart()
<<create>>
2
3 : execute()
4 : addToCart()
<<create>>
5
6 : setQuantity()
7 : setProduct()
8 : addCommand()
9 : undo()
10 : unexecute()
11 : removeFromCart()

Question 2 [20 points] {20 minutes}

Suppose we have the following Stock class:

```
public class Stock {
      private String symbol;
      private double price;

      //getter and setter methods are not shown
}
```

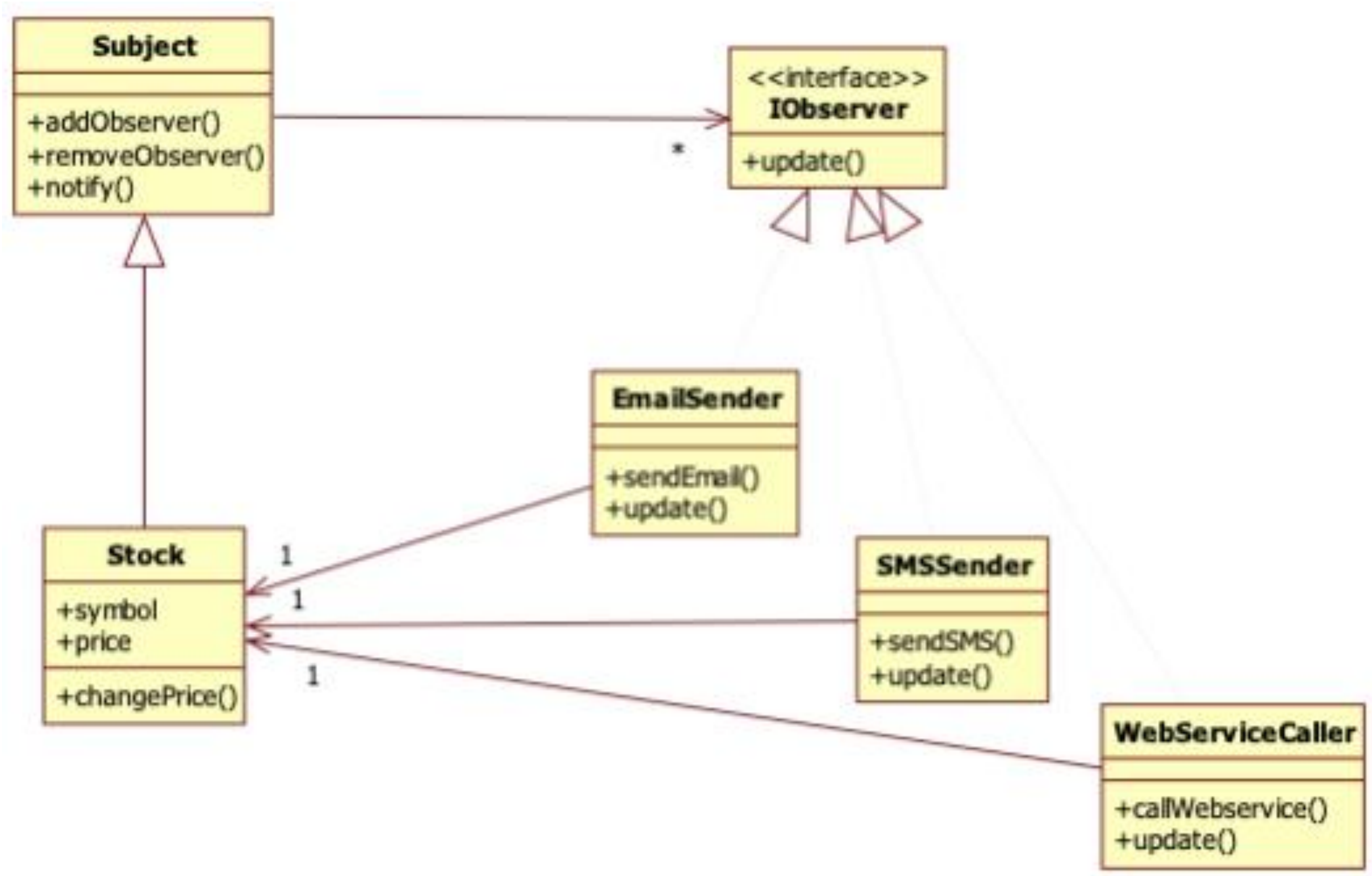
We need to add the following functionality:

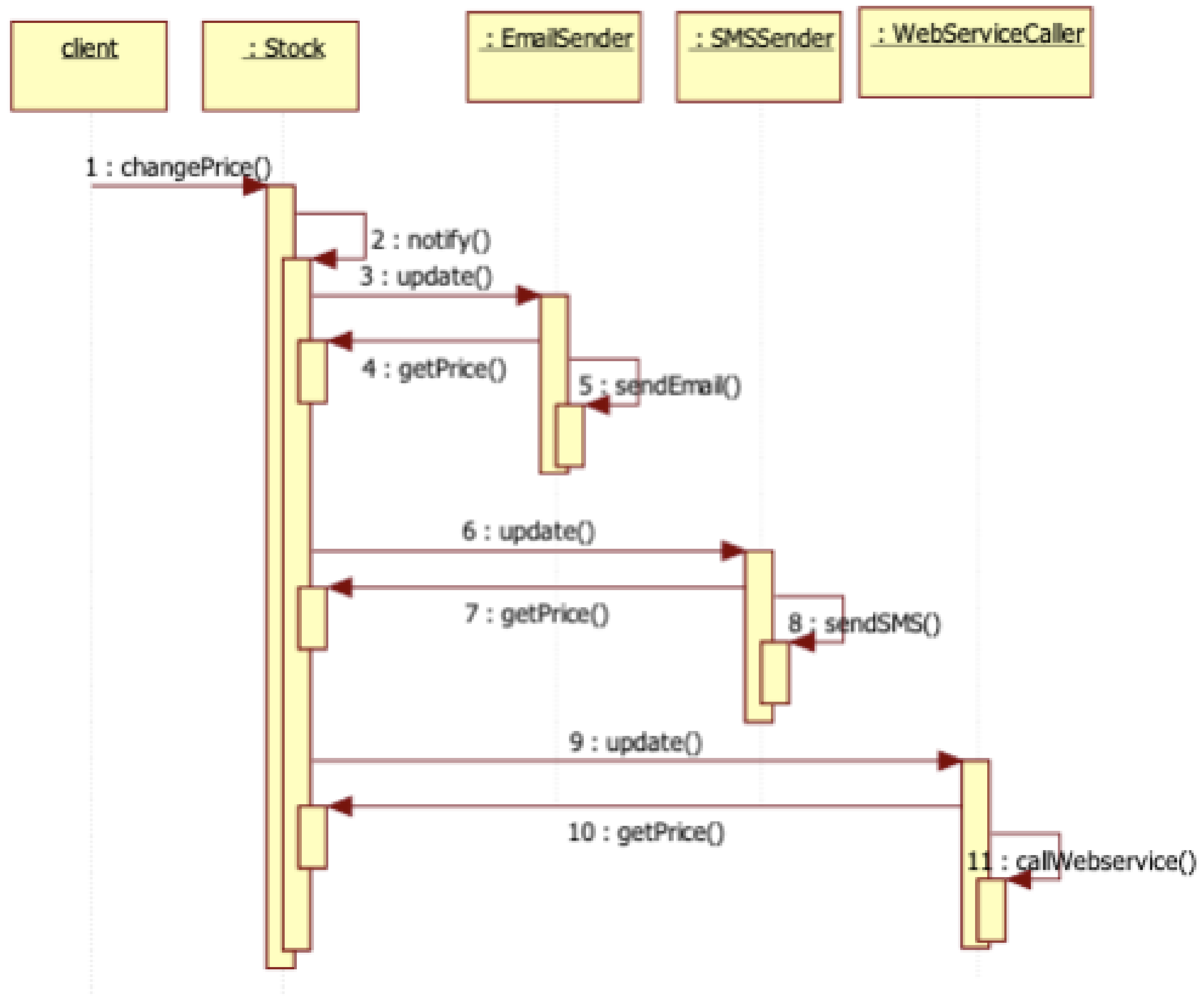
> Every time when the price of the Stock changes, other objects (EmailSenders, SMSSenders, WebServiceCallers) want to know about this change. The **number** of objects and the **type** of objects that need to know when the stock price changes should be variable and is not known upfront.

Draw the **class diagram** and a corresponding **sequence diagram** that clearly communicates your design.

Make sure you add all necessary UML elements to communicate the important parts of your design.

RESULT 2


Subject
+addObserver()
+removeObserver()
+notify()
<<interface>>
IObserver
+update()
*
EmailSender
+sendEmail()
+update()
Stock
+symbol
+price
+changePrice()
1
1
1
SMSSender
+sendSMS()
+update()
WebServiceCaller
+callWebservice()
+update()


client
: Stock
: EmailSender
: SMSSender
: WebServiceCaller
1 : changePrice()
2 : notify()
3 : update()
4 : getPrice()
5 : sendEmail()
6 : update()
7 : getPrice()
8 : sendSMS()
9 : update()
10 : getPrice()
11 : callWebservice()

Question 3 [20 points] {20 minutes}

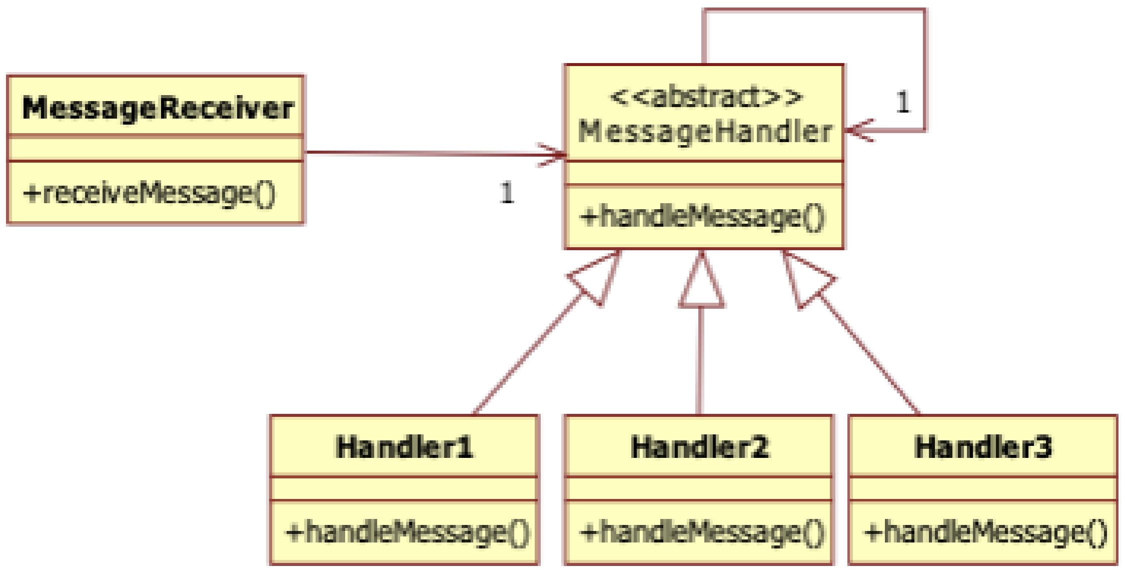
Suppose we have to design an order application that receives orders in the form of order messages. All of our clients send these order messages in a different format, so the message sent by client 1 has a different format compared to the message from client 2. The only thing that all messages have in common is that every message has a header field with name "clientid" which contains the unique client number.
The receiver object that receives these messages has to make sure the correct logic is executed based on this clientid. For every message with a different clientid, some different logic needs to be executed. We can have a lot of different clients.
It is important that the design of the application allows us to easily add more clients with different message structures with the least amount of code change. What pattern would you apply in your design?

a. Draw the class diagram of your design. You only should show the part that receives and processes the message.
b. Draw the sequence diagram of your design. It should show the scenario of what happens when we receive an order message.

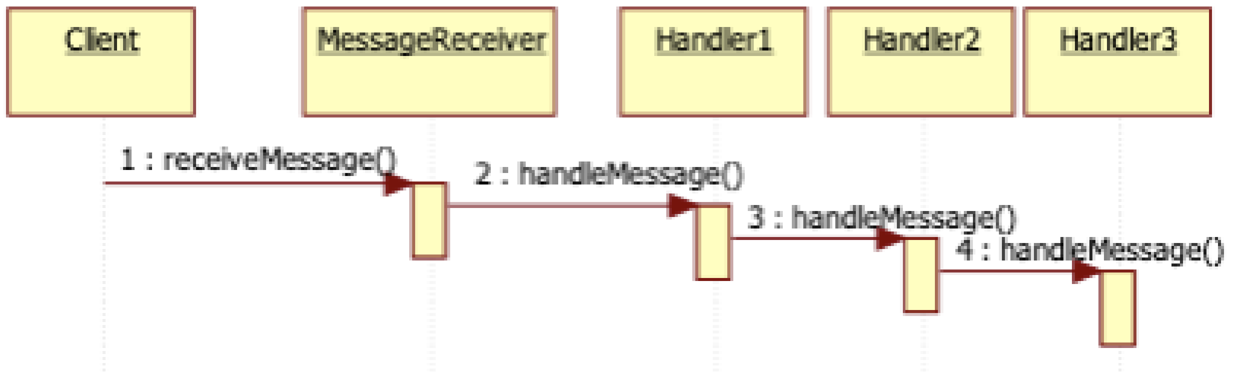
RESULT 3

a. Chain of responsibility

b.



c.



Question 4 [10 points] {15 minutes}

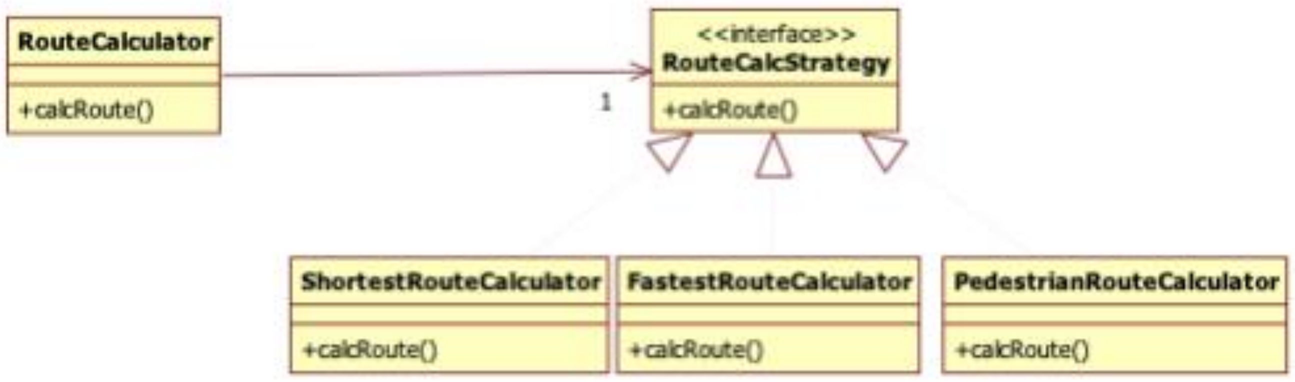
We need to design a route planner application that computes the route between point A and point B. The route planner application allows us to compute the fastest route, the shortest route, the best route for bicycles, and the best route for pedestrians, etc. It should be easy to add new ways to compute the route between point A and B.

a. What pattern would you apply in your design?
b. Draw the class diagram of your design.

RESULT 4

a. Strategy

b.


RouteCalculator
+calcRoute()
1
<<interface>>
RouteCalcStrategy
+calcRoute()
ShortestRouteCalculator
+calcRoute()
FastestRouteCalculator
+calcRoute()
PedestrianRouteCalculator
+calcRoute()