CS 525 - ASD

Advanced Software Development

MS.CS Program

Department of Computer Science

Rene de Jong, MsC.

Maharishi University
OF MANAGEMENT

CS 525 - ASD

Advanced Software Development

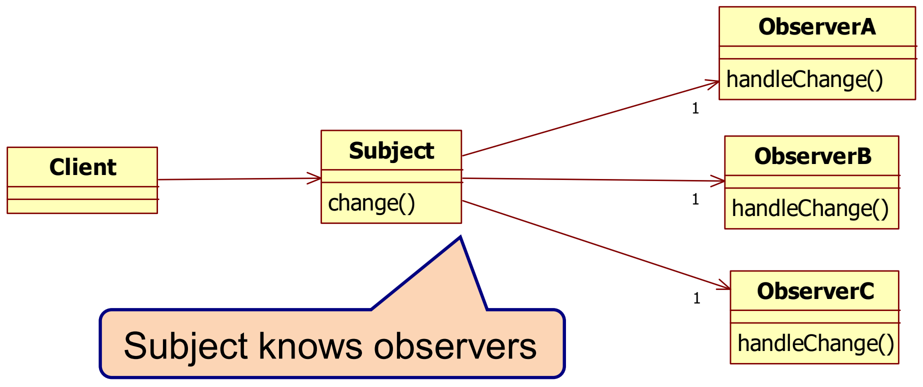
Maharishi University
OF MANAGEMENT

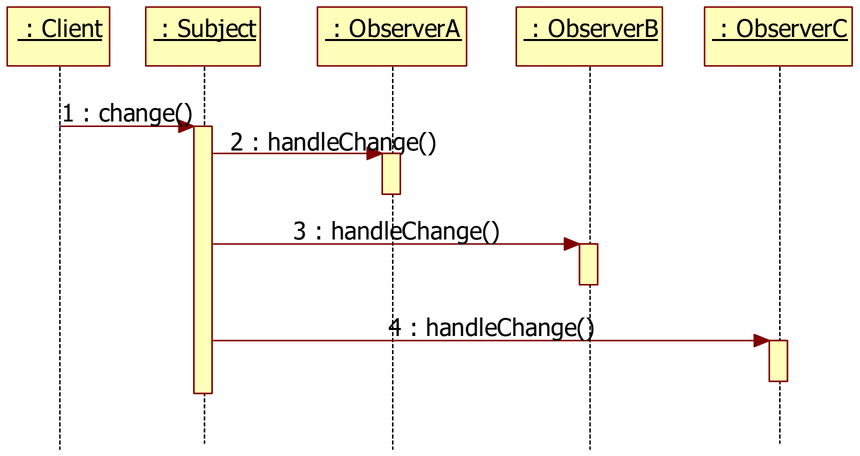
Lesson 3

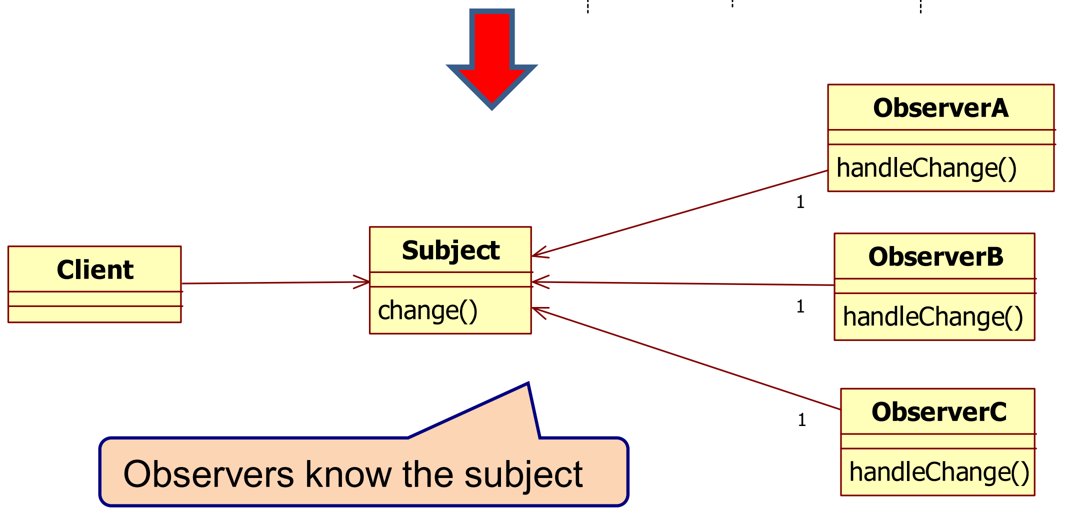
Observer pattern

- The Observer design pattern lets several observer objects be notified when a subject is changed in some way.

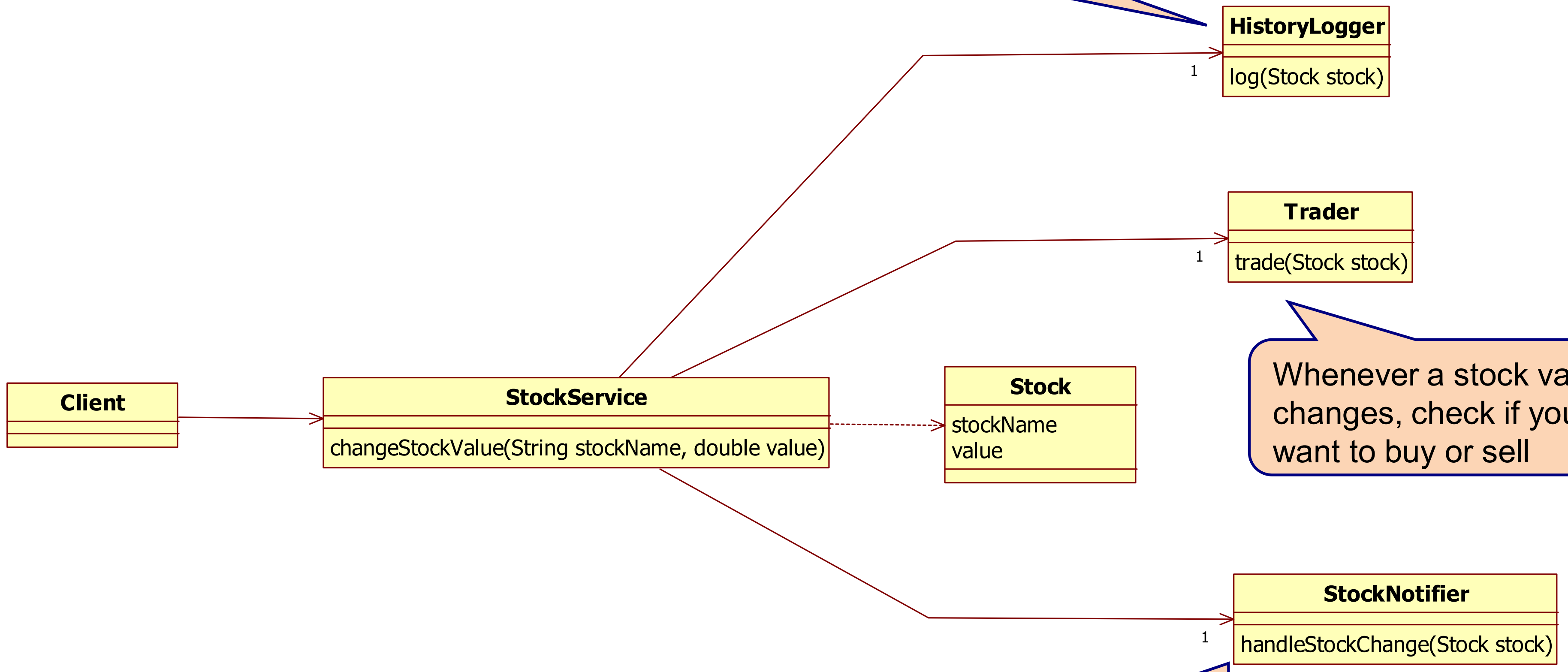
Observer pattern







Example application

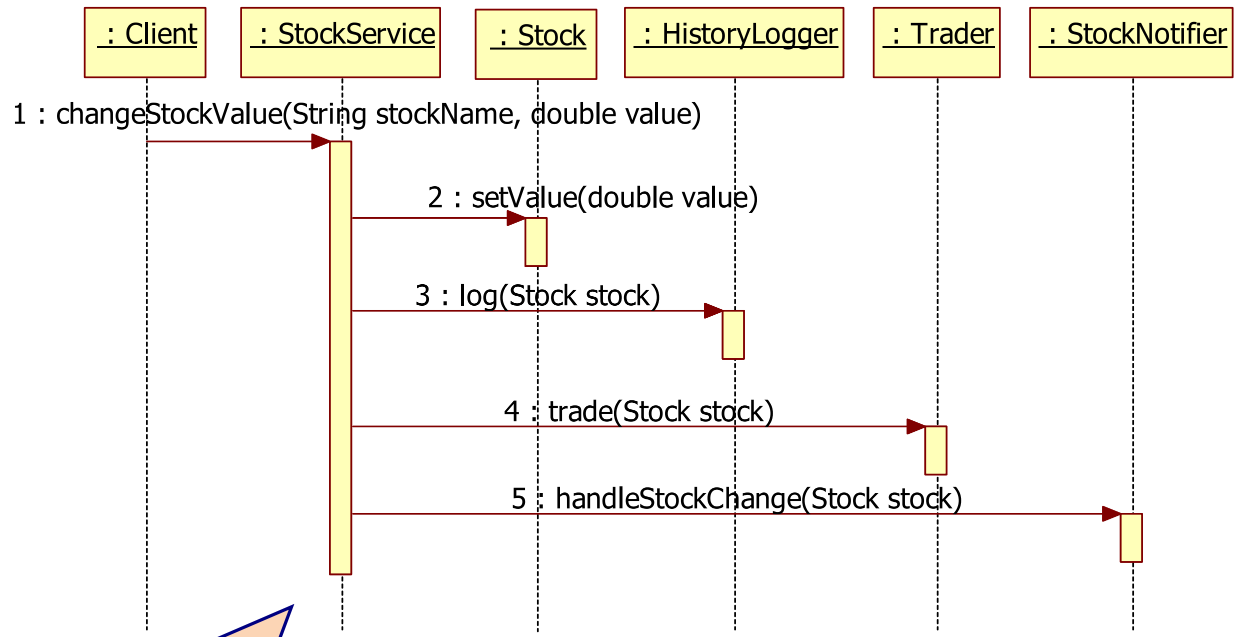


Whenever a stock value changes, write this to a log file

Whenever a stock value changes, check if you want to buy or sell

Whenever a stock value changes, notify people who want to know about this price change

Example application


: Client
: StockService
: Stock
: HistoryLogger
: Trader
: StockNotifier
1 : changeStockValue(String stockName, double value)
2 : setValue(double value)
3 : log(Stock stock)
4 : trade(Stock stock)
5 : handleStockChange(Stock stock)

Whenever you add a new class that wants to know about stock value changes, you need to change the StockService

The observers and Stock

```java
public class HistoryLogger {

  public void log(Stock stock) {
    System.out.println("HistoryLogger log stock :" + stock);
  }
}
```

```java
public class Trader {

  public void trade(Stock stock) {
    System.out.println("Trader trade stock :" + stock);
  }
}
```

```java
public class StockNotifier {

  public void handleStockChange(Stock stock) {
    System.out.println("StockNotifier handle stock :" + stock);
  }
}
```

```java
public class Stock {
  private String stockName;
  private double value;
  ...
}
```

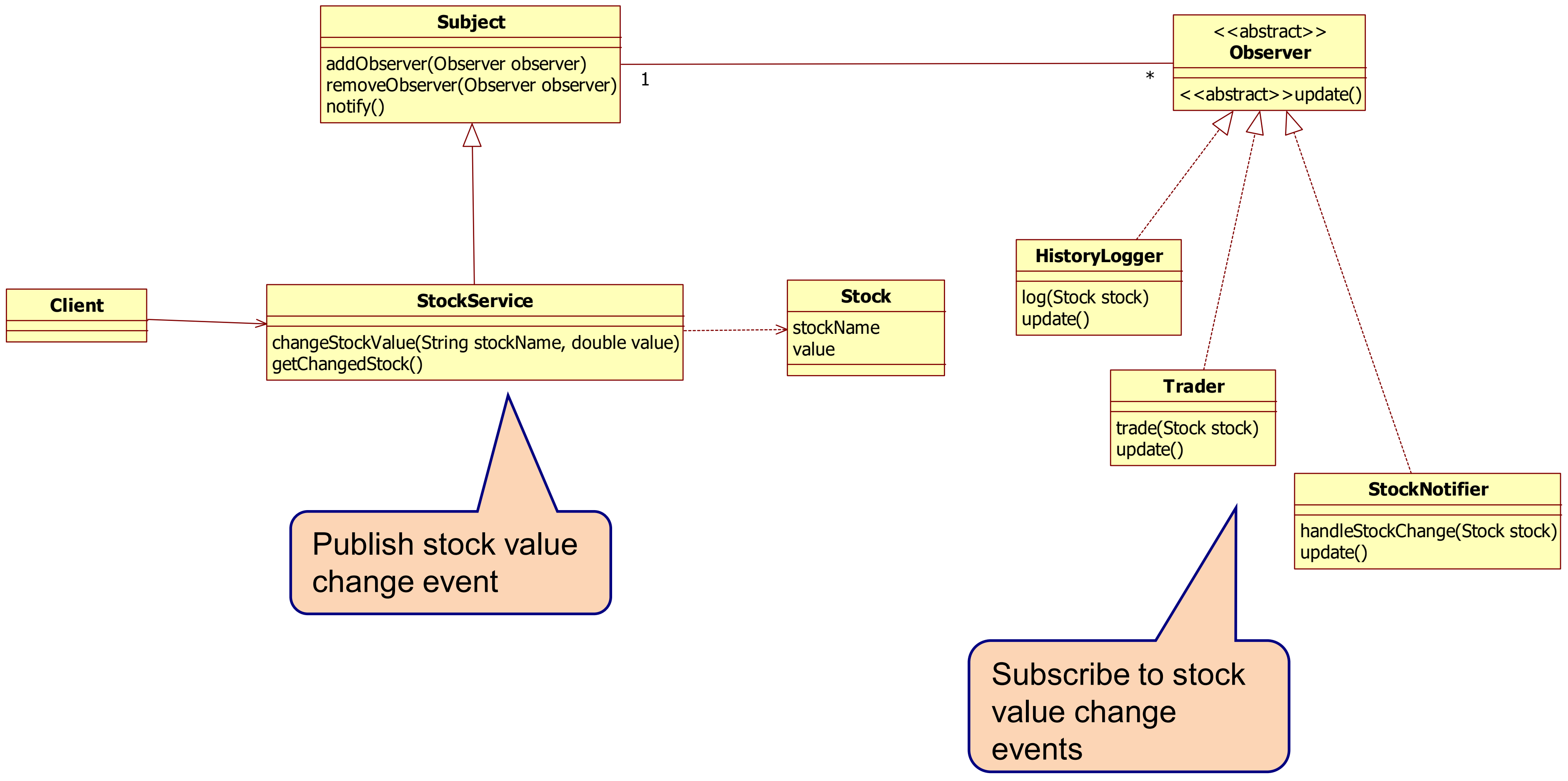
StockService

```java
public class StockService {
  private HistoryLogger historyLogger;
  private Trader trader;
  private StockNotifier stockNotifier;

  public void changeStockValue(String stockName, double value) {
    Stock stock = new Stock(stockName, value);
    historyLogger.log(stock);
    trader.trade(stock);
    stockNotifier.handleStockChange(stock);
  }

  ...
}
```

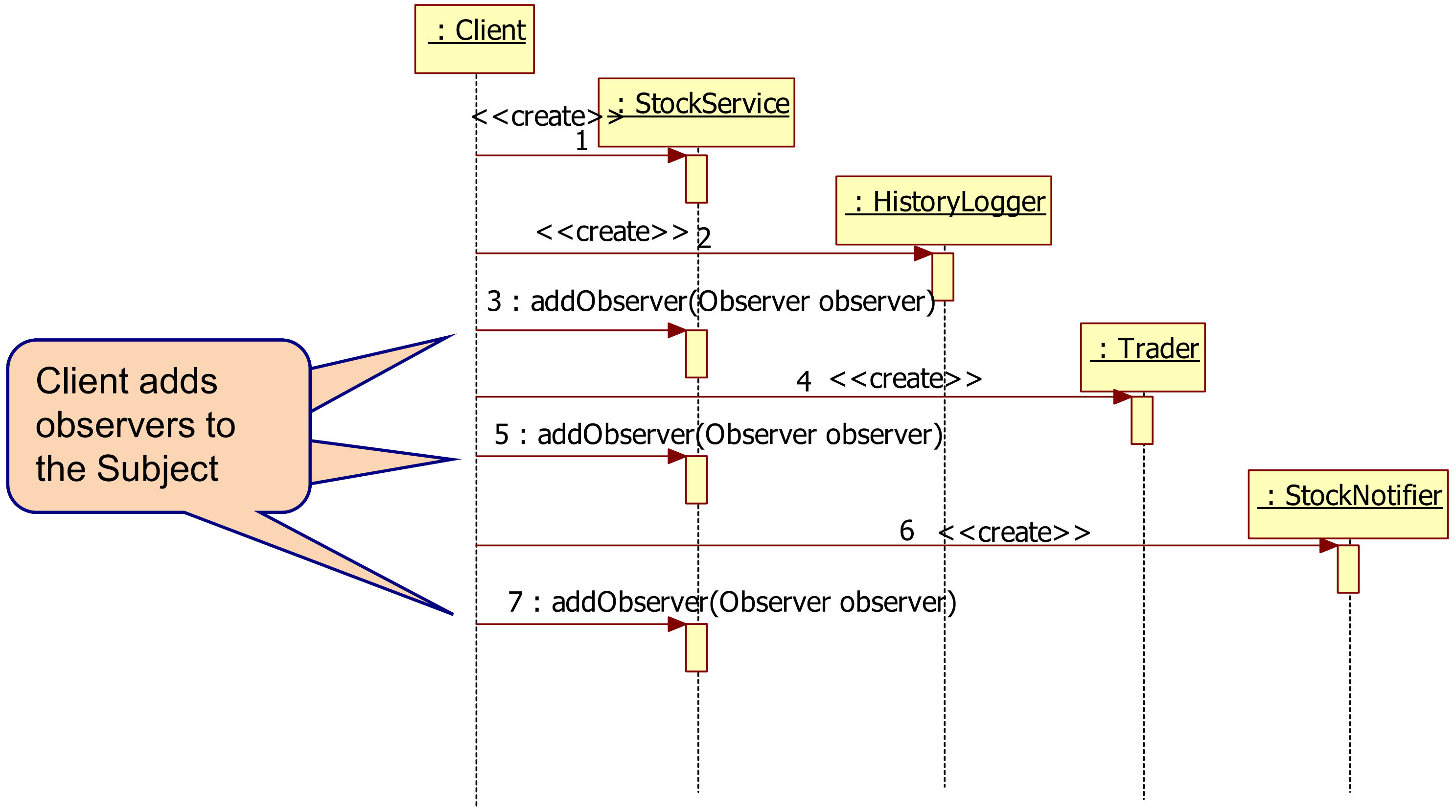
Application

```java
public class Application {

  public static void main(String[] args) {
    StockService stockService = new StockService();
    HistoryLogger historyLogger= new HistoryLogger();
    Trader trader = new Trader();
    StockNotifier stockNotifier = new StockNotifier();

    stockService.setHistoryLogger(historyLogger);
    stockService.setTrader(trader);
    stockService.setStockNotifier(stockNotifier);

    stockService.changeStockValue("AMZN", 2310.80);
    stockService.changeStockValue("MSFT", 890.45);
  }
}
```

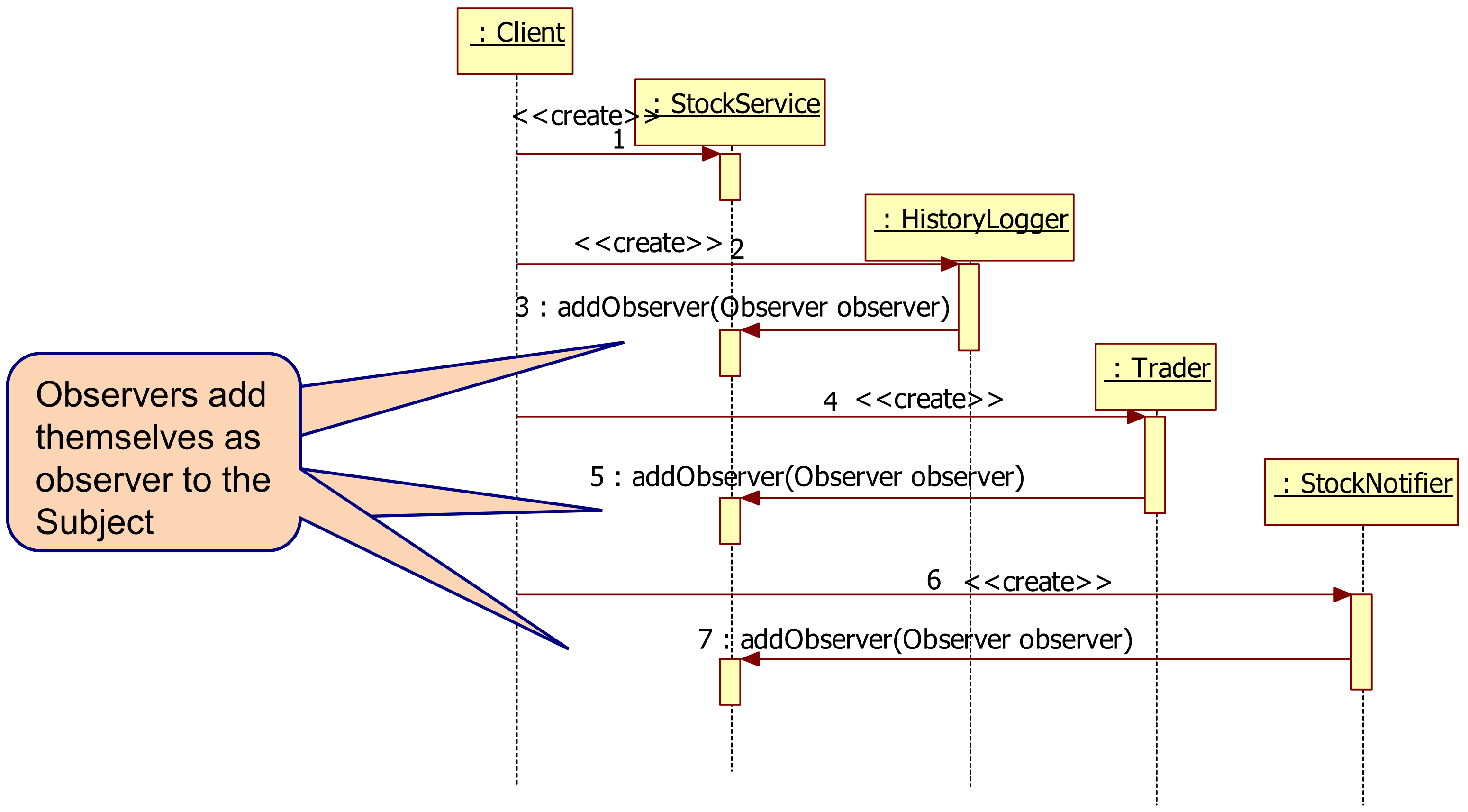
Observer pattern


Subject
addObserver(Observer observer)
removeObserver(Observer observer)
notify()
1
*
<<abstract>>
Observer
<<abstract>>update()
HistoryLogger
log(Stock stock)
update()
Client
StockService
changeStockValue(String stockName, double value)
getChangedStock()
Stock
stockName
value
Trader
trade(Stock stock)
update()
StockNotifier
handleStockChange(Stock stock)
update()
Publish stock value change event
Subscribe to stock value change events

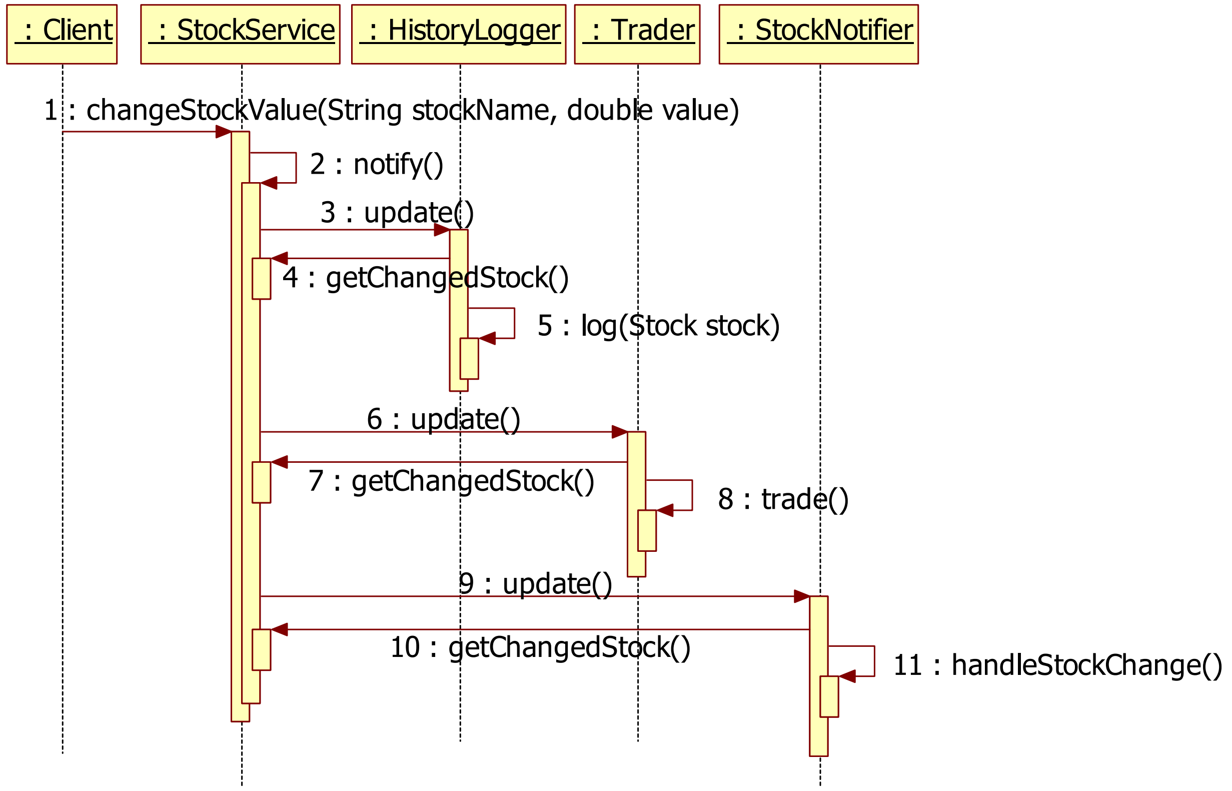
Connecting the Subject and Observers


: Client
: StockService
<<create>>
1
: HistoryLogger
<<create>> 2
3 : addObserver(Observer observer)
: Trader
4 <<create>>
5 : addObserver(Observer observer)
: StockNotifier
6 <<create>>
7 : addObserver(Observer observer)
Client adds observers to the Subject

Connecting the Subject and Observers


: Client
: StockService
<<create>>
1
: HistoryLogger
<<create>>
2
3 : addObserver(Observer observer)
: Trader
4 <<create>>
5 : addObserver(Observer observer)
: StockNotifier
6 <<create>>
7 : addObserver(Observer observer)
Observers add themselves as observer to the Subject

Calling the observers



Subject, Observer and Stock

```
public class Subject  {
  private Collection<Observer> observerlist = new ArrayList<Observer>();

  public void addObserver(Observer observer){
    observerlist.add(observer);
  }

  public void donotify(){
    for (Observer observer: observerlist){
      observer.update();
    }
  }
}
```

```
public abstract class Observer {
  private StockService stockService;

  public Observer(StockService stockService) {
    this.stockService = stockService;
  }

  public abstract void update();
}
```

StockService and Stock

```java
public class StockService extends Subject{
  private Stock lastChangedStock;

  public void changeStockValue(String stockName, double value) {
    lastChangedStock = new Stock(stockName, value);
    donotify();
  }

  public Stock getLastChangedStock() {
    return lastChangedStock;
  }
}
```

```java
public class Stock {
  private String stockName;
  private double value;
  ...
}
```

HistoryLogger

```
public class HistoryLogger extends Observer {


  public HistoryLogger(StockService stockService) {
    super(stockService);
  }

  public void log(Stock stock) {
    System.out.println("HistoryLogger log stock :" + stock);
  }

  @Override
  public void update() {
    Stock stock = stockService.getLastChangedStock();
    log(stock);
  }
}
```

Trader

```java
public class Trader extends Observer {

  public Trader(StockService stockService) {
    super(stockService);
  }

  public void trade(Stock stock) {
    System.out.println("Trader trade stock :" + stock);
  }

  @Override
  public void update() {
    Stock stock = stockService.getLastChangedStock();
    trade(stock);
  }
}
```

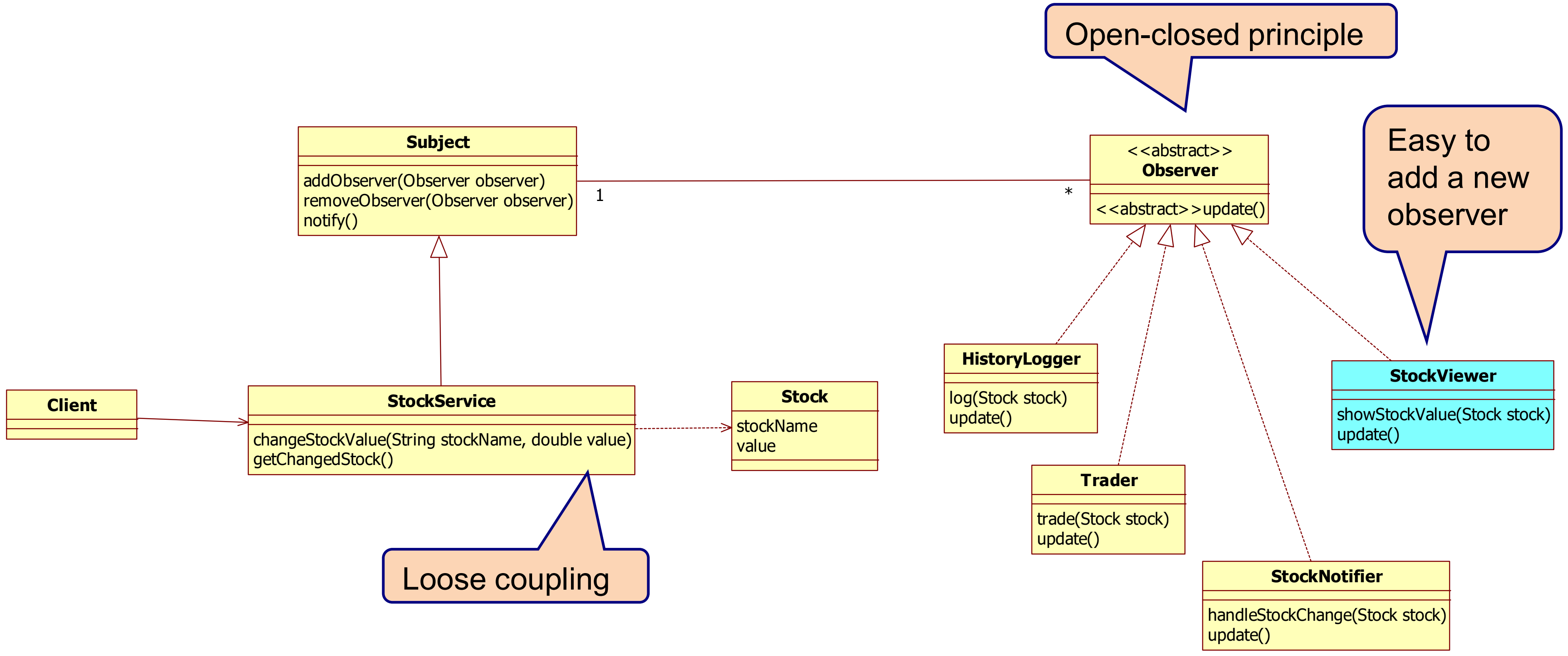
StockNotifier

```
public class StockNotifier extends Observer {


  public StockNotifier(StockService stockService) {
    super(stockService);
  }

  public void handleStockChange(Stock stock) {
    System.out.println("StockNotifier handle stock :" + stock);
  }

  @Override
  public void update() {
    Stock stock = stockService.getLastChangedStock();
    handleStockChange(stock);
  }
}
```

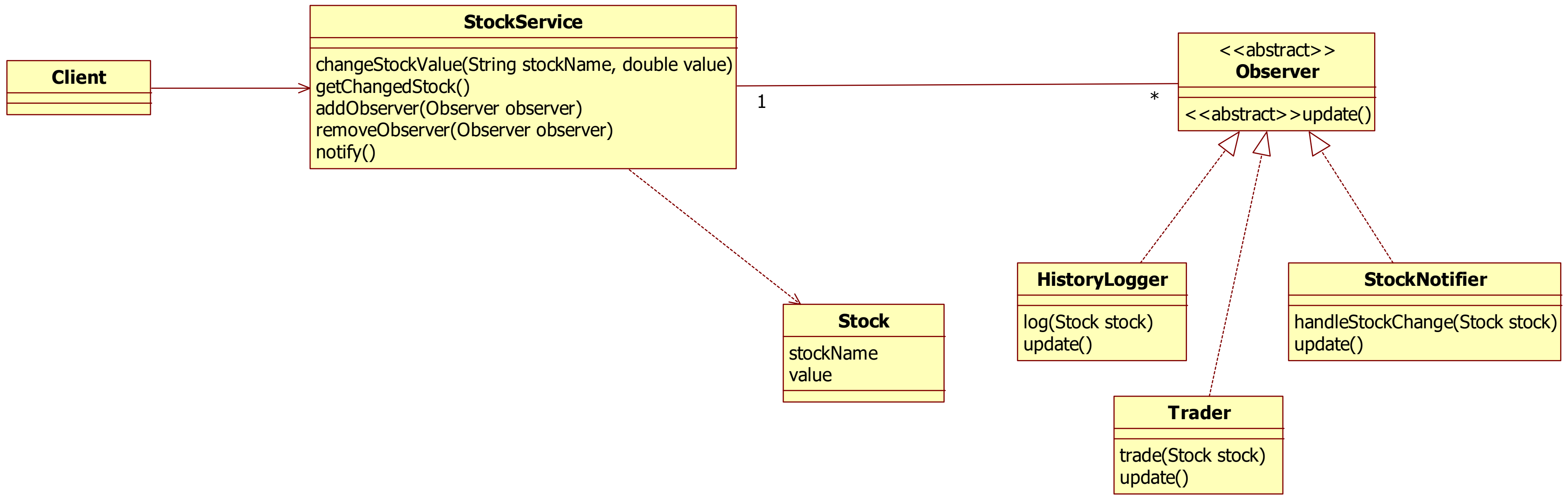
Application

```
public class Application {

  public static void main(String[] args) {
    StockService stockService = new StockService();
    HistoryLogger historyLogger= new HistoryLogger(stockService);
    Trader trader = new Trader(stockService);
    StockNotifier stockNotifier = new StockNotifier(stockService);

    stockService.addObserver(historyLogger);
    stockService.addObserver(trader);
    stockService.addObserver(stockNotifier);

    stockService.changeStockValue("AMZN", 2310.80);
    stockService.changeStockValue("MSFT", 890.45);
  }
}
```

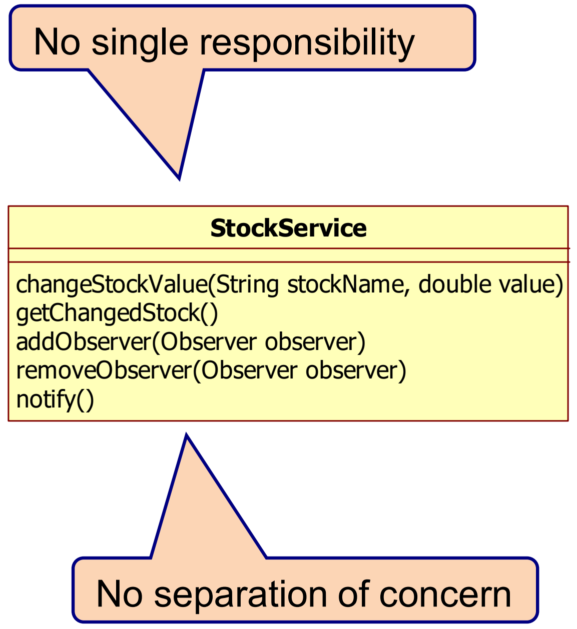
Advantage of Observer

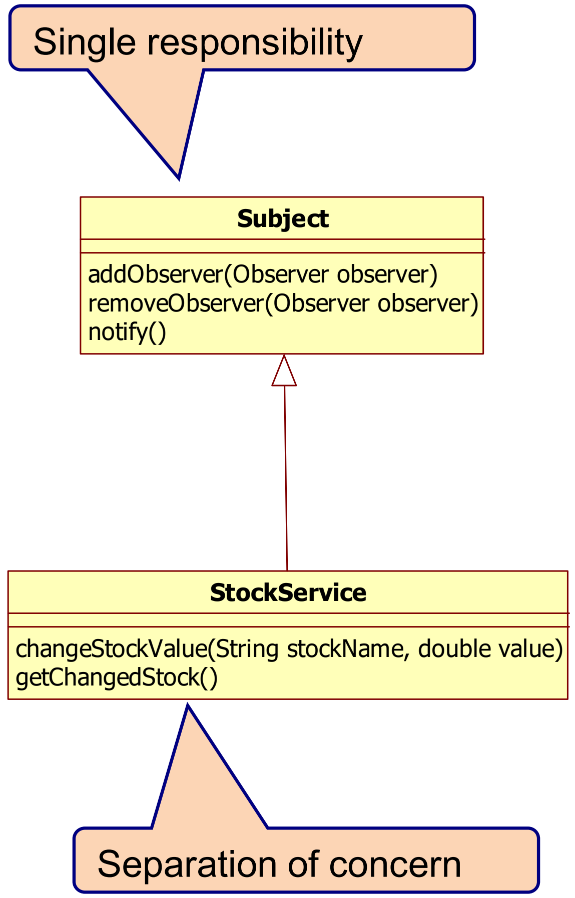


What is wrong with this?

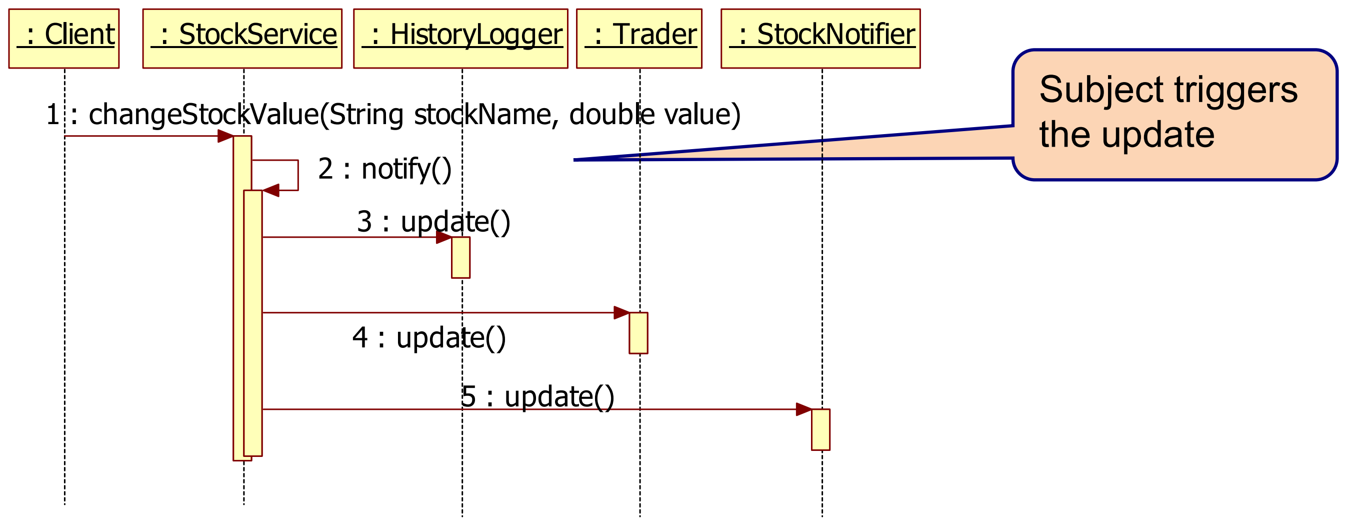


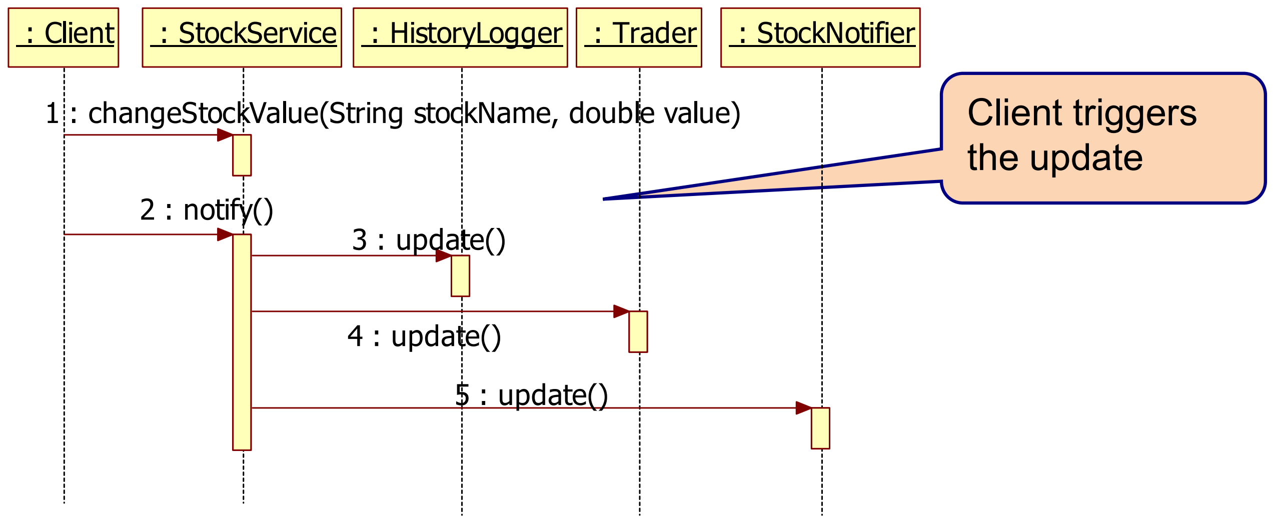
Separate Subject


No single responsibility
StockService
changeStockValue(String stockName, double value)
getChangedStock()
addObserver(Observer observer)
removeObserver(Observer observer)
notify()
No separation of concern


Single responsibility
Subject
addObserver(Observer observer)
removeObserver(Observer observer)
notify()
StockService
changeStockValue(String stockName, double value)
getChangedStock()
Separation of concern

Who triggers the update?





Main point

- The observer pattern makes observables (publishers) independent of observers (subscribers)

- All human beings have the ability to observe and live the intelligence of nature