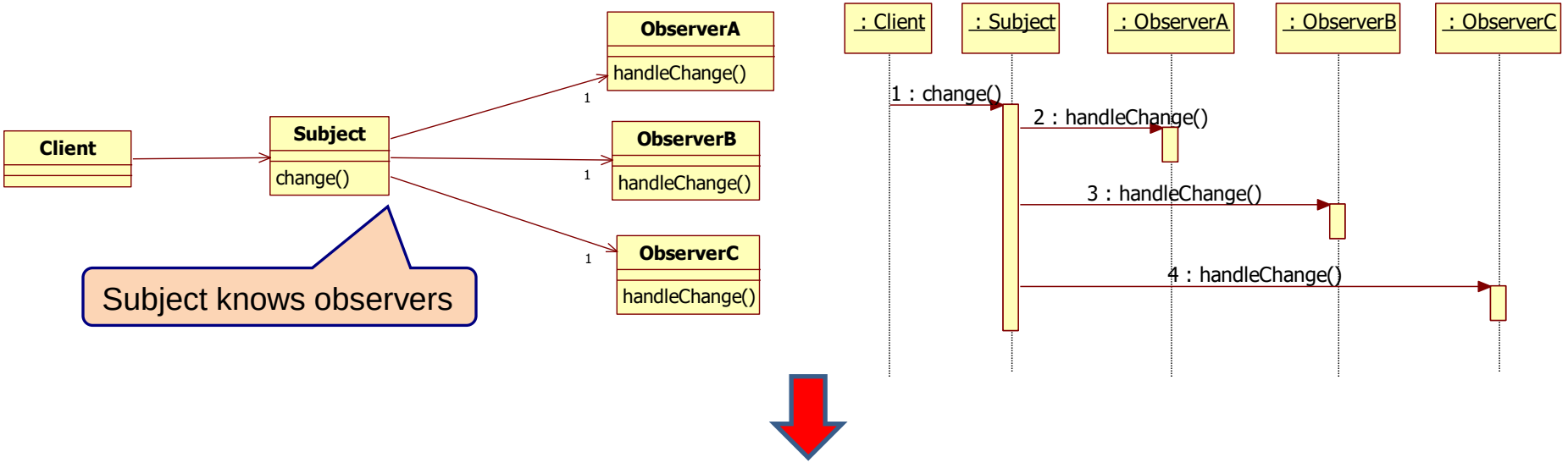
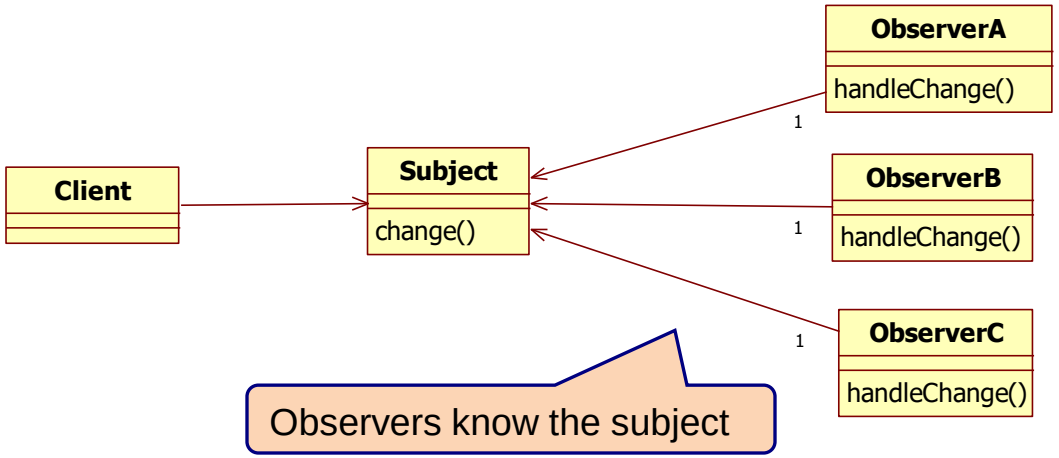
LESSON 4
OBSERVER PATTERN

Observer pattern

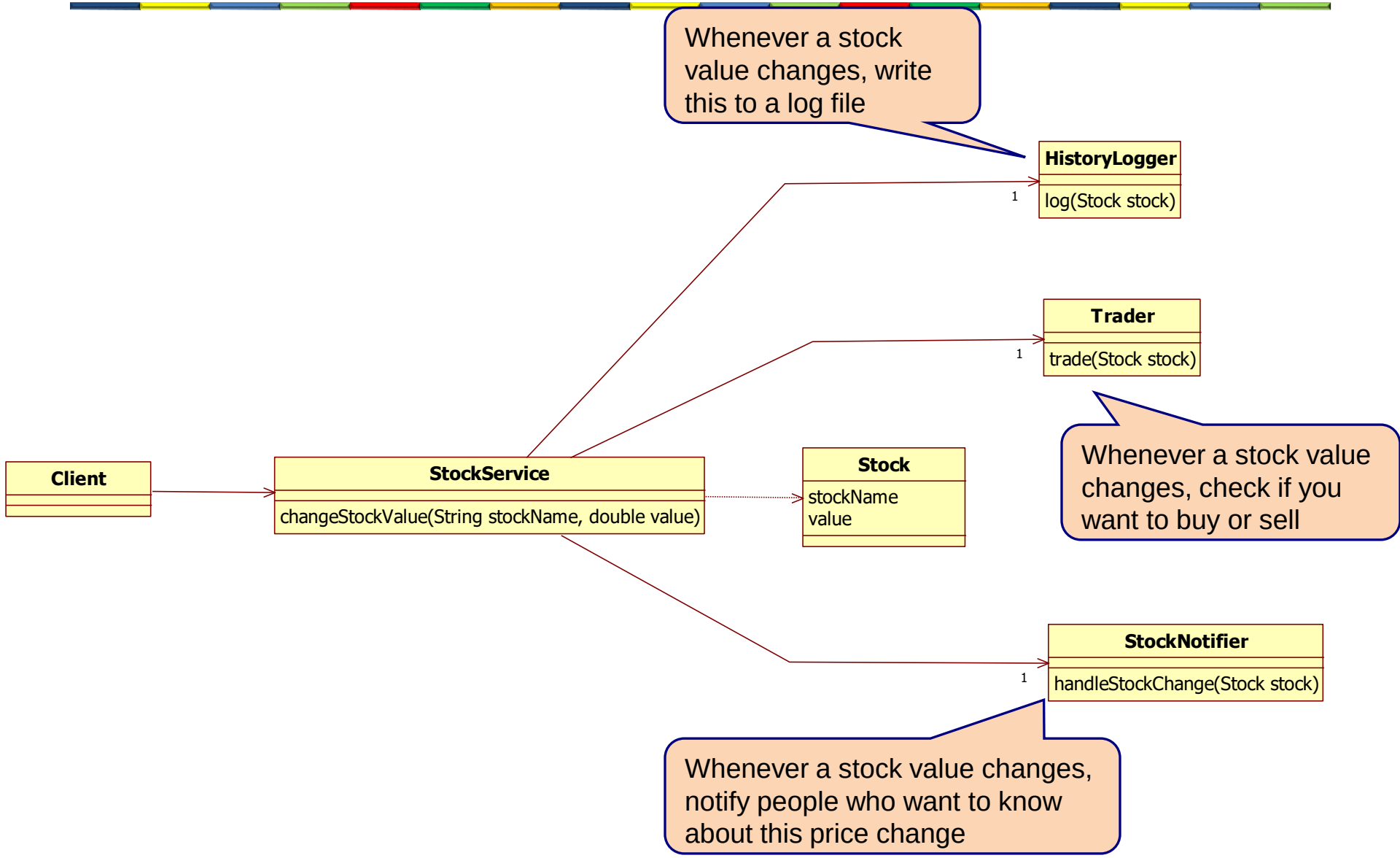
- The Observer design pattern lets several observer objects be notified when a subject is changed in some way.

Observer pattern

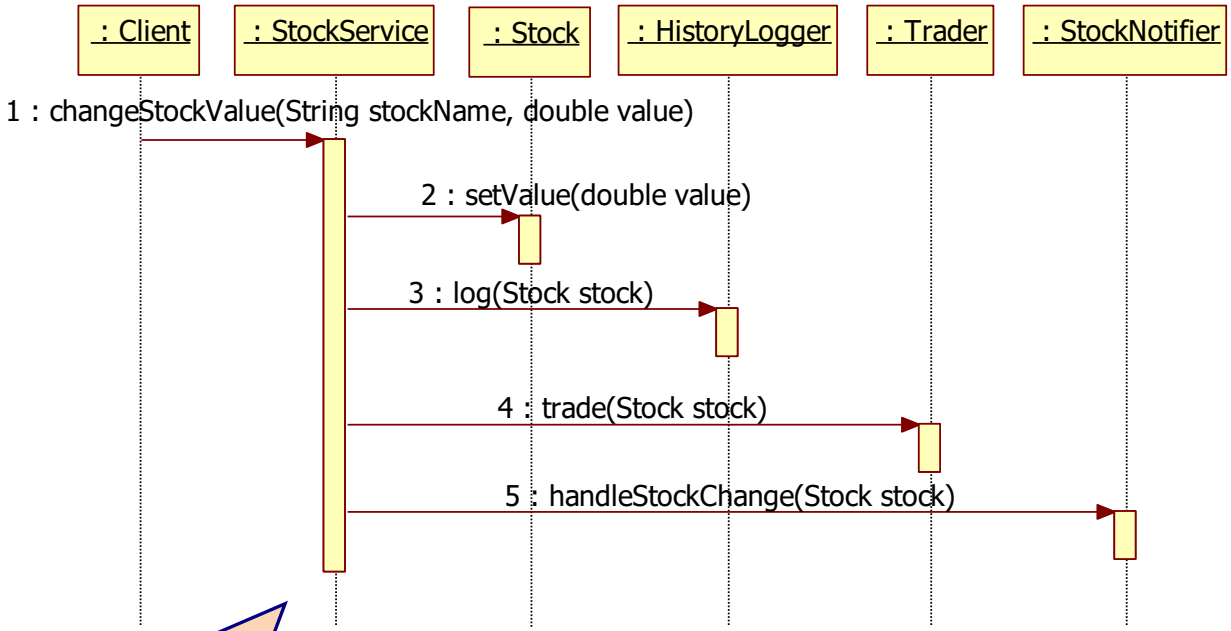




Example application



Example application



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

The observers and Stock

```
public class HistoryLogger {

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

```
public class Trader {

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

```
public class StockNotifier {

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

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

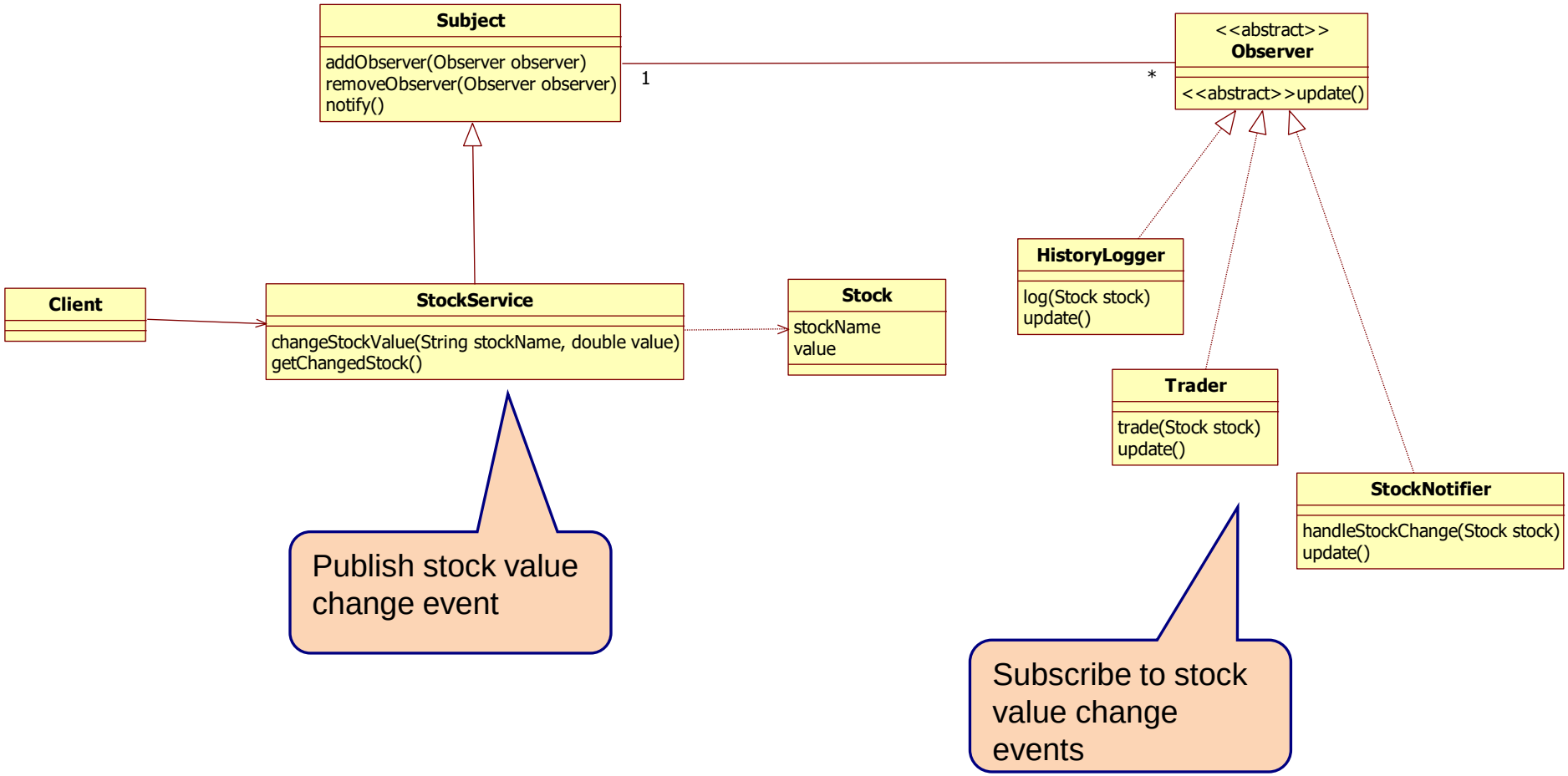
StockService

```
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);
  }

  ...
}
```

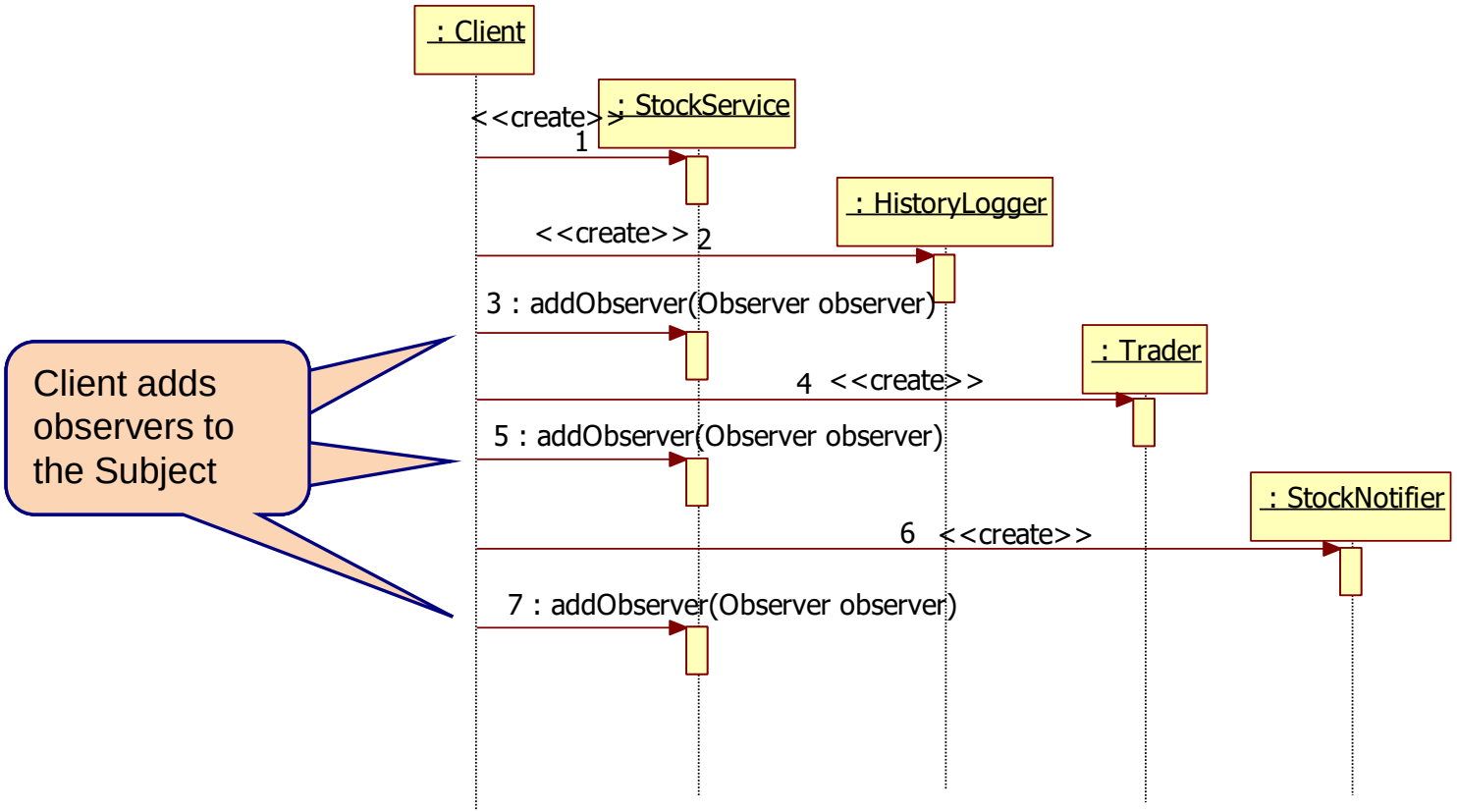
Application

```
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);
  }
}
```

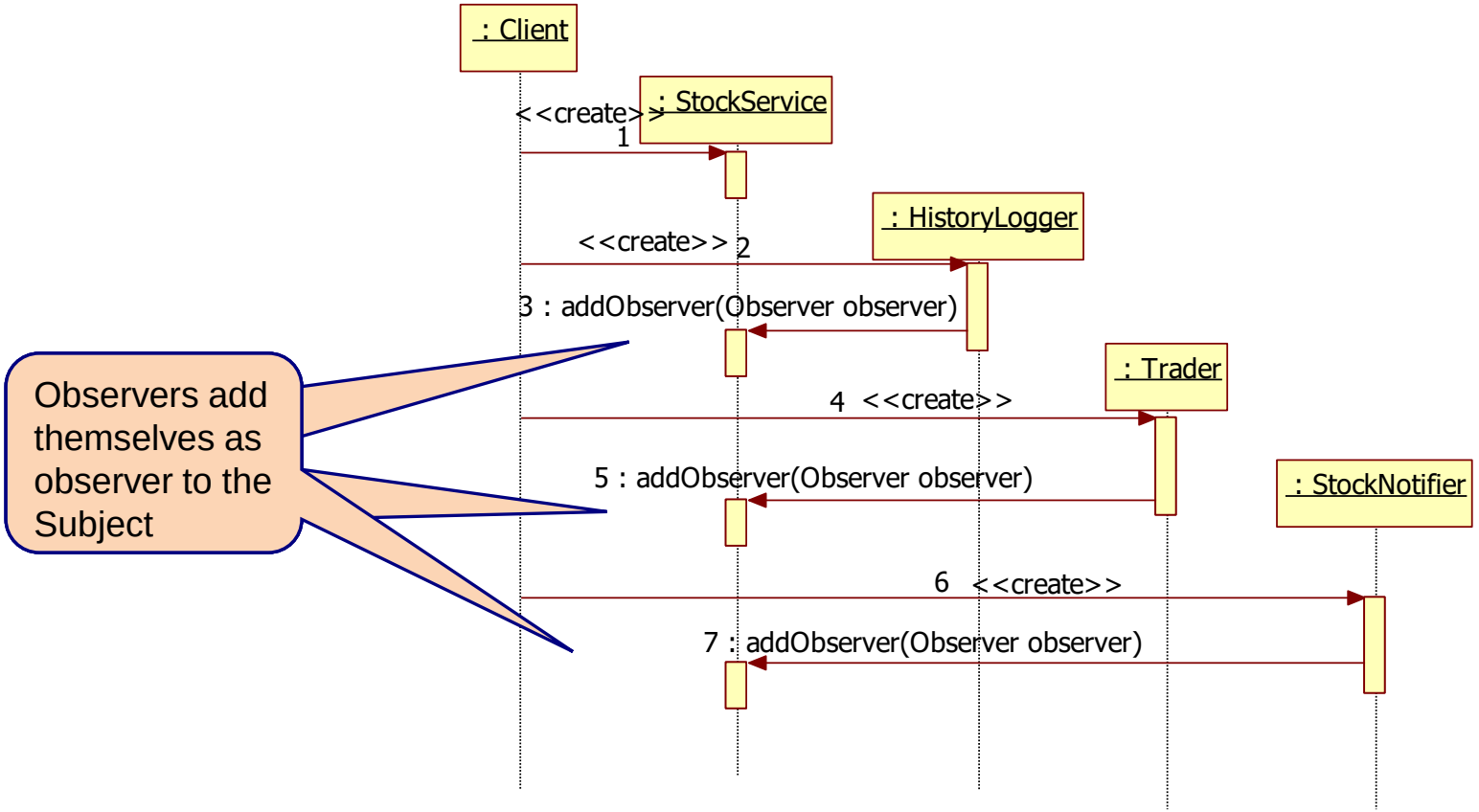
Observer pattern



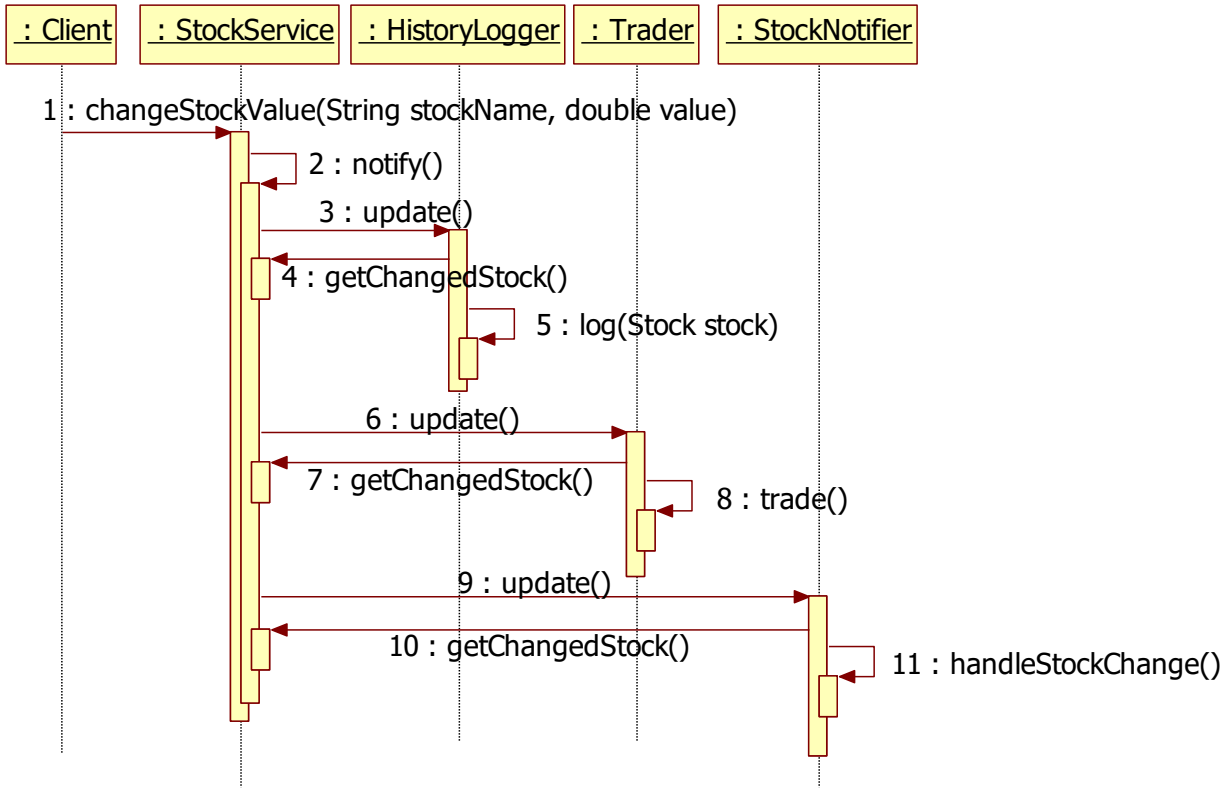
Connecting the Subject and Observers



Connecting the Subject and Observers



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

```
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;
  }
}
```

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

HistoryLogger

```
public class HistoryLogger extends Observer {
  private StockService stockService;

  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

```
public class Trader extends Observer {
  private StockService stockService;

  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);
  }
}
```

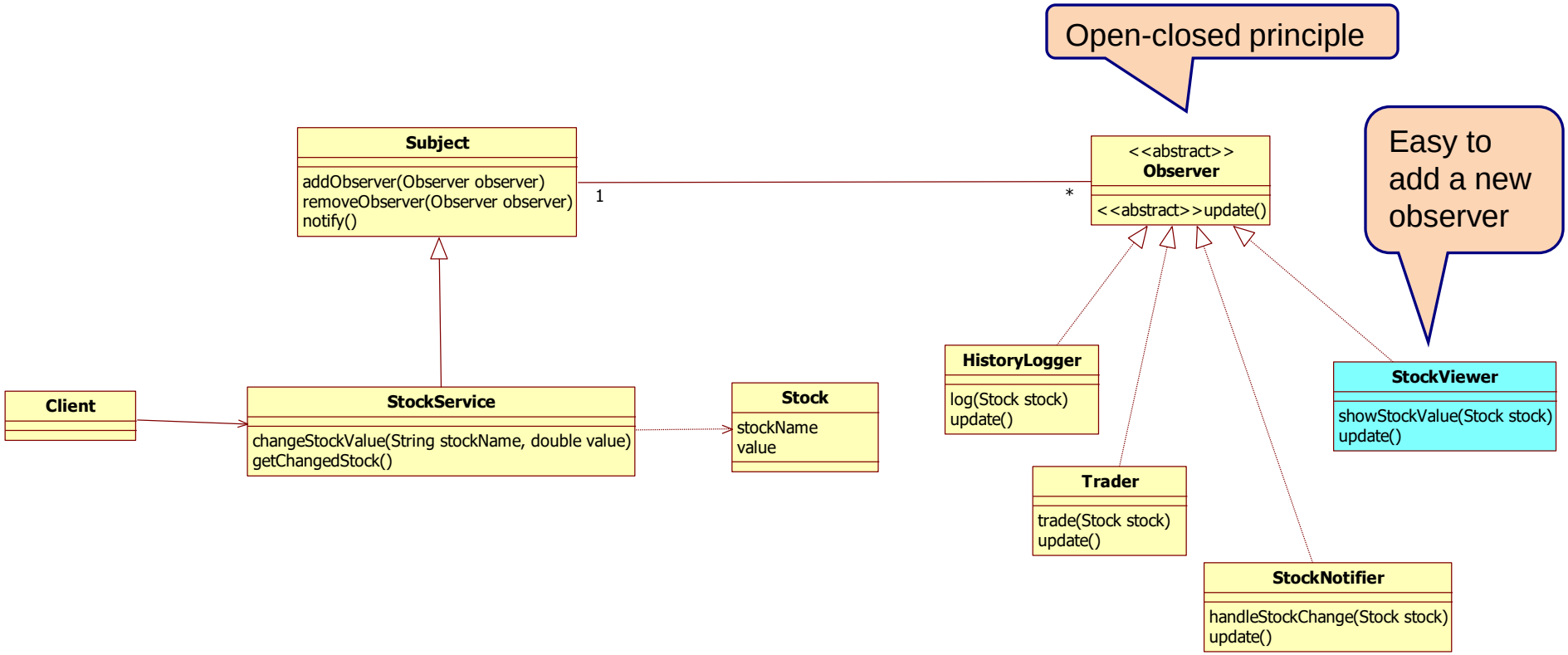
StockNotifier

```
public class StockNotifier extends Observer {
  private StockService stockService;

  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);
  }
}
```

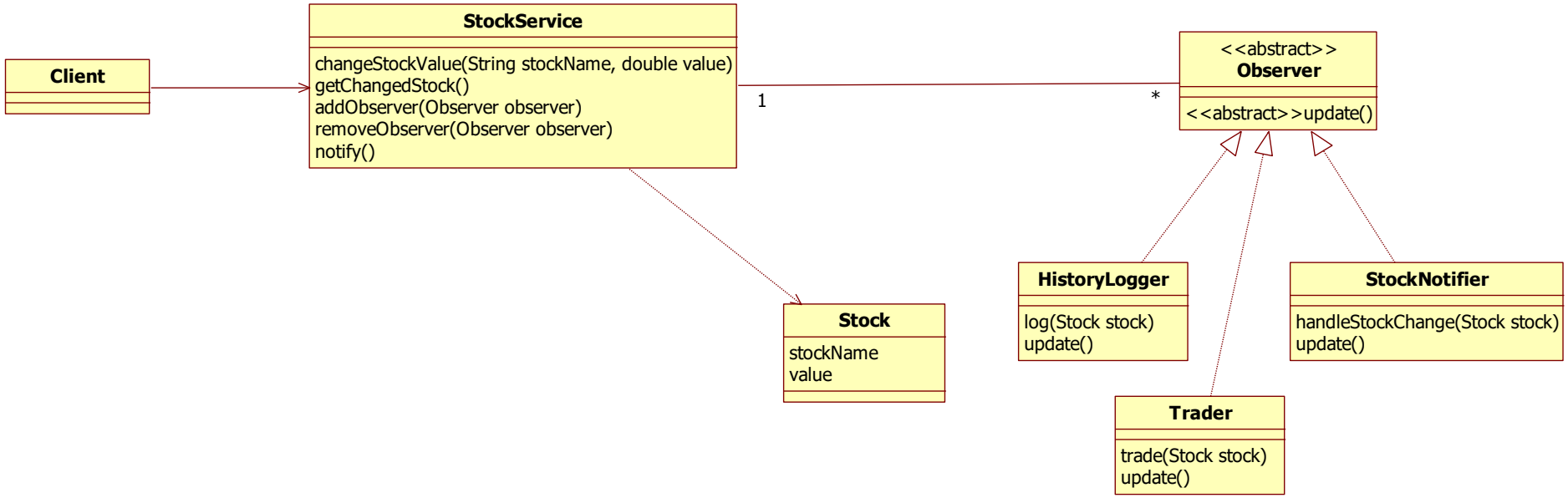
Application

```
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);
  }
}
```

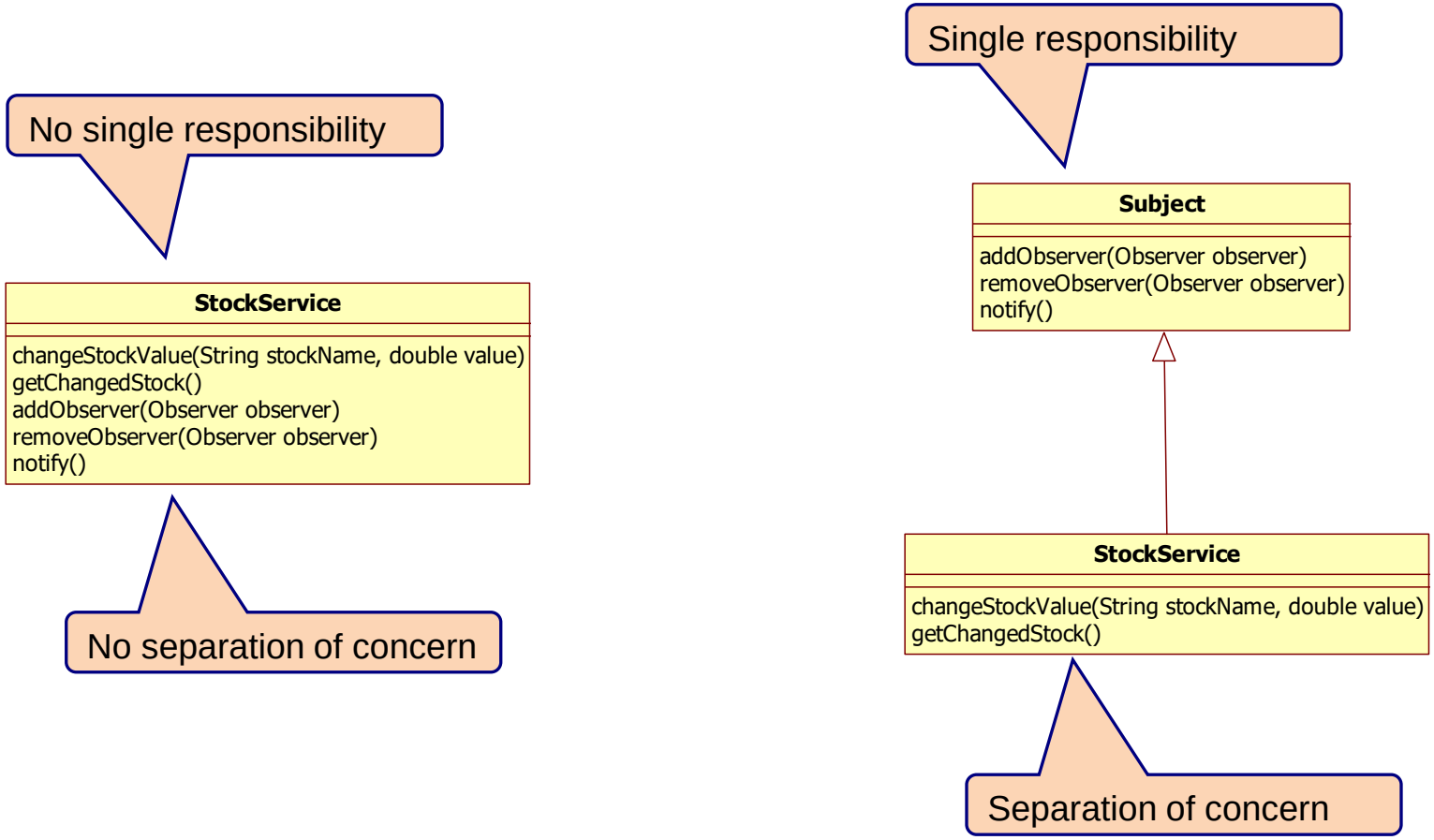
Advantage of Observer



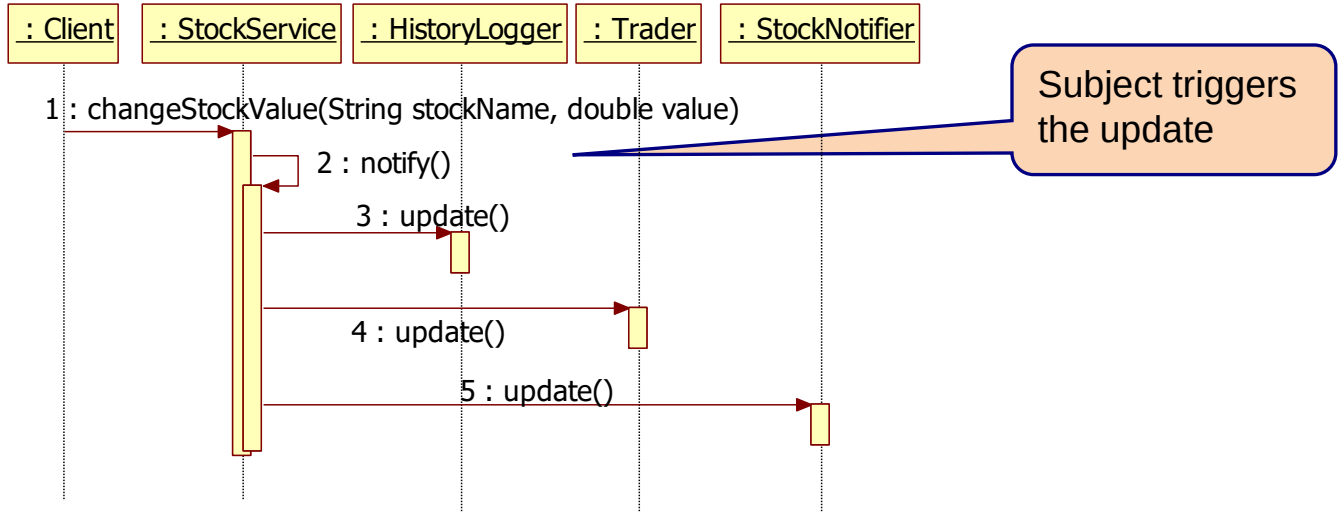
What is wrong with this?

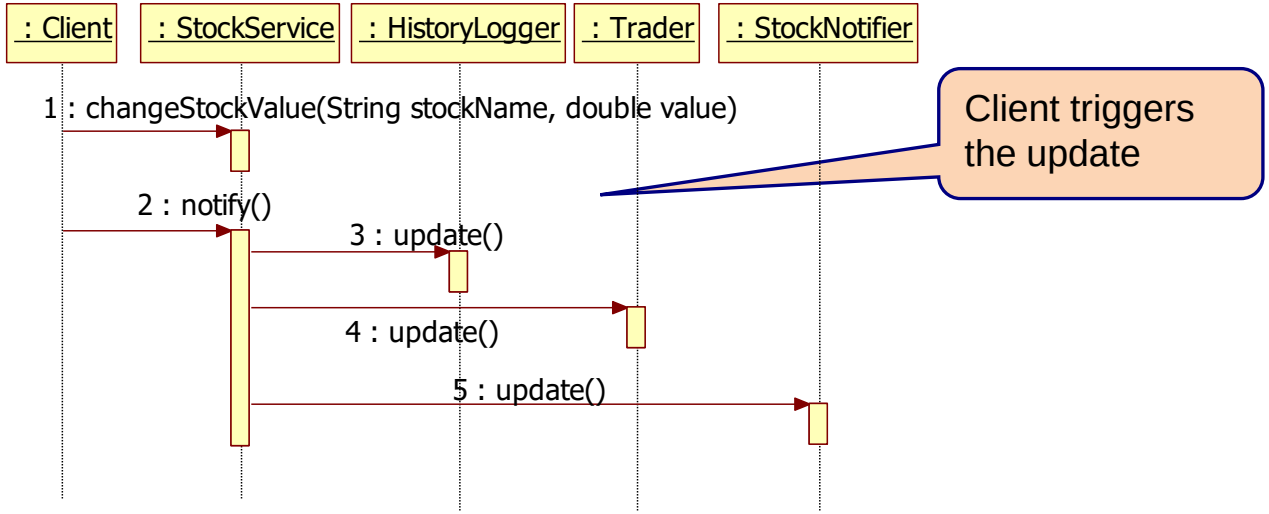


Separate Subject

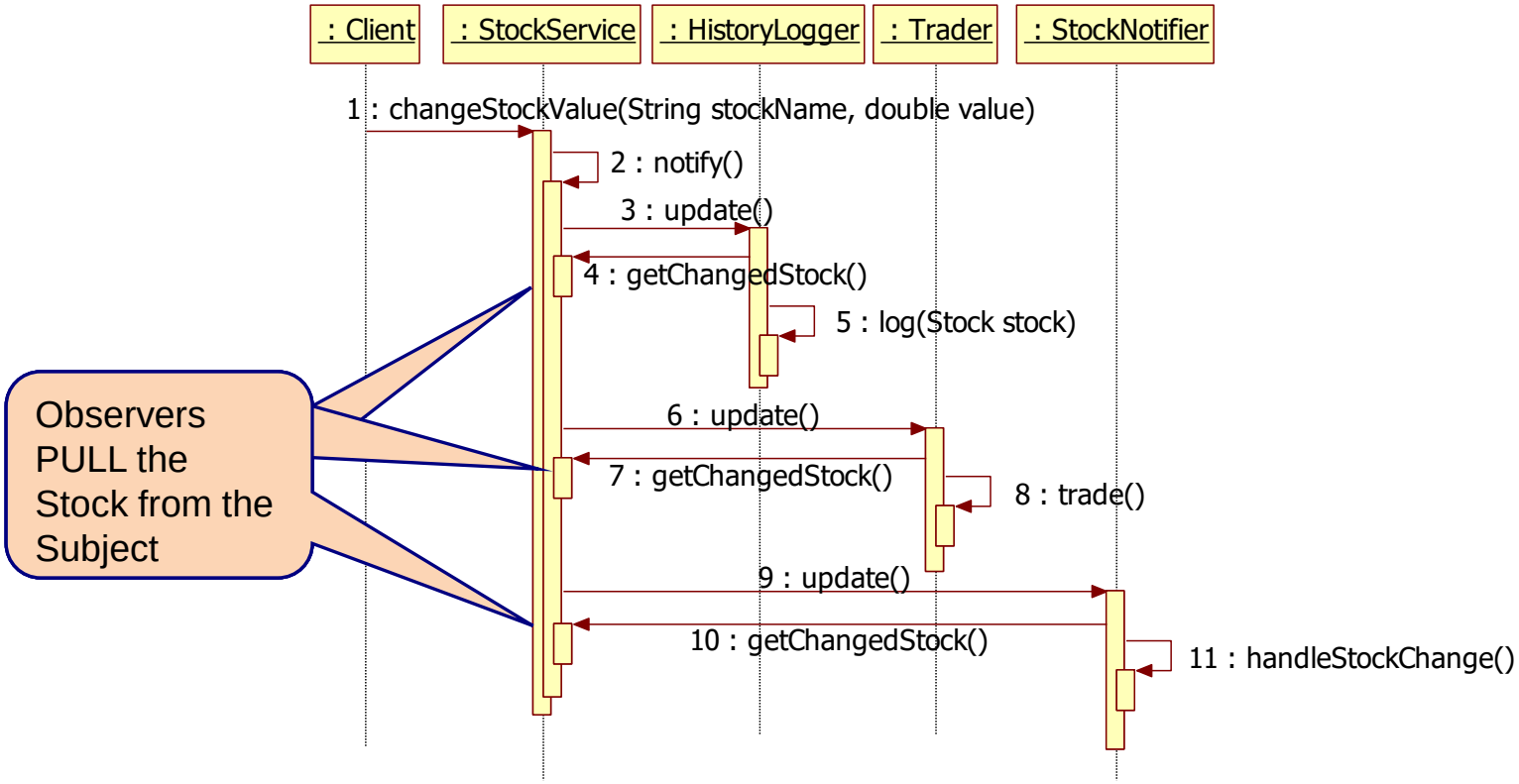


Who triggers the update?

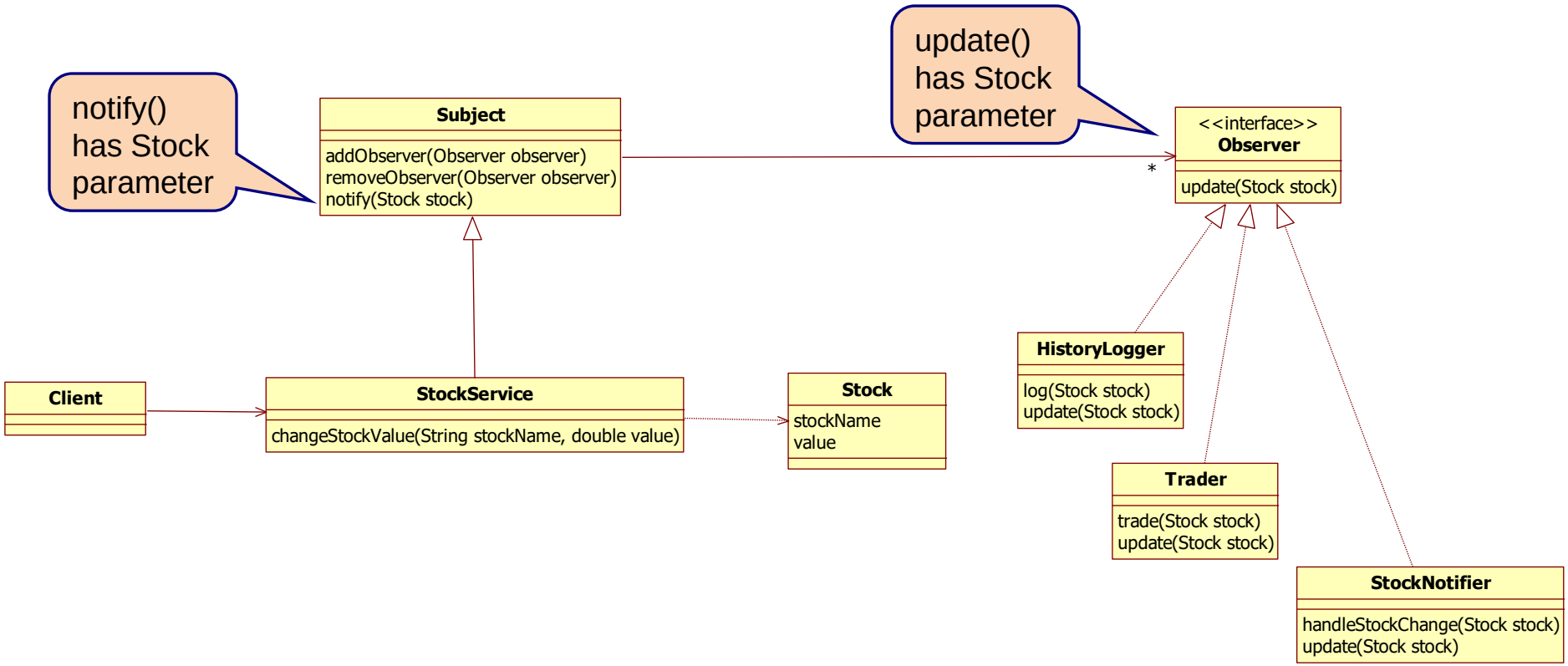




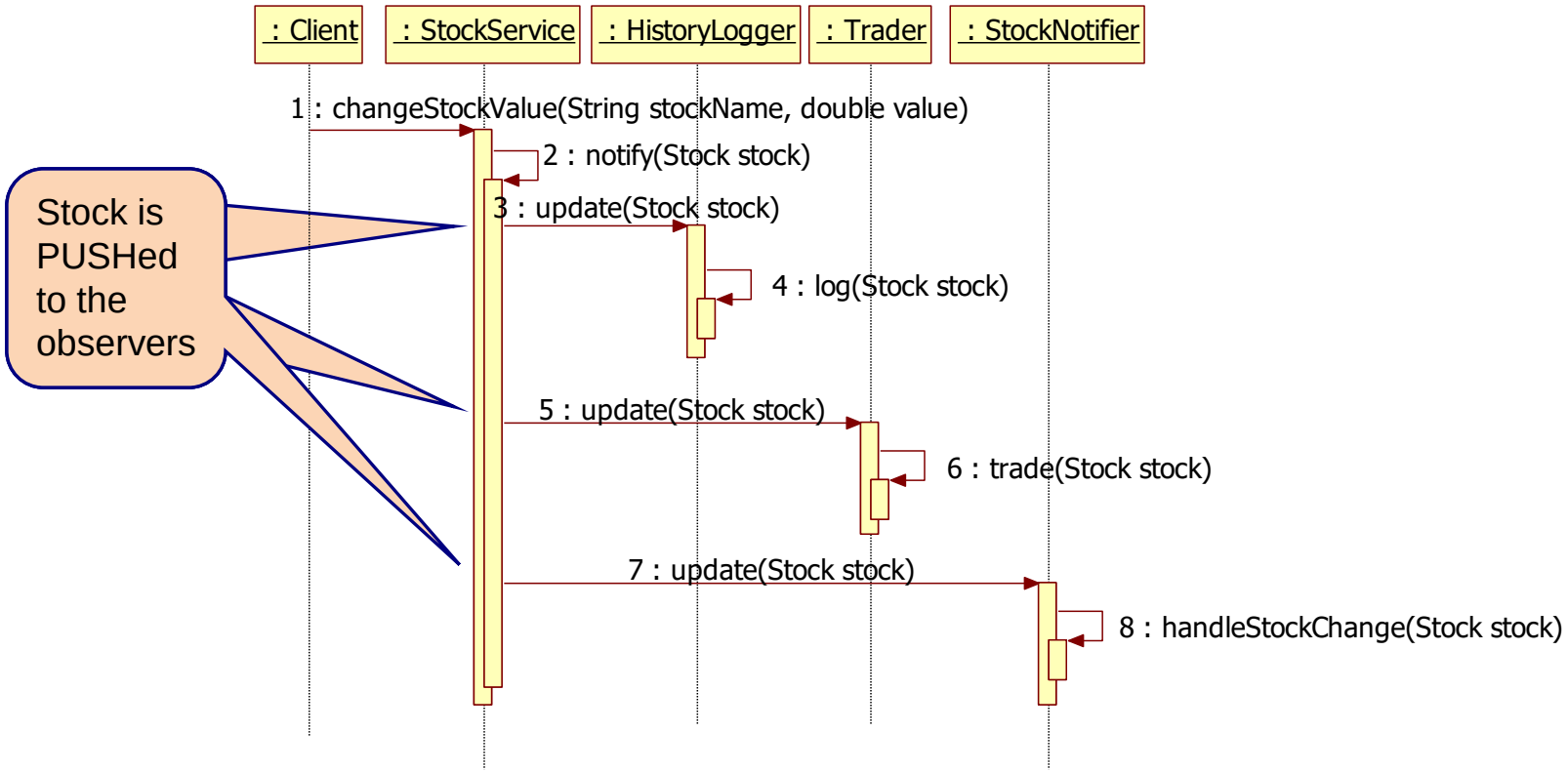
Pull model



Push model


notify()
has Stock
parameter
Subject
addObserver(Observer observer)
removeObserver(Observer observer)
notify(Stock stock)
update()
has Stock
parameter
<<interface>>
Observer
update(Stock stock)
*
Client
StockService
changeStockValue(String stockName, double value)
Stock
stockName
value
HistoryLogger
log(Stock stock)
update(Stock stock)
Trader
trade(Stock stock)
update(Stock stock)
StockNotifier
handleStockChange(Stock stock)
update(Stock stock)

Push model


: Client
: StockService
: HistoryLogger
: Trader
: StockNotifier
1 : changeStockValue(String stockName, double value)
2 : notify(Stock stock)
3 : update(Stock stock)
4 : log(Stock stock)
5 : update(Stock stock)
6 : trade(Stock stock)
7 : update(Stock stock)
8 : handleStockChange(Stock stock)
Stock is PUSHed to the observers

Subject, IObserver and Stock

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

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

  public void donotify(Stock stock){
    for (IObserver observer: observerlist){
      observer.update(stock);
    }
  }
}
```

```
public interface IObserver {
  public void update(Stock stock);
}
```

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

HistoryLogger

```
public class HistoryLogger implements IObserver {

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

  @Override
  public void update(Stock stock) {
    log(stock);
  }
}
```

Trader

```
public class Trader implements IObserver{

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

  @Override
  public void update(Stock stock) {
    trade(stock);
  }
}
```

StockNotifier

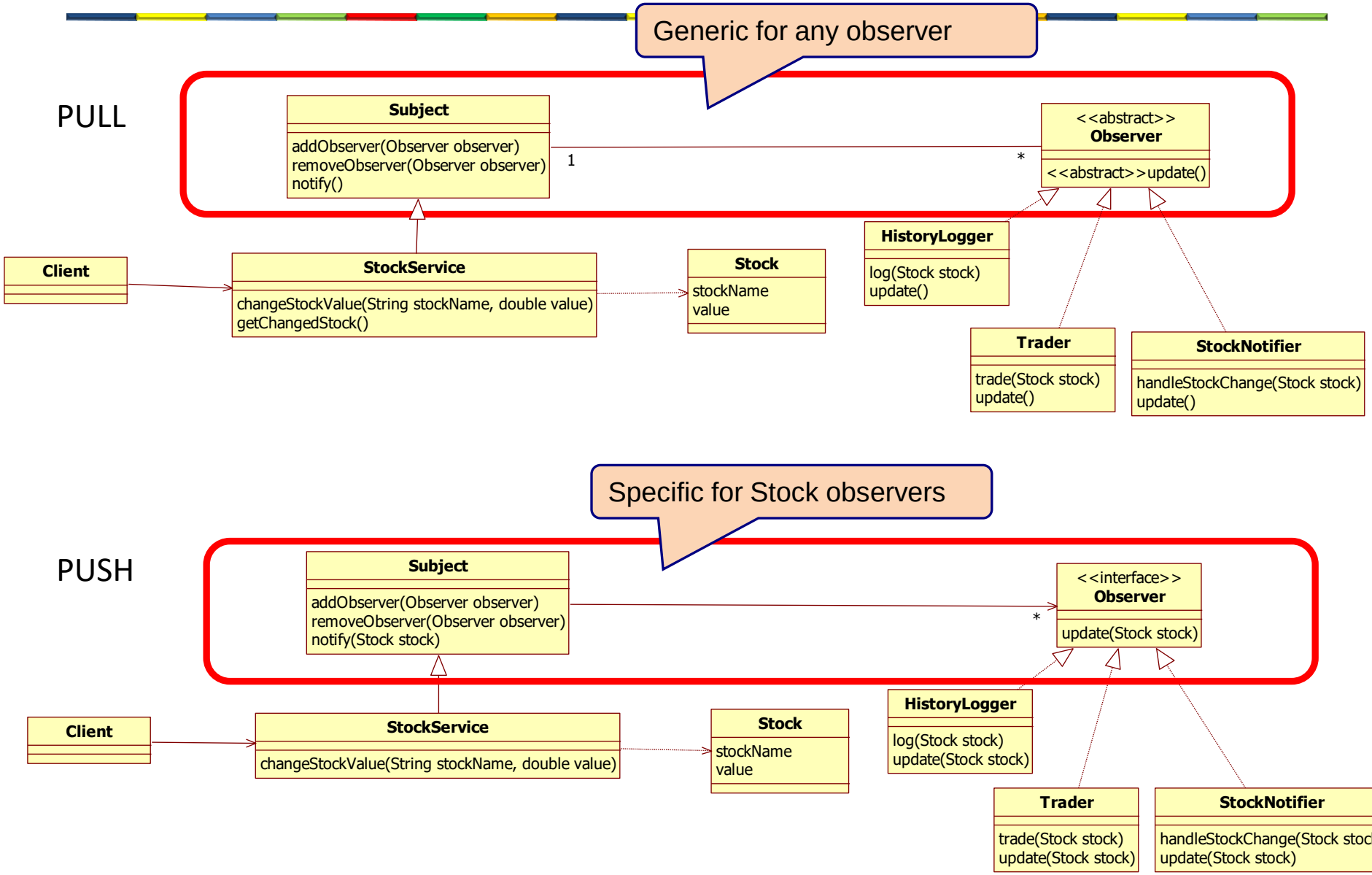
```
public class StockNotifier implements IObserver {

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

  @Override
  public void update(Stock stock) {
    handleStockChange(stock);
  }
}
```

StockService and Application

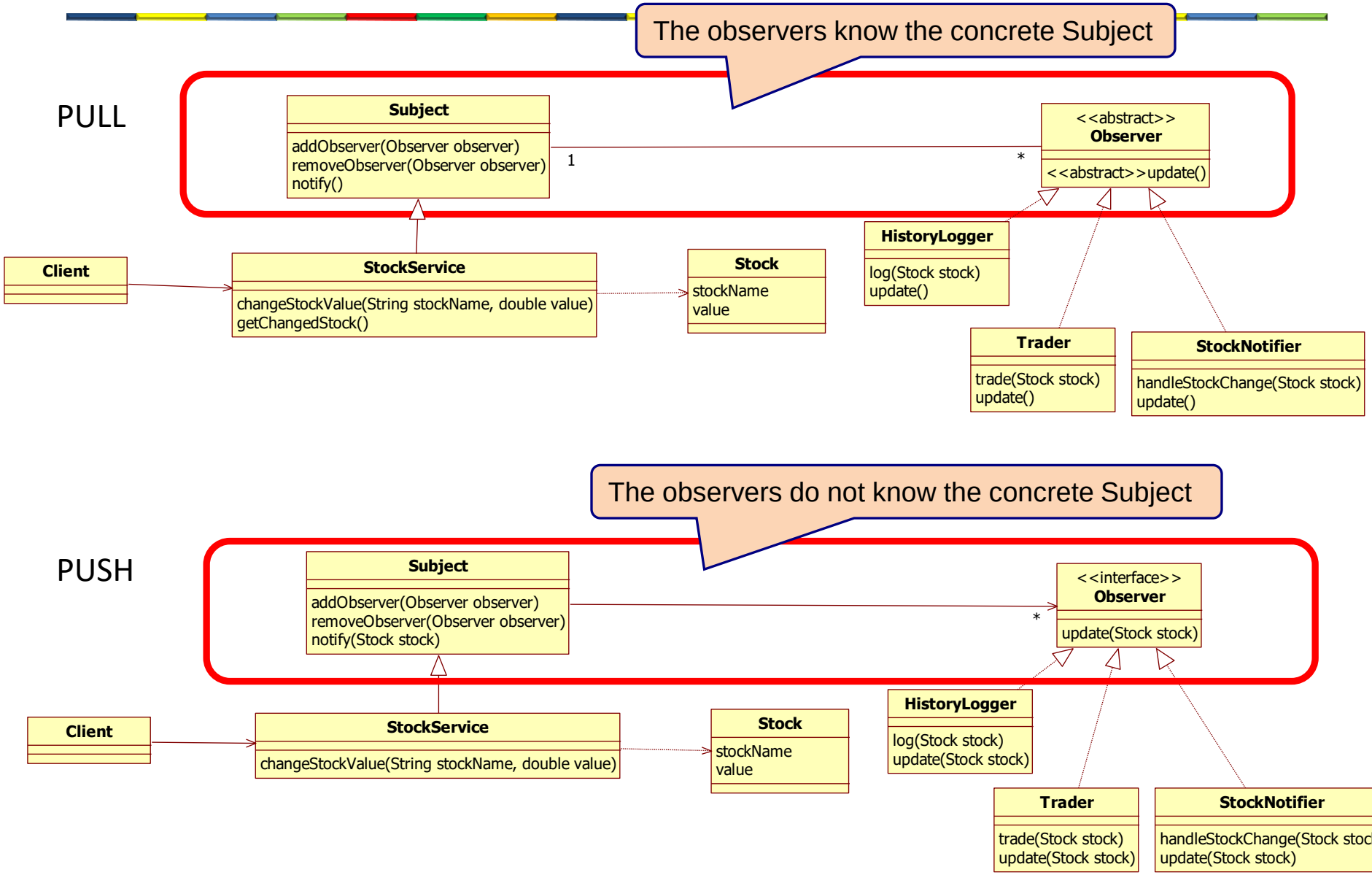
```
public class StockService extends Subject{

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

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

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

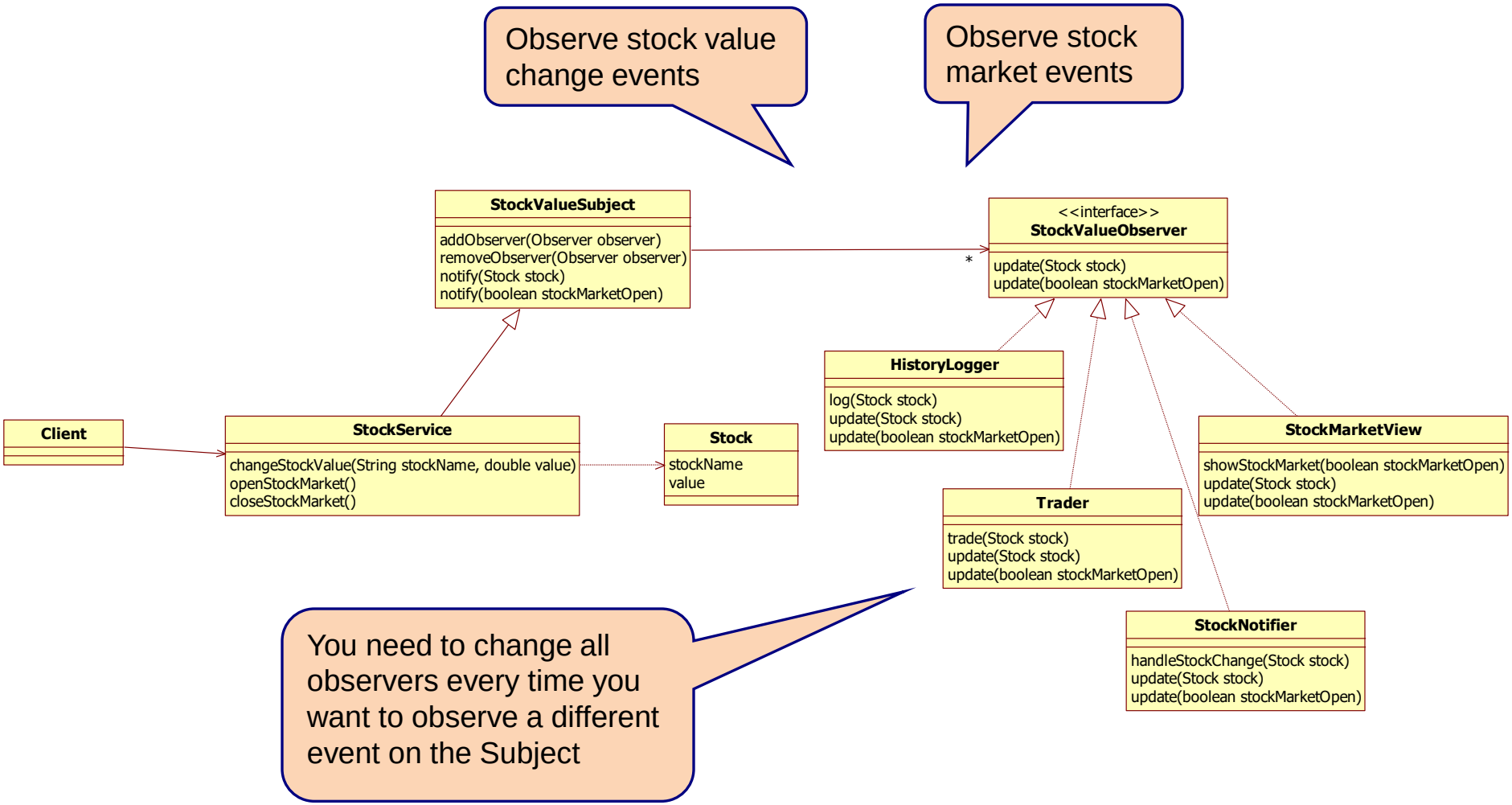
Difference push and pull



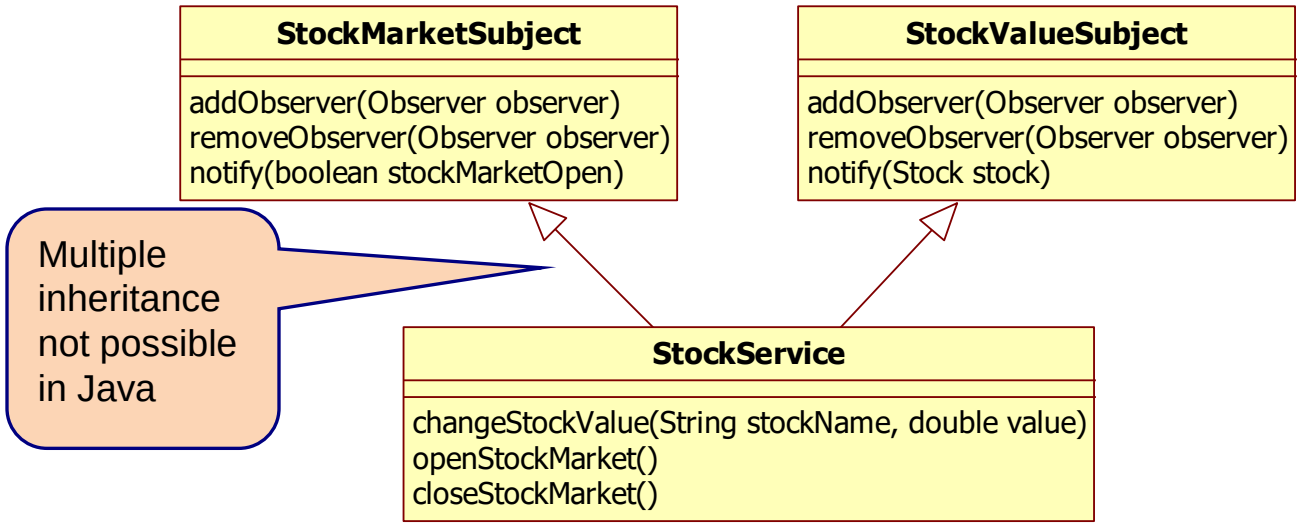
Difference push and pull



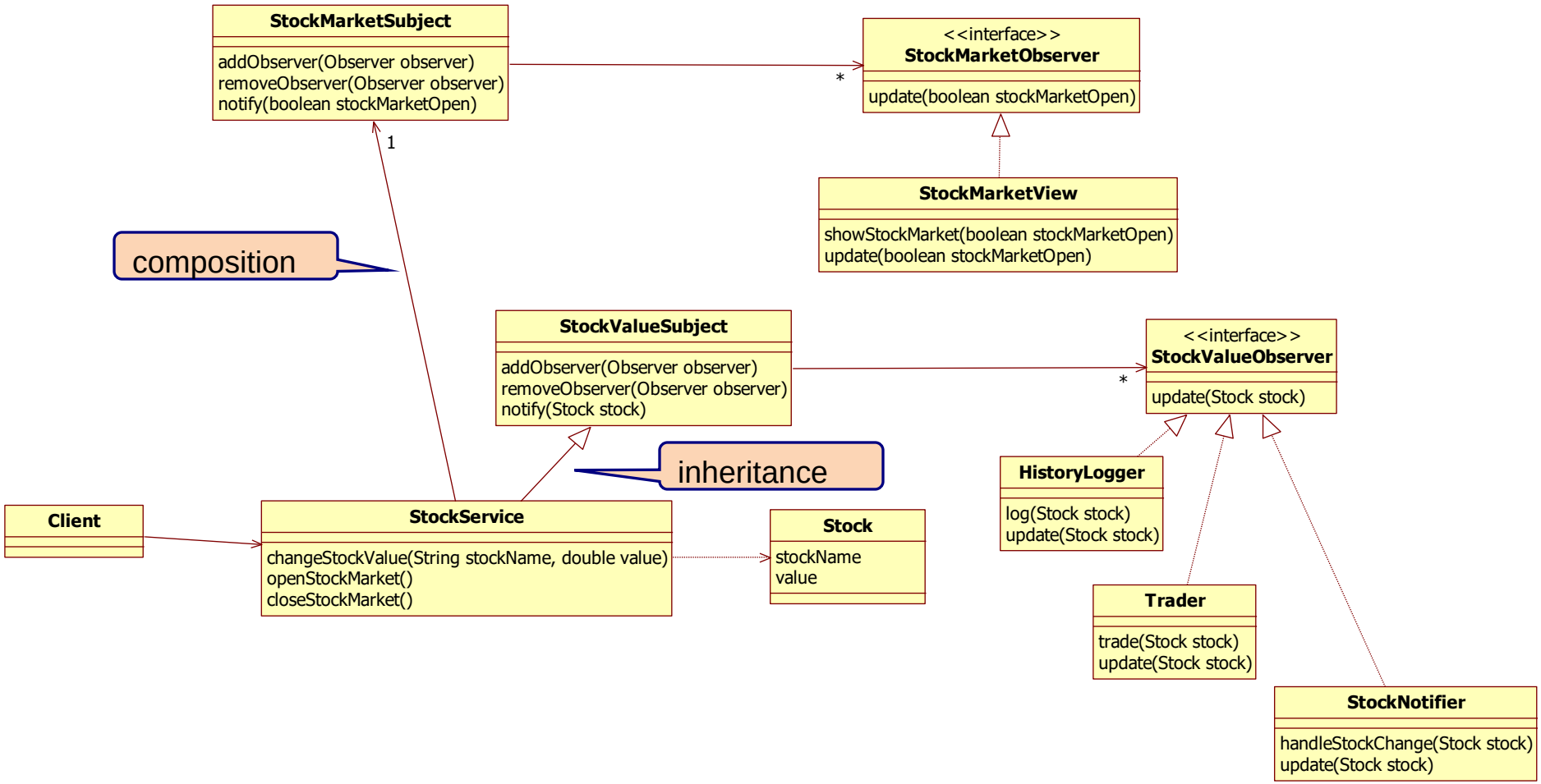
Observing multiple events



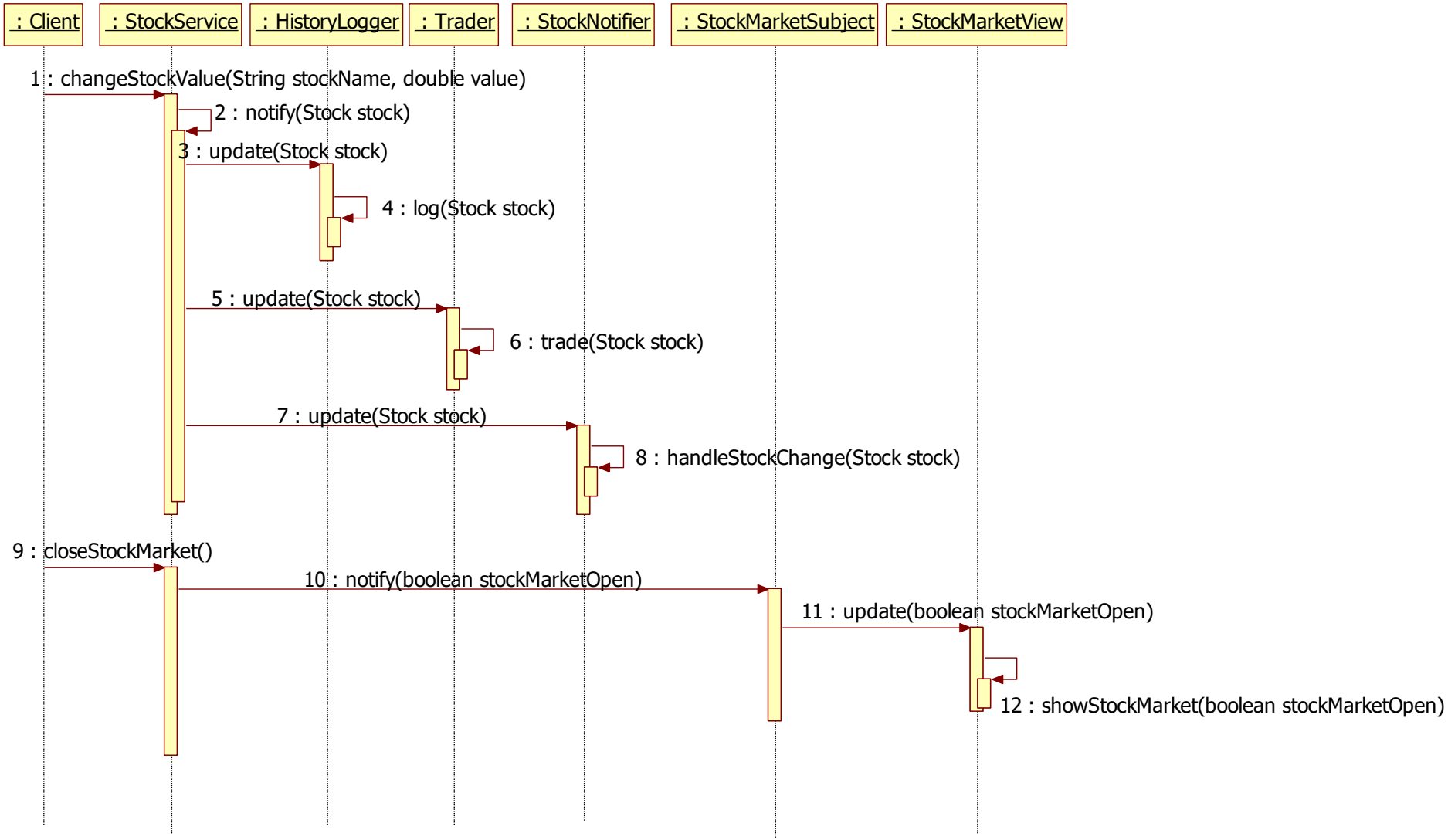
Multiple subjects



Multiple Subjects



Multiple Subjects



StockService

```
public class StockService extends StockValueSubject{
  private boolean stockMarketOpen=false;
  private StockMarketSubject stockMarketSubject;

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

  public void openStockMarket() {
    stockMarketOpen=true;
    stockMarketSubject.donotify(stockMarketOpen);
  }

  public void closeStockMarket() {
    stockMarketOpen=false;
    stockMarketSubject.donotify(stockMarketOpen);
  }

  public StockMarketSubject getStockMarketSubject() {
    return stockMarketSubject;
  }

  public void setStockMarketSubject(StockMarketSubject stockMarketSubject) {
    this.stockMarketSubject = stockMarketSubject;
  }
}
```

The Subjects and observer interfaces

```
public class StockValueSubject  {
  private Collection<StockValueObserver> observerlist = new ArrayList<StockValueObserver>();

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

  public void donotify(Stock stock){
    for (StockValueObserver observer: observerlist){
      observer.update(stock);
    }
  }
}
```

```
public interface StockValueObserver {
  public void update(Stock stock);
}
```

```
public class StockMarketSubject  {
  private Collection<StockMarketObserver> observerlist = new ArrayList<StockMarketObserver>();

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

  public void donotify(boolean stockMarketOpen){
    for (StockMarketObserver observer: observerlist){
      observer.update(stockMarketOpen);
    }
  }
}
```

```
public interface StockMarketObserver {
  public void update(boolean stockMarketOpen);
}
```

The concrete observers

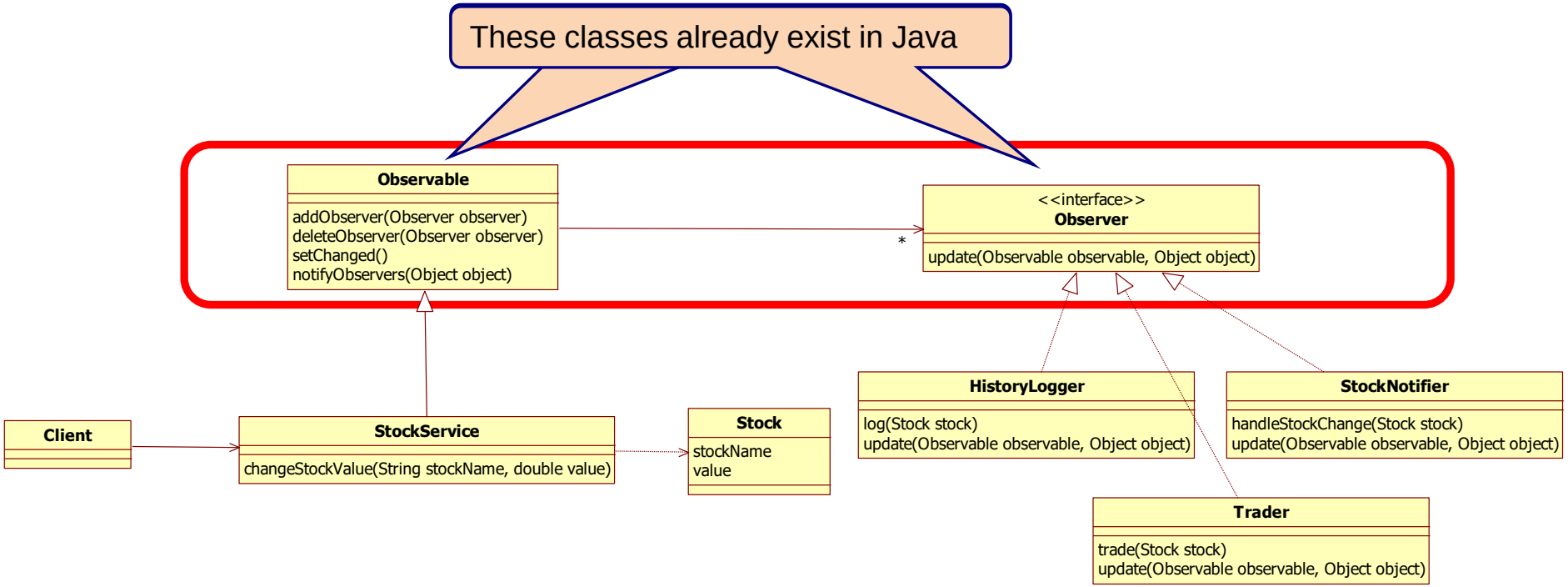
```
public class HistoryLogger implements IObserver {

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

  @Override
  public void update(Stock stock) {
    log(stock);
  }
}
```

```
public class StockMarketView implements StockMarketObserver{

  @Override
  public void update(boolean stockMarketOpen) {
    showStockMarket( stockMarketOpen);
  }

  public void showStockMarket(boolean stockMarketOpen) {
    if (stockMarketOpen) {
      System.out.println("The stock market is open");
    }
    else {
      System.out.println("The stock market is closed");
    }
  }
}
```

Application

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

    StockMarketSubject stockMarketSubject = new StockMarketSubject();
    StockMarketView stockMarketView = new StockMarketView();
    stockMarketSubject.addObserver(stockMarketView);
    stockService.setStockMarketSubject(stockMarketSubject);

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

Observer in Java



HistoryLogger and Trader

```java
import java.util.Observable;
import java.util.Observer;

public class HistoryLogger implements Observer {

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

  public void update(Observable observable, Object stock) {
    log((Stock) stock);
  }
}
```

```java
import java.util.Observable;
import java.util.Observer;

public class Trader implements Observer{

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


  public void update(Observable observable, Object stock) {
    trade((Stock) stock);
  }
}
```

StockNotifier and StockService

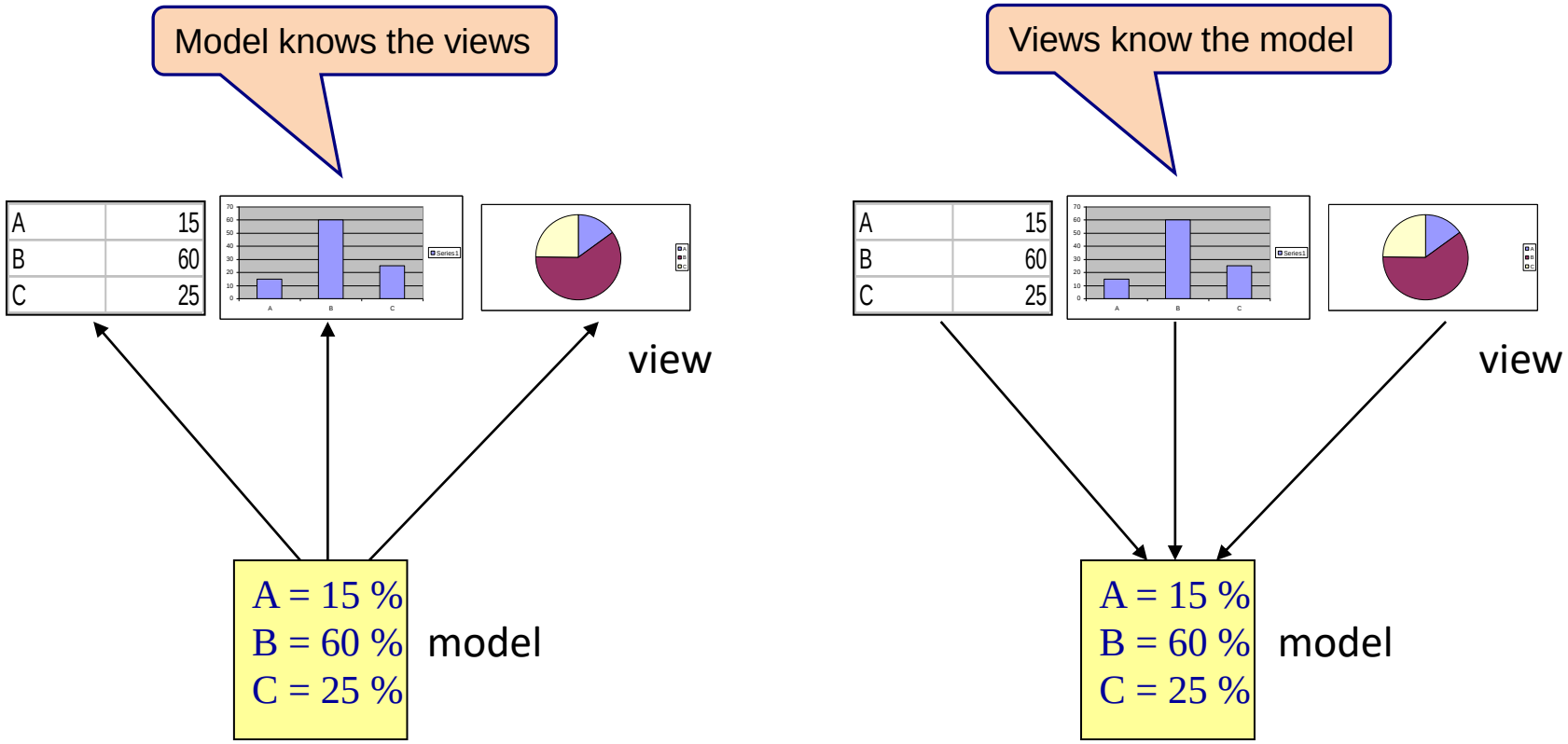
```
import java.util.Observable;
import java.util.Observer;

public class StockNotifier implements Observer {

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

  public void update(Observable observable, Object stock) {
    handleStockChange((Stock) stock);
  }
}
```

```
import java.util.Observable;

public class StockService extends Observable{

  public void changeStockValue(String stockName, double value) {
    Stock stock = new Stock(stockName, value);
    setChanged();
    notifyObservers(stock);
  }
}
```

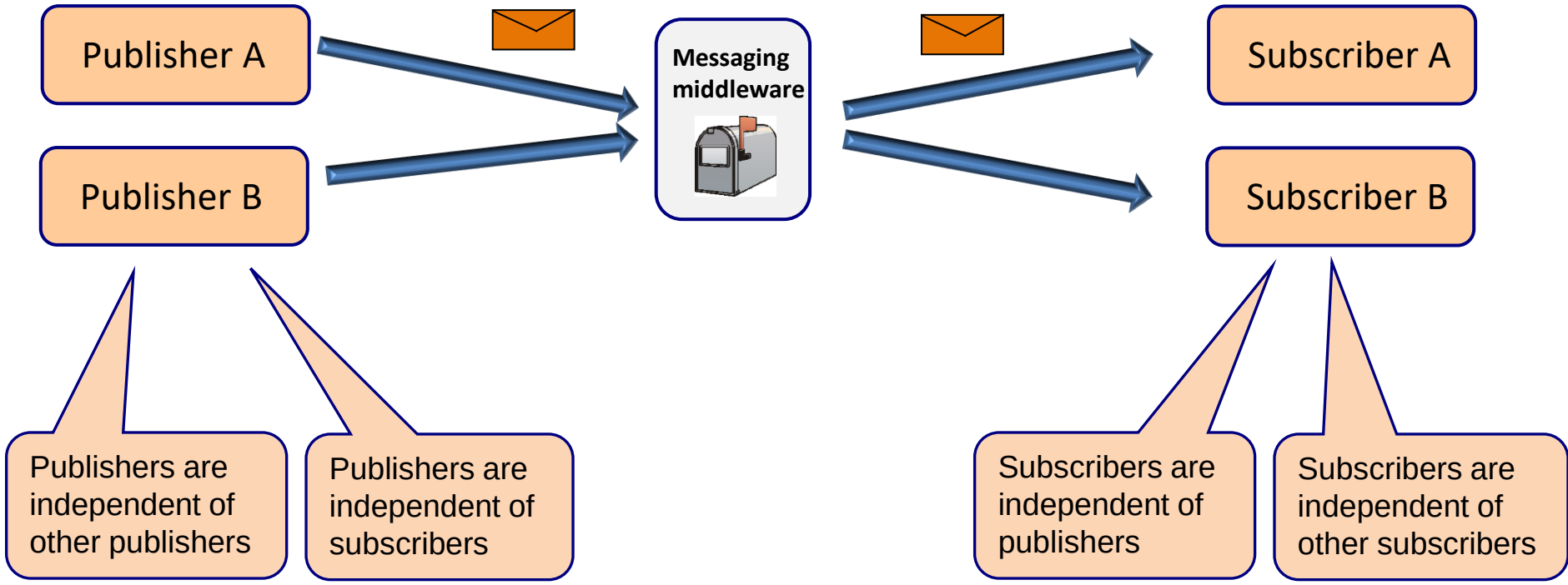
Application

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

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

Model View Controller (MVC)

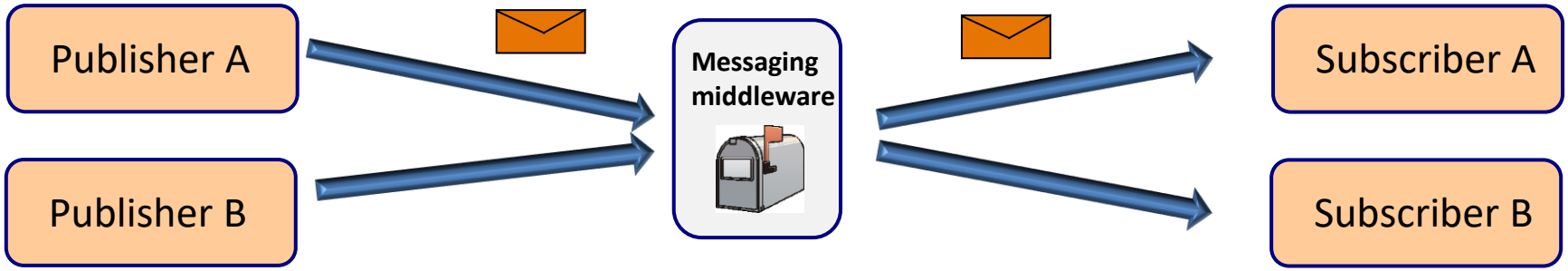


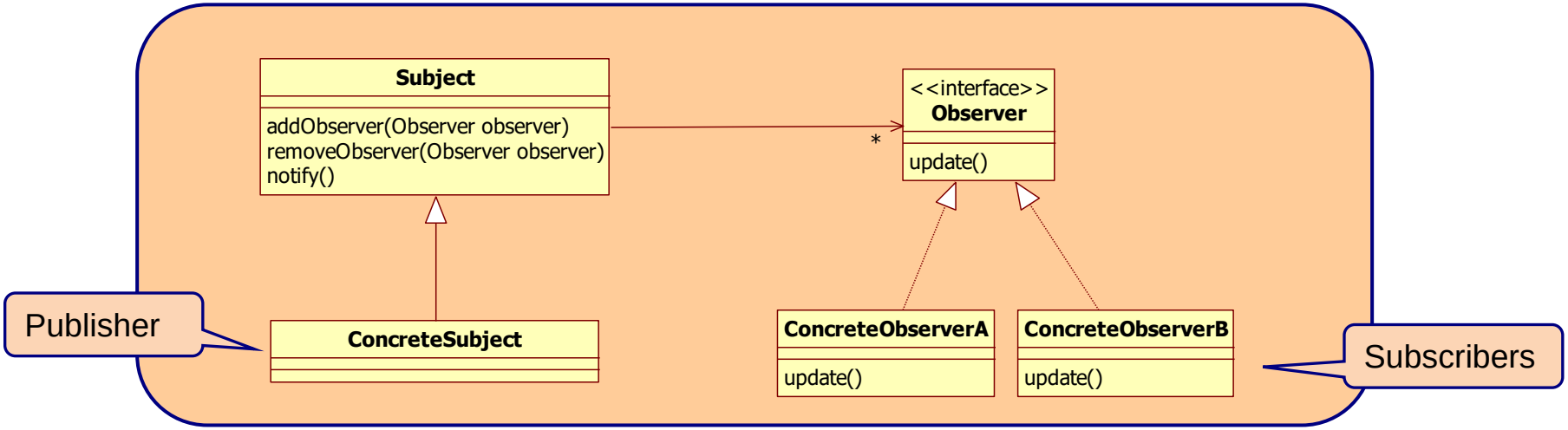
Publish-subscribe (pub-sub)



Loose coupling

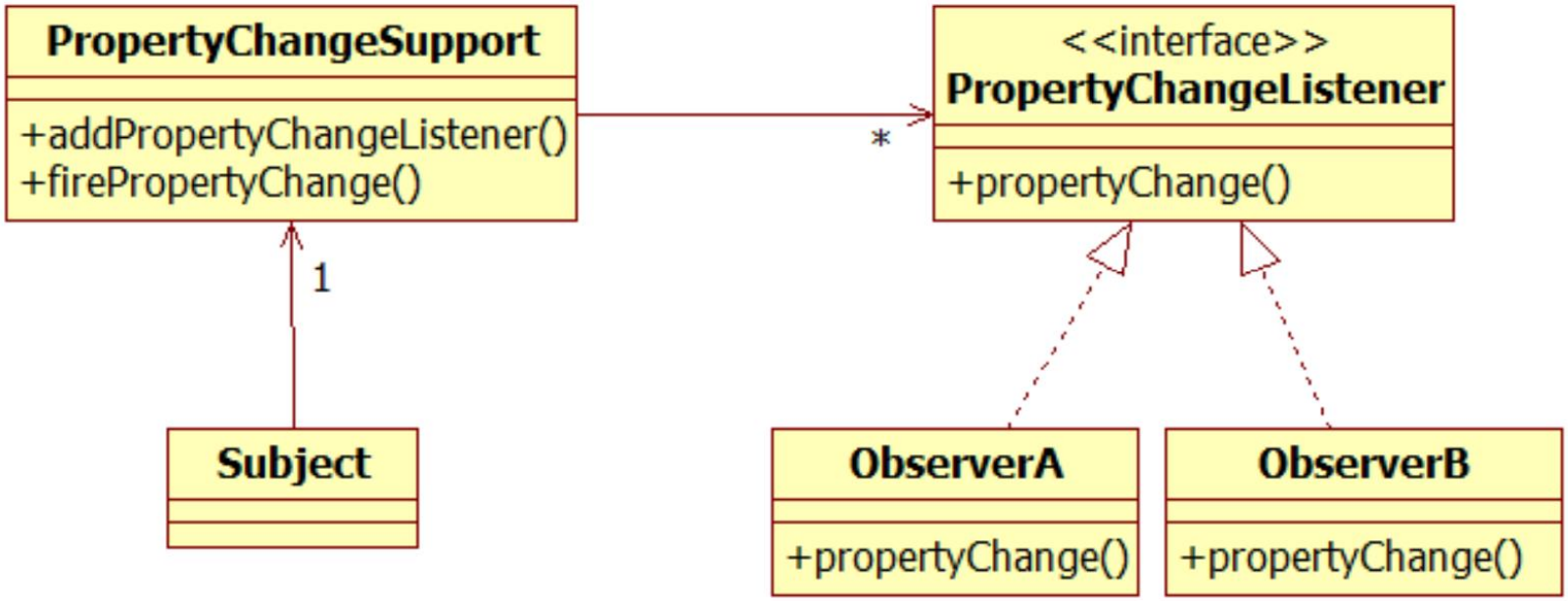
Publish-subscribe (pub-sub)





Loose coupling

PropertyChangeListener in Java



HistoryLogger and Trader

```
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;

public class HistoryLogger implements PropertyChangeListener {
   @Override
   public void propertyChange(PropertyChangeEvent evt) {
     log((Stock)evt.getNewValue());
   }

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

Subscribe to an event

```
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;

public class Trader implements PropertyChangeListener {
   @Override
   public void propertyChange(PropertyChangeEvent evt) {
     trade((Stock)evt.getNewValue());
   }

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

Subscribe to an event

StockNotifier

```
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;

public class StockNotifier implements PropertyChangeListener {
   @Override
   public void propertyChange(PropertyChangeEvent evt) {
     handleStockChange((Stock)evt.getNewValue());
   }

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

Subscribe to an event

StockService

```
import java.beans.PropertyChangeSupport;
import java.beans.PropertyChangeListener;

public class StockService {
   private PropertyChangeSupport support;

   public StockService() {
      support = new PropertyChangeSupport(this);
   }

   public void addPropertyChangeListener(PropertyChangeListener pcl) {
      support.addPropertyChangeListener(pcl);
   }

   public void changeStockValue(String stockName, double value) {
      Stock stock = new Stock(stockName, value);
      support.firePropertyChange("stock", stock, stock);
   }
}
```

Publish an event

Name of the event

Old value

New value

Application

```
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.addPropertyChangeListener(historyLogger);
     stockService.addPropertyChangeListener(trader);
     stockService.addPropertyChangeListener(stockNotifier);


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

   }
}
```

Observer pattern

- What problem does it solve?
  - When a change to one object requires changing others, and you don't know how many objects need to be changed.
  - When an object should be able to notify other objects without making assumptions about who these objects are. In other words, you don't want these objects tightly coupled.

Main point

- The observer pattern makes observables (publishers) independent of observers (subscribers)
- All human beings have the ability to observe the intelligence of nature