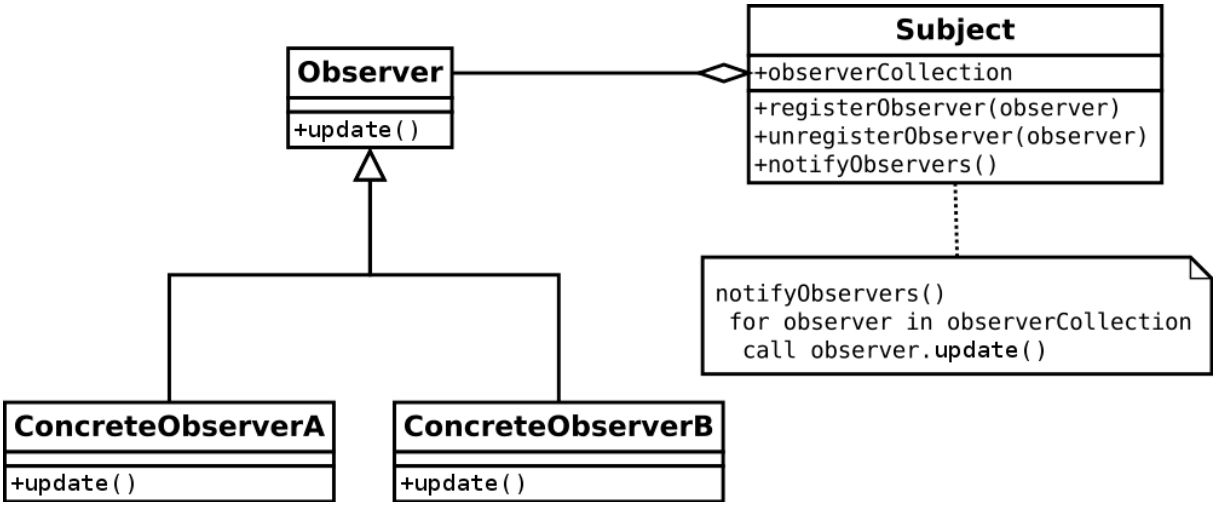
Merkblatt Observer / Observable

Zweck

Wenn Objekte informiert werden müssen, wenn sich ein bestimmtes Objekt verändert, können diese als Observer einfach von dem Subject über eine Veränderung informiert werden.

UML



Problem

Wenn sich der Preis einer Aktie verändert, dann müssen alle Börsen darüber informiert werden. In diesem Fall wäre eine Aktie ein Subject, welches die Börse, also den Observer über die Veränderung informiert.

Lösung

Die einzelnen Aktien werden als Subjects implementiert und informieren die Börsen, welche hier die Observer sind, über Veränderungen im Preis.

Vorteile

Es entsteht eine lose Kopplung, durch die die Objekte zu einem gewissen Grad kommunizieren können, sie müssen aber nichts übereinander wissen. Jeder Observer kann bei jedem Subject für Updates angemeldet werden.

Code

```
------------ Main : GrabStocks.java ------------
package ooPattern;

/**
 * Dies ist das Main.
 *
 * Verwendete Klassen:
 * Observer : Ein Interface für Klassen, welche Observer /
Beobachter sein sollen
```

```
 * StockObserver : Implementiert Observer, beobachtet die
Preise und erhält Nachricht von StockGrabber, wenn es in einem
Subject ein Update gab
 * Subject : Ein Interface für Klassen, welche beobachtet
werden sollen
 * StockGrabber : Implementiert Subject, wird von
StockObserver beobachtet/benachrichtigt diesen, wenn es ein
Update gab
 * GetTheStock : Verändert die Preise, damit wir sehen können,
wie Observer und Subject interagieren
 *
 */

public class GrabStocks{

public static void main(String[] args){

    // Das erste Subject wird generiert
    StockGrabber stockGrabber = new StockGrabber();

    // Es werden zwei Observer generiert, welche auf Updates
vom Subject warten
    // Es werden zudem die Start-Preise für die Aktien
festgelegt
    StockObserver observer1 = new StockObserver(stockGrabber);
    stockGrabber.setIBMPrice(197.00);
    stockGrabber.setAAPLPrice(677.60);
    stockGrabber.setGOOGPrice(676.40);
    StockObserver observer2 = new StockObserver(stockGrabber);
    stockGrabber.setIBMPrice(197.00);
    stockGrabber.setAAPLPrice(677.60);
    stockGrabber.setGOOGPrice(676.40);

    // Einer der Observer wird entfernt
    stockGrabber.unregister(observer2);

    // Es werden 3 threads erzeugt, mit denen die Preise
automatisch verändert werden
    // Das hat keinen Zusammenhang mit den OO-Pattern, sondern
dient nur der Veranschaulichung der Interaktion von Observer
und Subject
    Runnable getIBM = new GetTheStock(stockGrabber, 2, "IBM",
197.00);
    Runnable getAAPL = new GetTheStock(stockGrabber, 2,
"AAPL", 677.60);
    Runnable getGOOG = new GetTheStock(stockGrabber, 2,
"GOOG", 676.40);
    // Die automatische veränderung der Preise wird gestartet
    new Thread(getIBM).start();
    new Thread(getAAPL).start();
    new Thread(getGOOG).start();
    }
```

```
}

------------ Observer Interface : Observer.java ------------
package ooPattern;

// In diesem Interface wird nur die update Methode vorgegeben,
welche ein Subject aufruft, wenn es ein Update gab
public interface Observer {

     public void update(double ibmPrice, double aaplPrice,
double googPrice);

}

------------ Observer Klasse : StockObserver.java ------------
package ooPattern;

public class StockObserver implements Observer {

     private double ibmPrice;
     private double aaplPrice;
     private double googPrice;

     // Ein Counter, der als ID für die Observer verwendet
wird
     private static int observerIDTracker = 0;
     // Hier wird die ID gespeichert
     private int observerID;

     // Das Subject, welches beobachtet werden soll wird hier
gespeichert
     private Subject stockGrabber;

     public StockObserver(Subject stockGrabber){
          // Hier wird das Subject zugewiesen
          this.stockGrabber = stockGrabber;
          // Hier wird die ID zugewiesen
          this.observerID = ++observerIDTracker;
          // Damit wir wissen, wenn ein neuer Observer
implementiert wurde erfolgt eine Ausgabe mit ID
          System.out.println("New Observer " +
this.observerID);
          // Damit auch das Subject weiss, dass es diesen
Observer benachrichtigen muss
          // wird der Observer in eine ArrayList im Subject
hinzugefügt
          stockGrabber.register(this);
     }

     // Mit dieser Methode werden die Observer aktualisiert
```

```
     public void update(double ibmPrice, double aaplPrice,
double googPrice) {
          this.ibmPrice = ibmPrice;
          this.aaplPrice = aaplPrice;
          this.googPrice = googPrice;
          printThePrices();
     }

     // Es erfolgt eine Ausgabe des Observers und der
aktuellen Preise
     public void printThePrices(){
          System.out.println(observerID + "\nIBM: " + ibmPrice
+ "\nAAPL: " +
          aaplPrice + "\nGOOG: " + googPrice + "\n");
     }

}

------------ Subject Interface : Subject.java ------------
package ooPattern;

// In diesem Interface für Subjects werden die Methoden zum
Hinzufügen, Entfernen und Updaten von Observern vorgegeben
public interface Subject {

     public void register(Observer o);
     public void unregister(Observer o);
     public void notifyObserver();

}

------------ Subject Klasse : StockGrabber.java ------------
package ooPattern;

import java.util.ArrayList;

public class StockGrabber implements Subject{

    private ArrayList<Observer> observers;
    private double ibmPrice;
    private double aaplPrice;
    private double googPrice;

    public StockGrabber(){
        // In der ArrayList eines Subjects werden die Observer
hinzugefügt, welche benachrichtigt werden müssen
        observers = new ArrayList<Observer>();
    }

    public void register(Observer newObserver) {
        // Hier kann ein neuer Observer zur ArrayList
hinzugefügt werden
```

```
        observers.add(newObserver);
    }

    public void unregister(Observer deleteObserver) {

         // Der Index des zu entfernenden Observers wird
ermittelt
         int observerIndex = observers.indexOf(deleteObserver);

         // Es wird ausgegeben, welcher Observer entfernt wurde
        System.out.println("Observer " + (observerIndex+1) + "
deleted");

        // Der Observer wird entfernt
        observers.remove(deleteObserver);
     }

    public void notifyObserver() {
        // Diese Methode läuft durch alle Observer und
benachrichtigt diese bei Updates
        for(Observer observer : observers){
        observer.update(ibmPrice, aaplPrice, googPrice);
        }
    }

    // Mit diesen Methoden können die Preise neu gesetzt
werden
    // In dem Fall werden die Observer auch gleich
benachrichtigt
    public void setIBMPrice(double newIBMPrice){
        this.ibmPrice = newIBMPrice;
        notifyObserver();
    }

    public void setAAPLPrice(double newAAPLPrice){
        this.aaplPrice = newAAPLPrice;
        notifyObserver();
    }

    public void setGOOGPrice(double newGOOGPrice){
        this.googPrice = newGOOGPrice;
        notifyObserver();
    }

}

------- passt Preise automatich an :GetTheStocks.java --------
package ooPattern;
import java.text.DecimalFormat;

public class GetTheStock implements Runnable{
```

```
    private String stock;
    private double price;
    // Hier wird gespeichert, welches Subject verändert werden
soll
    private Subject stockGrabber;

    public GetTheStock(Subject stockGrabber, int newStartTime,
String newStock, double newPrice){
        // Hier wird das Subject zugewiesen
        this.stockGrabber = stockGrabber;
        // Bezeichnung und Preis für eine Aktie werden
festgelegt
        stock = newStock;
        price = newPrice;
    }

    public void run(){
        for(int i = 1; i <= 20; i++){
            try{
                // Wartezeit bavor die Preise erneut
aktualisiert werden
                Thread.sleep(3500);
            }
            catch(InterruptedException e)
            {}

            // Generiert eine zufällige Zahl, um die der Preis
angepasst wird
            // Das hat keinen Zusammenhang mit den OO-Pattern,
            //sondern dient nur der Veranschaulichung der
Interaktion von Observer und Subject
            // durch die automatische Preisveränderung
            double randNum = (Math.random() * (.06)) - .03;
            DecimalFormat df = new DecimalFormat("#.##");
            price = Double.valueOf(df.format((price +
randNum)));

            if(stock == "IBM") ((StockGrabber)
stockGrabber).setIBMPrice(price);
            if(stock == "AAPL") ((StockGrabber)
stockGrabber).setAAPLPrice(price);
            if(stock == "GOOG") ((StockGrabber)
stockGrabber).setGOOGPrice(price);

            System.out.println(stock + ": " + df.format((price
+ randNum)) +
            " " + df.format(randNum));
            System.out.println();
        }
    }
```

}