Übungsbeispiel 2 für die LVA „Objektorientiertes Programmieren“

Deadline: 27.04.2014!

WICHTIG: Alle Studierenden, die ihr Beispiel nicht bis zum 27.04.2014, 24 Uhr am Institutsserver hochgeladen haben, können ihr Beispiel nicht abgeben. Diese Studierenden haben in Folge nicht mehr die Möglichkeit, die Übung der LVA im laufenden Semester erfolgreich abzuschließen. Damit haben sie auch keine Möglichkeit, zur LVA-Prüfung anzutreten und diese im SS 2014 abzuschließen!

Abgabe:

Die Abgabegespräche für das erste Beispiel finden am *28.04.2014* und am *29.04.2014* statt (jeweils zwischen 13:00 und 17:00 Uhr: Den genauen Zeitpunkt können sie beim Upload auswählen).

Allgemeine Übungsinformation

Im Laufe des Übungsteiles der LVA 384.061 „Objektorientiertes Programmieren“ muss jeder Teilnehmer drei Beispiele ausarbeiten und abgeben. **JEDES dieser drei Beispiele muss positiv bewertet werden, um die Übung zu bestehen** und damit die Möglichkeit zu erhalten, zur Abschlussprüfung anzutreten. Genauere Informationen zum Bewertungsschema finden Sie in den „Folien zur Übungseinweisung“ auf unserer Webseite. Weitere Informationen zu Vorlesung und Übung finden sie im Web unter:

http://www.ict.tuwien.ac.at/lva/384.061/

Jeder Teilnehmer muss die Beispiele **eigenständig** ausarbeiten. Bei der Abgabe jedes Beispiels gibt es ein kurzes Abgabegespräch mit einem Tutor oder Assistenten. Diese Abgabegespräche bestehen aus

- einem Test des Programms und Fragen dazu,
- der Durchführung kleiner Änderungen am Programm,
- und theoretischen Fragen zu den jeweils in der Übung behandelten Konzepten.

Um ein Beispiel abzugeben müssen Sie ihre Quellcode-Dateien in einem .zip-Archiv zusammenfassen und dieses über unser Upload Interface auf unseren Institutsserver hochladen. Bitte verwenden sie **ausschließlich .zip-Dateien** (z.B., bsp1.**zip**) und **keine** anderen Archivformate! Der Link zum Upload Interface ist

http://www.ict.tuwien.ac.at/lva/384.061/?ue_upload=

Verwenden Sie keine packages um ihren Code zu strukturieren. Zum Zeitpunkt, wenn Sie ihr Beispiel hochladen, **muss ihr Code auf der Kommandozeile kompilieren** (d.h., ohne IDE). **Kommentieren** Sie Ihre Klassen und Methoden. Vergessen Sie insbesondere nicht, ihren Namen und ihre Matrikelnummer am Beginn jeder Klasse anzuführen. Halten Sie sich an die Java Code Conventions, http://www.oracle.com/technetwork/java/codeconv-138413.html.

Die Software für die LVA ist die Java Standard Edition (JDK Version 1.7). Diese finden sie unter

http://www.oracle.com/technetwork/java/javase/downloads/index.html.

Inhaltliche Fragen zu den Beispielen können Sie im OOP-Forum im TISS (http://tiss.tuwien.ac.at) posten. Hier erhalten Sie am schnellsten Hilfe, da ihre Fragen durch andere Studierende, Tutoren oder Assistenten beantwortet werden können.

Organisatorische Fragen zur LVA können Sie direkt an die Assistenten unter oop@ict.tuwien.ac.at richten.

Als Unterstützung für die Programmierung der Beispiele empfehlen wir das Java Tutorial von Oracle:

http://docs.oracle.com/javase/tutorial/

Aufgabe: Implementierung einer Börsensimulation unter Verwendung des Observer-Patterns

Ziel:

Praktische Implementierung des Observer Patterns, Java Interfaces und Polymorphismus.

Im Rahmen dieses Beispiels soll (stark vereinfacht) eine Börse simuliert werden.

Wichtige Konzepte:

Aktie: Eine Aktie (`Stock`) wird durch den Namen, ein Kürzel und ihren aktuellen Wert beschrieben. Eine Aktie kann einem Aktienindex oder mehreren Aktienindizes (siehe unten) angehören.
Bsp:
Name: Erste Bank
Kürzel: EBS
Wert: €33,50

Aktienindex: Ein Index (`StockIndex`) ist eine repräsentative Mischung von Aktien, die den Kursverlauf einer bestimmten Börse oder Branche widerspiegelt. Der ATX (Austrian Traded Index) enthält beispielsweise die wichtigsten an der Wiener Börse gehandelten Aktien. Ein Index enthält daher eine gewisse Anzahl von Aktien, die unterschiedlich gewichtet sind und wird durch seinen Namen charakterisiert. Ihr System soll zumindest den Index „ATX Five“ enthalten:

Beispiel:ATX Five:

Aktienname:	**Gewichtung**
Bank Austria Creditanstalt:	11,5405%
Erste Bank:	29,4574%
OMV:	14,9603%
Telekom Austria:	29,9157%
Wienerberger:	14,1262%

Um den Wert des Indexes zu berechnen, verwenden Sie folgende vereinfachte Formel:

$$Indexstand = \sum (Aktienwert * Gewichtung)$$

Um den Aktienindex im Programm zu repräsentieren, sollen Sie eine Klasse `StockIndex` mit entsprechenden Attributen erzeugen.

Um einen Aktienindexeintrag darzustellen bzw. die Verbindung zwischen einer Aktie und ihrer Gewichtung in einem Aktienindex herzustellen, erstellen Sie eine eigene Klasse `StockIndexEntry`, die beim Aktienindex die Gewichtung der Aktie mit dem Wert dieser Aktie verbindet. Ein Objekt dieser Klasse beinhaltet somit eine Aktie und die Gewichtung dieser Aktie. Ein Aktienindex hat eine Liste der Aktienindexeinträge zu verwalten. Der Indexstand eines Aktienindexes wird auf folgende Weise berechnet:

Für jeden Aktienindexeintrag im Aktienindex:
Indexstand = Indexstand + (Wert der Aktie * Gewichtung der Aktie)

Portfolio: Ein Portfolio (`Portfolio`) enthält Aktien in einer gewissen Stückzahl und wird durch einen Portfolio-Namen eindeutig identifiziert. Ein Portfolio wird normalerweise von einem Kunden zusammengestellt.

Beispiel:
Name: „Mein Portfolio“
Aktien:
- Erste Bank, 100 Stück
- OMV, 200 Stück
- Wienerberger, 250 Stück

Gesamtwert: €16.893

Analog zum Aktienindex sollen Sie eine Klasse `Portfolio` mit entsprechenden Attributen erzeugen, um Portfolio im Programm zu repräsentieren.

Um einen Portfolioeintrag darzustellen bzw. die Verbindung zwischen einer Aktie und ihrer Stückzahl in einem Portfolio herzustellen, erstellen Sie eine eigene Klasse `PortfolioEntry`. Ein Portfolioeintrag beinhaltet somit eine Aktie und die Stückzahl dieser Aktie. Um eine Aktie zum Portfolio hinzuzufügen, soll ein neuer Portfolioeintrag erzeugt werden und zu der Liste der Portfolioeinträge im Portfolio hinzugefügt werden.

Der Gesamtwert eines Portfolios kann auf folgende Weise berechnet werden:

Für jeden Portfolioeintrag im Portfolio:
GesamtwertDesPortfolios = GesamtwertDesPortfolios + Aktienwert * Aktienstückzahl

Observer Pattern

Da sowohl der Indexstand eines Indexes als auch der Gesamtwert eines Portfolios direkt von den Werten einzelner Aktien abhängen, soll sich jede Änderung eines Aktienwertes in der Änderung des Indexstandes und den Gesamtwerten einzelner Portfolios widerspiegeln. Um das zu gewährleisten und auch die Funktionalitäten zwischen einer Aktie, eines Indexes und eines Portfolios zu trennen, verwenden Sie das **Observer Pattern** (siehe Abbildung 1).

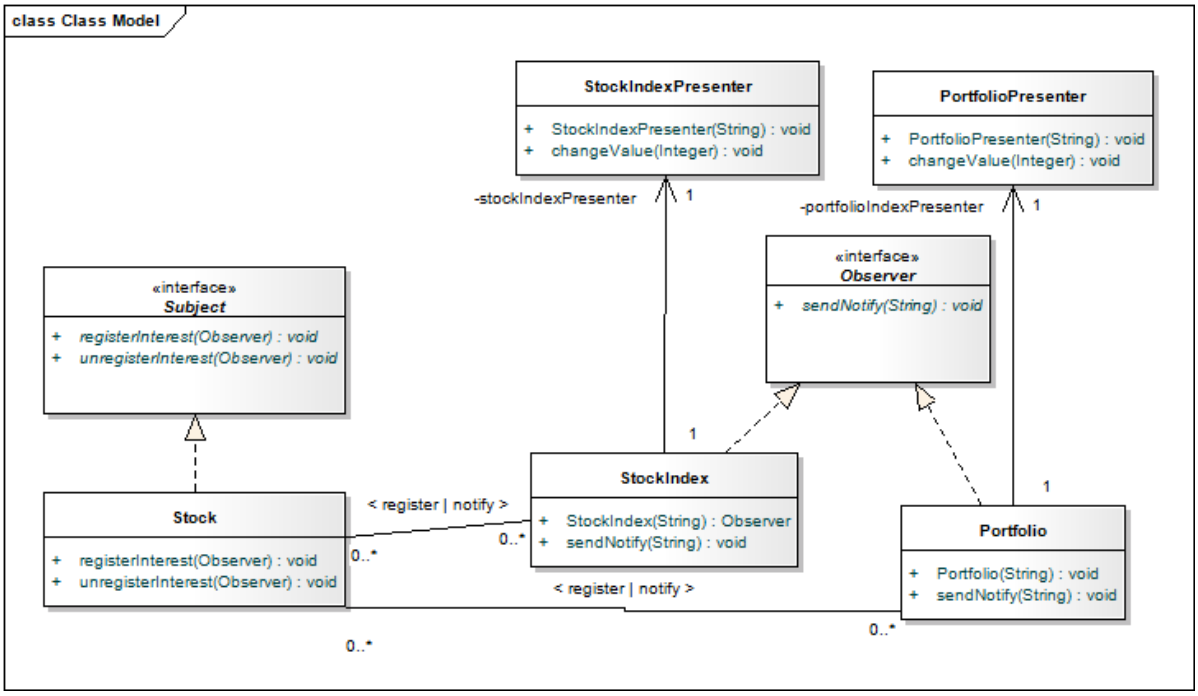

Abbildung 1: Observer Pattern

Im Kontext des hier dargestellten Observer Patterns stellt eine Aktie das *Subject* und das Portfolio den *Observer* dar. Um ein „Interesse" an der Änderung des *Subjects* (`Stock`) bekannt zu geben, muss sich der Observer (z.B. Objekt `myPortfolio` der Klasse `Portfolio`) bei dem *Subject* registrieren. Ein Hinzufügen einer Aktie (`Stock`) zum Portfolio soll folgendermaßen erfolgen:

Erzeuge einen neuen Portfolioeintrag mit dieser Aktie und einer bestimmten Stückzahl.
Füge diesen Portfolioeintrag zur Portfolioeintragsliste hinzu.
Rufe die Methode `registerInterest` dieser Aktie auf und übergebe das Portfolio als Parameter.

Beim Entfernen einer Aktie aus einem bestimmten Portfolio soll der entsprechende Portfolioeintrag aus der Liste der Portfolioeinträge im Portfolio entfernt werden und die Methode `unregisterInterest(...)` der Aktie im Portfolioeintrag aufgerufen werden.

Auch der Aktienindex (z.B. Objekt `myIndex` der Klasse `Stockindex`) stellt einen *Observer* dar, der an den Änderungen eines Aktienwertes „interessiert" ist, und soll sein „Interesse" an der Änderung dieser Aktie (`Stock`) bekannt

geben. Das Hinzufügen einer Aktie zum Aktienindex soll folgendermaßen erfolgen:

> ***Erzeuge einen neuen Aktienindexeintrag mit dieser Aktie und einer bestimmten Gewichtung.***
> ***Füge diesen Aktienindexeintrag zur Aktienindexeintragsliste hinzu.***
> ***Rufe die Methode `registerInterest` dieser Aktie auf und übergebe den Aktienindex als Parameter.***

Die Methode `registerInterest(obs Observer)` der Klasse `Stock` speichert Observerobjekte (z.B. `myPortfolio oder myIndex)` in einer Liste.

Die Klasse `Stock` soll eine Methode zum Ändern ihres Aktienwertes bereitstellen (z.B. `setValue(...)`). Innerhalb dieser Methode soll der Wert der Aktie neu gesetzt und alle registrierten *Observer* über diese Änderung benachrichtigt werden. Dies kann folgendermaßen implementiert werden:

> ***Setze den neuen Aktienwert.***
> ***Für jeden Observer in der ObserverListe:***
> ***Rufe die Methode `sendNotify(...)` auf.***

Dadurch wird *Polymorphismus* praktisch angewendet, da die Methode auf gleiche Weise für Objekte unterschiedlichen Typs aufgerufen wird.

Die Methode `sendNotify(...)` kann in der Klasse `Portfolio` folgendermaßen implementiert werden:

> ***Berechne den GesamtwertDesPortfolios.*** *//siehe oben*
> ***Gib diesen Wert mit Hilfe der Methode `portfolioPresenter.changeValue(…)` aus.***

Die gleiche Methode kann in der Klasse `StockIndex` folgendermaßen implementiert werden:

> ***Berechne den Indexstand.*** *//siehe oben*
> ***Gib diesen Wert mit Hilfe der Methode `stockIndexPresenter.changeValue(…)` aus.***

Die Klasse `Stock` soll das im jar File mitgelieferte Interface `Subject` und die Klassen `StockIndex` sowie `Portfolio` das Interface `Observer` implementieren (siehe Abbildung 1): die Klassen müssen alle in den Interfaces definierten Methoden beinhalten und die entsprechende Funktionalität bereitstellen. In Java wird dies mit dem Schlüsselwort `implements` realisiert. Die Klasse Portfolio soll die im jar File mitgelieferten Klasse `PortfolioPresenter` für die Ausgabe benutzen. Objekte dieser Klasse stellen graphische Fenster dar und bieten die Möglichkeit, die aktuellen Portfoliowerte anzuzeigen. Um eine Instanz dieser Klasse zu benutzen, deklarieren Sie eine Variable dieser Klasse (z.B. `private PortfolioPresenter portfolioPresenter;`) und erzeugen Sie die Instanz im Konstruktor (z.B. `portfolioPresenter = new PortfolioPresenter(name);`). Damit wird die navigierbare UML-Assoziation zwischen den Klassen `Portfolio` und `PortfolioPresenter` (siehe Abbildung 1) in Java implementiert. Der `name` im Konstruktor von `PortfolioPresenter` wird in der Titelleiste des Fensters angezeigt. Um die Änderungen der Portfoliowerte anzuzeigen, benutzen Sie die Methode `changeValue (int value)`. Machen Sie dies analog auch für die Klasse `StockIndex`, welche die Klasse `StockIndexPresenter` benutzen soll.

Die Klassendeklarationen sollen folgendermaßen aussehen:

```
public class Stock implements Subject {…
public class Portfolio implements Observer {…
public class StockIndex implements Observer {…
```

Anforderungen im Detail:

Schreiben Sie ein Programm, mit welchem Sie mehrere Aktienportfolios und den Börsenindex ständig verfolgen können. Folgende Funktionalität ist zu erfüllen:

- **Textbasiertes User-Interface für die Dateneingabe (nur Konsole, kein GUI):**
  Es soll ein textbasiertes Menü erstellt werden, welches dazu dient, die folgende Funktionalität zu steuern.

- **Automatische Testdatengenerierung (beim Start Ihres Programms):**
  - Erzeugen Sie mindestens zehn verschiedene Aktien.
  - Legen Sie mindestens zwei verschiedene Portfolios an.

- Legen Sie mindestens drei Aktienindizes an, wobei der „ATX-Five“ enthalten sein muss. Die Aktienindizes müssen Sie nicht über das Menü veränderbar sein, es genügt, wenn Sie diese „hardcoded“ implementieren.

- **Verwaltung:**
  - Aktie zum System hinzufügen
  - Verfügbare Aktien anzeigen
  - Portfolio anlegen
  - Portfolio entfernen
  - Aktie zu einem Portfolio hinzufügen (Das Portfolio soll sich als Observer bei dieser Aktie anmelden)
  - Aktie aus einem Portfolio entfernen (Das Portfolio soll sich als Observer bei dieser Aktie abmelden)

- **Änderungen:**
  - Kurs einer Aktie ändern (Die Aktie soll alle registrierten Observer benachrichtigen.)

Jede solche Änderung soll sich in einer graphischen Ausgabe, wie in **Abbildung 2** illustriert, niederschlagen (Anwendung des Observer Patterns). Für Testzwecke kann man die Werte zusätzlich auch auf der Konsole ausgeben.

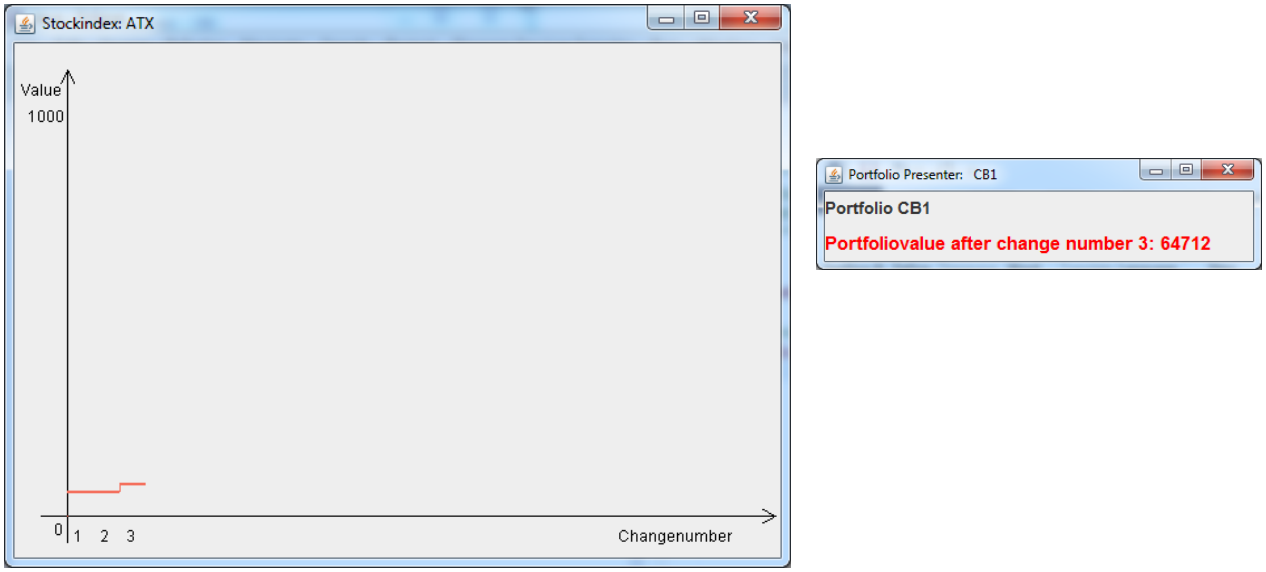

Abbildung 2: Graphische Ausgabe

Hinweise:

- Verwenden Sie **nicht** die in der Java-Plattform vorhandene Implementierung des Observer-Patterns, da dieses für unsere Zwecke zu umfangreich ist und das selbständige Implementieren des Observer-Patterns das Verständnis wesentlich erhöht. Importieren Sie daher **weder** `java.util.Observer` **noch** `java.util.Observable`.

- Für Eclipse-Benutzer: Für einen reibungslosen Ablauf mit Eclipse erzeugen Sie am besten ein *lib* Verzeichnis, in dem Sie das jar File (`Presenter.jar`) speichern, und inkludieren sie es in den Java Classpath (Project → Properties → Java Build Path → Libraries → Add Jar File). Dieses jar File enthält die Klassen `PortfolioPresenter` und `AktienPresenter` sowie das `Observer` und `Subject` Interface.

- Um einen String von der Konsole einzulesen, können Sie folgende Methode definieren:

```
private String readLine() {
       BufferedReader keyb = new BufferedReader(new InputStreamReader(System.in));
       String helpString;
       try {
              helpString = keyb.readLine();
              return helpString;
       } catch (Exception e) {
```

```
                System.out.println("Input Error");
                return null;
        }
}
```

(`java.io.BufferedReader` und `java.io.InputStreamReader` müssen Sie importieren.)

- Um ein Menü zu erzeugen, können Sie eine `switch`-Kontrollschleife benutzen (http://docs.oracle.com/javase/tutorial/java/nutsandbolts/switch.html):

```
int selection;
boolean isRunning = true;
while (isRunning) {// Endlessloop for menu, terminated with "0"
        …
        // read selection from command line!
        …
        switch (selection) {
        …
        case 0:
                isRunning = false; // End
                …
        }
        …
}
```