Dokumentation

Java Projekt

2020

Projektname: Aktien-Depot

Kurs: ITA19

Matrikelnummer: 2533282

Dozent: Dr. Werner Jost

# Grobe Spezifikation der gewünschten Leistung

## Ist-Zustand

Der Kunde hat in der Vergangenheit Aktien von Firmen aus dem DAX Leitindex gekauft. Die Transaktionen hat der Kunde in einer CSV Datei dokumentiert (Datei: database/KaufHistorie.csv). Ein Eintrag (eine Zeile) in dieser Dokumentation enthält:

* Das Datum des Aktienkaufs bzw. -Verkaufs
* Das offizielle Aktienkürzel der gekauften bzw. verkauften Aktie (z.B. DAI.DE für Daimler-Aktien)
* Den Preis der Aktie, für den die Aktie gekauft bzw. verkauft wurde
* Die Anzahl der gekauften bzw. verkauften Aktien und
* Die durchgeführte Aktion („k“ für kaufen und und „v“ für verkaufen)

Die historischen Kursdaten der Firmen im DAX sind in CSV-Dateien gespeichert und werden im Internet bereitgestellt (z.B. <https://finance.yahoo.com>).

## Soll-Zustand

1. Der Kunde möchte alle historischen Kursdaten der 30 DAX Unternehmen aus dem Internet laden und auf seinem lokalen Rechner in einem Ordner „database“ speichern.
2. Der Kunde möchte, dass die CSV-Datei, in der er seine Kaufhistorie dokumentiert hat, vom Programm ausgewertet wird. Der Kunde möchte sehen,
   1. welche Aktien an einem bestimmten Datum in seinem Depot vorhanden sind,
   2. wie viele Aktien er an diesem Datum von einer Firma besitzt,
   3. wie viel er im Durchschnitt für eine Aktie gezahlt hat,
   4. wie viel er bereits in den DAX investiert hat und
   5. ob seine Transaktionen gewinnbringend waren oder nicht.

Diese Informationen möchte er nach jeder Transaktion (also jedem Aktienkauf bzw. – verkauf) erhalten, um sich somit einen Überblick der zeitlichen Entwicklung seines Depots zu verschaffen.

Zusätzlich soll der Kunde die Möglichkeit erhalten:

1. die Kaufhistorie durch Käufe bzw. Verkäufe von Aktien zu erweitern und
2. sich alle historischen Kursdaten der im DAX gehandelten Aktien anzeigen zu lassen
   1. für eine ausgewählte Aktie (für alle für diese Aktie gespeicherten Daten der historischen Kursinformationen) und
   2. für ein ausgewähltes Datum (für alle historischen Kursinformationen der im Dax gehandelten Aktien).

## Entwurf

***Wer wird das System benutzen?***

Es gibt nur einen Benutzer. Der Benutzer verwaltet mit dem System sein Aktiendepot.

***Wer gibt Daten in das System ein?***

* Das System lädt die Kursdaten der 30 DAX-Unternehmen automatisch aus dem Internet.
* Der Kunde muss Transaktionsdaten (Käufe und Verkäufe von Aktien) in Form einer CSV-Datei dem System zur Verfügung stellen.
* Der Kunde kann über ein Formular Käufe und Verkäufe von Aktien durchführen. Diese Transaktionen werden der CSV-Datei hinzugefügt.

***Wer erhält Daten aus dem System?***

* Der Kunde erhält anhand der übergebenen Transaktionsdaten Informationen zur Wirtschaftlichkeit seiner Transaktionen vom System sowie einen Überblick über die zeitliche Entwicklung seines Depots.
* Der Kunde erhält die Informationen über die historischen Kursdaten der 30 DAX Unternehmen, je nach Auswahl eines Unternehmens über alle gespeicherten Daten oder über alle Unternehmen für ein konkretes Datum.

***Welche Aufgaben werden die Akteure mit dem System durchführen?***

Der Kunde kann mit dem System Transaktionen auswerten, die er mit Aktien der DAX-Unternehmen getätigt hat. Des Weiteren kann er Transaktionen simulieren und sie auf ihre Wirtschaftlichkeit überprüfen.

***Welches Ergebnis will der Akteur nach der Ausführung eines Anwendungsfalls haben?***

Der Kunde erhält nach Ausführung des Programms Informationen über die zeitliche Entwicklung seines Depots. Darin enthalten sind:

* der Investitionsbetrag,
* die von ihm gehaltenen Aktien mit jeweiliger Anzahl,
* der für eine Aktie bezahlte Durchschnittspreis sowie
* der aktuelle Wert Gewinn bzw. Verlust.

***Welches Ziel hat er?***

Sein Ziel liegt in der Auswertung von Daten.

***Was ist der Standard-Fall eines Anwendungsfalls?***

Der Kunde wertet eine CSV-Datei seiner Transaktionen (Aktienkäufe und -verkäufe) aus.

***Welche Erweiterungen und Sonderfälle sind bei einem Anwendungsfall***

***möglich?***

Der Kunde erweitert seine CSV-Datei mit Transaktionen (Aktienkäufe und -verkäufe) über das Programm.

|  |  |
| --- | --- |
| Anwendungsfall 1 | Kursdaten der 30 DAX-Unternehmen aus dem Internet laden und auf dem lokalen Rechner speichern. |
| Ziel | Die Kursdaten der 30 DAX-Unternehmen in Form von CSV-Dateien befinden sich auf dem lokalem Rechner. |
| Vorbedingung | Internetverbindung |
| Nachbedingung Erfolg | CSV-Dateien werden im Ordner „database“ gespeichert. Sie werden für die weiteren Anwendungsfälle benötigt |
| Nachbedingung Fehler | Das Programm terminiert. |
| Akteure | Kunde |
| Auslösendes Ereignis | Beim Start des Programmes |
| Erweiterungen | Depot erstellen |

|  |  |
| --- | --- |
| Anwendungsfall 2 | Depot erstellen |
| Ziel | Anhand von Transaktionsdaten wird ein Depot angelegt. Im Depot sind Aktien gespeichert, die gekauft bzw. verkauft wurden. Das Depot beinhaltet Informationen über die Anzahl der gekauften Aktien und über den Durchschnittspreis der gekauften Aktien. |
| Vorbedingung | Der Kunde hat eine CSV-Datei mit Informationen über seine Transaktionen dem Programm zur Verfügung gestellt. Die Transaktionen beziehen sich auf Aktien im DAX.  Das Programm hat die Kursdaten der 30 DAX Unternehmen erfolgreich runtergeladen. |
| Nachbedingung Erfolg | Ein Depot wurde erstellt durch sukzessive Auswertung der Transaktionen, d.h. Aktien, die gekauft wurden, werden dem Depot hinzugefügt und Aktien, die verkauft wurden, werden aus dem Depot entfernt. Für die Aktien im Depot wird der durchschnittliche Einkaufskaufpreis ermittelt. |
| Nachbedingung Fehler | Ein Depot konnte nicht erstellt werden, das Programm terminiert. |
| Auslösendes Ereignis | Dem Programm wurde eine CSV-Datei mit Transaktionen übergeben. |
| Erweiterung | Die zeitliche Entwicklung des Depots wird ausgegeben. |

|  |  |
| --- | --- |
| Anwendungsfall 3 | Die zeitliche Entwicklung des Depots wird ausgegeben |
| Ziel | Es wird eine CSV-Datei erstellt, in der die Zeitliche Entwicklung des Depots abgebildet wird. |
| Vorbedingung | Es liegt ein Depot vor |
| Nachbedingung Erfolg | Die Zeitliche Entwicklung des Depot wird in der Konsole ausgegeben. |
| Nachbedingung Fehler |  |
| Akteure | Ein Kunde |
| Auslösendes Ereignis | Dem Programm wurde eine CSV Datei mit Transaktionen übergeben. |

## Nutzerschnittstelle (Ein- und Ausgabe)

Für das Nutzerkonzept wurde ein Use-Case- Diagramm erstellt (siehe Abbildung 1).

Ein Bild, das Text, Karte, Screenshot enthält.

Automatisch generierte Beschreibung

Abbildung 1: Use-Case-Diagramm

Die vier Nutzerschnittstellen sind:

1. Die Ausgabe der zeitlichen Entwicklung des Depots. Dazu muss das Depot unter Verwendung der Kauf- und Verkaufhistorie und der Kursdaten der entsprechenden Aktien zunächst erstellt werden.
2. Der Kauf einer bestimmten Anzahl von Aktien zu einem bestimmten Datum.
3. Der Verkauf einer bestimmten Anzahl von Aktien zu einem bestimmten Datum.
4. Die Anzeige von Kursdaten der Aktien gefiltert nach
   1. entweder einer Aktie, deren Kursdaten über den gespeicherten Zeitraum, angezeigt werden oder
   2. einem Datum, um die Kursdaten aller Aktien zu diesem Datum anzuzeigen.

### Aktivitätsdiagramm

Das Aktivitätsdiagramm ist in der Abbildung 2 dargestellt und wird im folgendem genauer erleutert.

Ein Bild, das Karte, Text enthält.

Automatisch generierte Beschreibung

Abbildung 2 Aktivitätsdiagramm

### Einlesen der Kursdaten aller DAX-Aktien

Nach dem Start des Programms erscheint zunächst folgende Ausgabe ohne jegliche Vorbedingung:

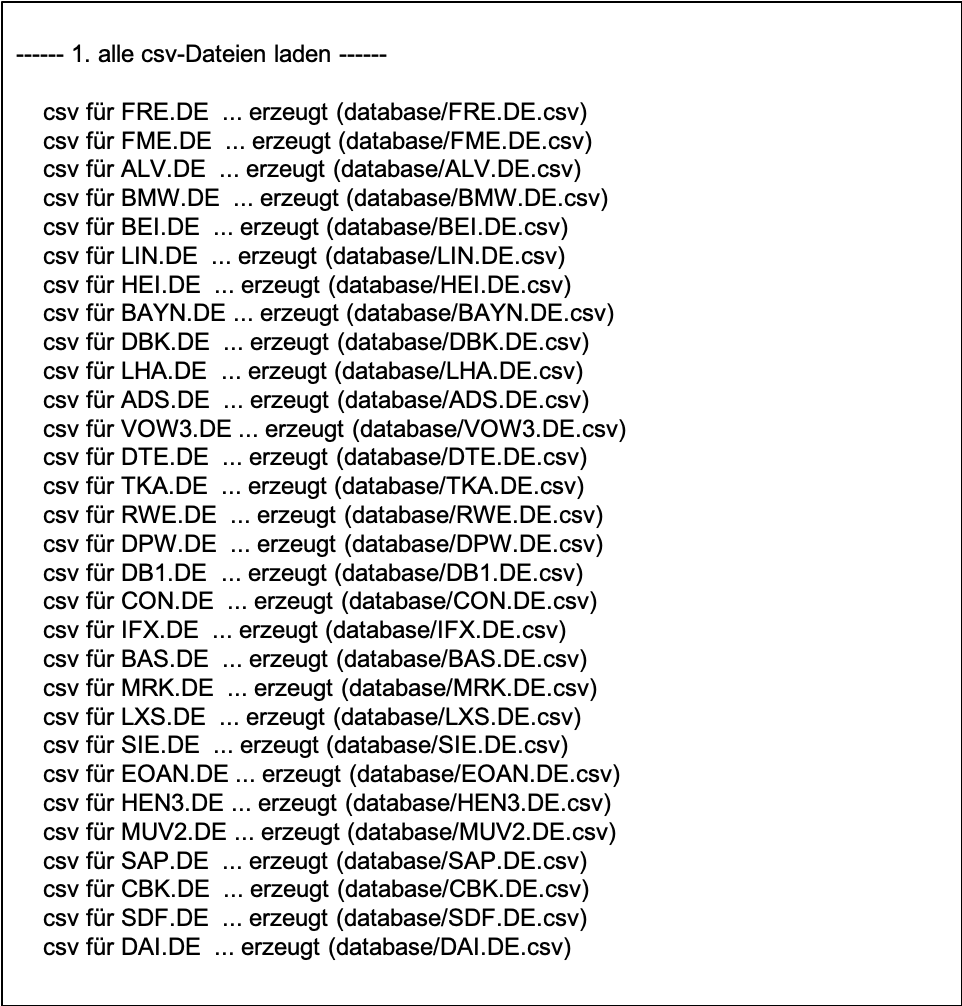


Abbildung 3: Ausgabe des Einlesens der historischen Kursdaten aller DAX-Aktien (Schritt 1)

Die Kursdaten werden aus dem Internet gezogen (finance.yahoo.com) und die CSV-Dateien im „database“-Ordner abgelegt.

### Database-Objekt erstellen

Aus diesen Dateien wird ein Objekt vom Typ „Database“ erstellt. Dazu geschieht folgendes:

1. jede Zeile in einer solchen CSV-Datei wird zu einem Objekt vom Typ „TagesInfo“ eingelesen,
2. eine gesamte CSV-Datei (die Menge aller TagesInfo-Objekte einer Aktie) wird zu einem Objekt vom Typ „HistorischeDatenListe“,
3. alle CSV-Dateien (die Menge aller HistorischeDatenListe-Objekte) werden zu einem „Database“-Objekt.

Für die Erstellung des Database-Objektes erscheint folgende Ausgabe (siehe Abbildung 3):

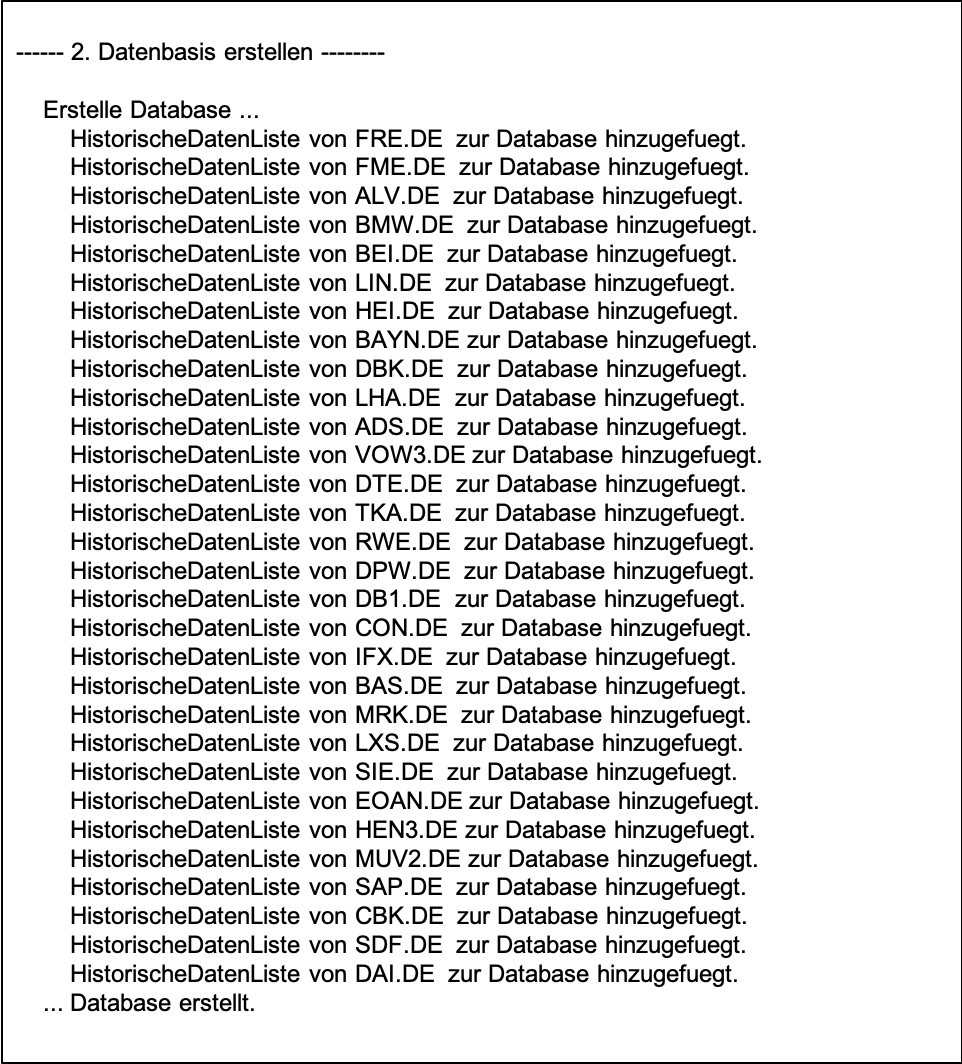


Abbildung 4: Ausgabe zur Erstellung des Database-Objektes

Das Einlesen der Kursdaten aller Aktien sowie die Erstellung des Database-Objektes erfolgt ohne Vorbedingung. Um das Depot zu erstellen, wird eine Kauf- und Verkaufhistorie benötigt. Diese wird in der KaufHistorie.csv zur Verfügung gestellt.

### Depot-Erstellung

Um das Depot zu erstellen, werden zunächst die Kauf- und Verkaufhistorie eingelesen. Dazu wird die Datei KaufHistorie.csv aus dem database-Ordner eingelesen. Diese muss als Vorbedingung existieren. Über das Einlesen der Kauf- und Verkaufhistorie wird der Nutzer informiert (siehe Abbildung 4)

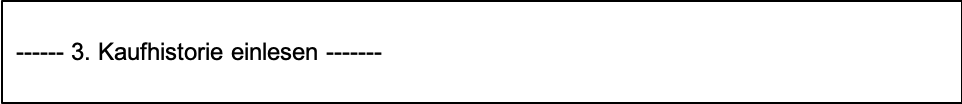


Abbildung 5: Information über das Einlesen der Kauf- und Verkaufhistorie

Wir nehmen im Folgenden an, die eingelesene KaufHistorie.csv sieht wie folgt aus:

|  |  |  |  |
| --- | --- | --- | --- |
| Datum | Aktien-Name | Anzahl | Handelsaktion |
| 2019-10-02 | SIE.DE | 10 | k |
| 2019-10-08 | SIE.DE | 20 | k |
| 2019-09-25 | SIE.DE | 30 | k |
| 2019-10-04 | IFX.DE | 30 | k |
| 2019-10-10 | IFX.DE | 30 | k |
| 2019-11-01 | DAI.DE | 20 | k |
| 2019-11-11 | DAI.DE | 100 | k |
| 2019-11-11 | SIE.DE | 100 | k |
| 2019-11-11 | SIE.DE | 100 | k |
| 2019-11-12 | SIE.DE | 100 | v |
| 2019-11-12 | DAI.DE | 50 | v |
| 2020-08-06 | DAI.DE | 20 | v |
| 2020-08-06 | ADS.DE | 10 | k |

Mithilfe der Kauf- und Verkaufhistorie wird das Depot erstellt.

### Investitionsbetrag und Gewinn beziehungsweise Verlust bestimmen

Das Programm berechnet anhand der Transaktionen, wie viel Geld investiert wurde, um die Aktien zu erwerben. Der investitionsbetrag, wird mit dem aktuellem Wert der Aktien verglichen. Die Differenz zwischen dem Investitionsbetrag und dem aktuellen Wert der Aktien wird als Gewinn oder Verlust angegeben.

### Über das Depot wird der Nutzer wie in Abbildung 5 gezeigt, informiert:

Ein Bild, das Himmel enthält.

Automatisch generierte Beschreibung

Abbildung 6: Ausgabe bei der Depoterstellung (gekürzt)

### Das Kaufen- und Verkaufenformular

Nach der Depoterstellung erscheint folgende Ausgabe auf der Konsole:

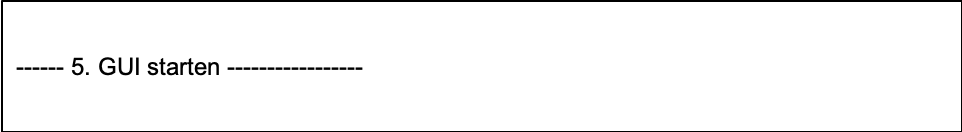


Abbildung 7: Ausgabe der Information, dass die GUI gestartet wurde

und das Fenster öffnet sich:

Ein Bild, das Screenshot enthält.

Automatisch generierte Beschreibung

Abbildung 8: Formulare zum Kauf bzw. Verkauf von Aktien

Sowohl beim Kaufen als auch beim Verkaufen müssen das jeweilige Datum und die Anzahl eingegeben sowie die Aktie ausgewählt werden. Sowohl die Eingabe des Datums als auch die Eingabe der Anzahl wird geprüft. Ist das Datum nicht im korrekten Format bzw. ist die Anzahl keine Zahl erscheint jeweils ein Fenster als Nachricht – die Eingabe muss korrigiert werden.

Ein Bild, das Screenshot enthält.

Automatisch generierte Beschreibung

Abbildung 9: jeweilige Warnfenster über fehlerhafte Eingabe

Es wird außerdem geprüft, ob das Datum in den Kursdaten der Aktie überhaupt vorkommt. Dazu wird das Database-Objekt verwendet. Taucht es nicht auf, wird ebenfalls eine Warnung ausgegeben (jedoch auf Konsole – JOptionPane hier in der Ausnahmebehandlung noch nicht implementiert).

error.TagesInformationenNichtVorhanden: Es existieren keine Tagesinformationen zu dem Datum 2020-08-01

Existiert das entsprechende Datum in den Kursdaten der Aktie wird daraus der Durchschnittspreis der Aktie an diesem Tag entnommen und die Aktie zu diesem Preis ge- bzw. verkauft. Die neue Transaktion wird in KaufHistorie.csv gespeichert und das Depot erneut erstellt und ausgegeben.

### Statistik aus den Kursdaten

Der Reiter „Statistik“ in der GUI zeigt den Kursverlauf einer Aktie (am Anfang SIE.DE), siehe Abbildung 9.

Ein Bild, das Screenshot enthält.

Automatisch generierte Beschreibung

Abbildung 10: Darstellung des Kursverlaufes einer Aktie (hier SIE.DE)

Die Daten werden dazu aus der entsprechenden CSV-Datei (z.B. database/SIE.DE.csv) eingelesen. Mit den Listen rechts oben hat der Nutzer die Möglichkeit, sich entweder den Kursverlauf einer anderen Aktie oder ein Datum auszuwählen, um sich die Kursdaten aller Aktien zu diesem Datum anzuzeigen (siehe Abbildung 10).

Ein Bild, das Screenshot enthält.

Automatisch generierte Beschreibung

Abbildung 11: Darstellung der Kursdaten aller Aktien zu einem Datum (hier 2020-08-06)

## Design Pattern

Im Folgenden soll noch kurz auf einige Implementierungsdetails eingegangen werden. Insbesondere soll kurz die Verwendung von Design Pattern erläutert werden.

### Observer Pattern

Bei allen Ereignisbehandlungen für Nutzereignisse auf der GUI wird (implizit) das Observer Pattern verwendet. Es wurde stets in der folgenden Form (als anonyme Klasse) implementiert:

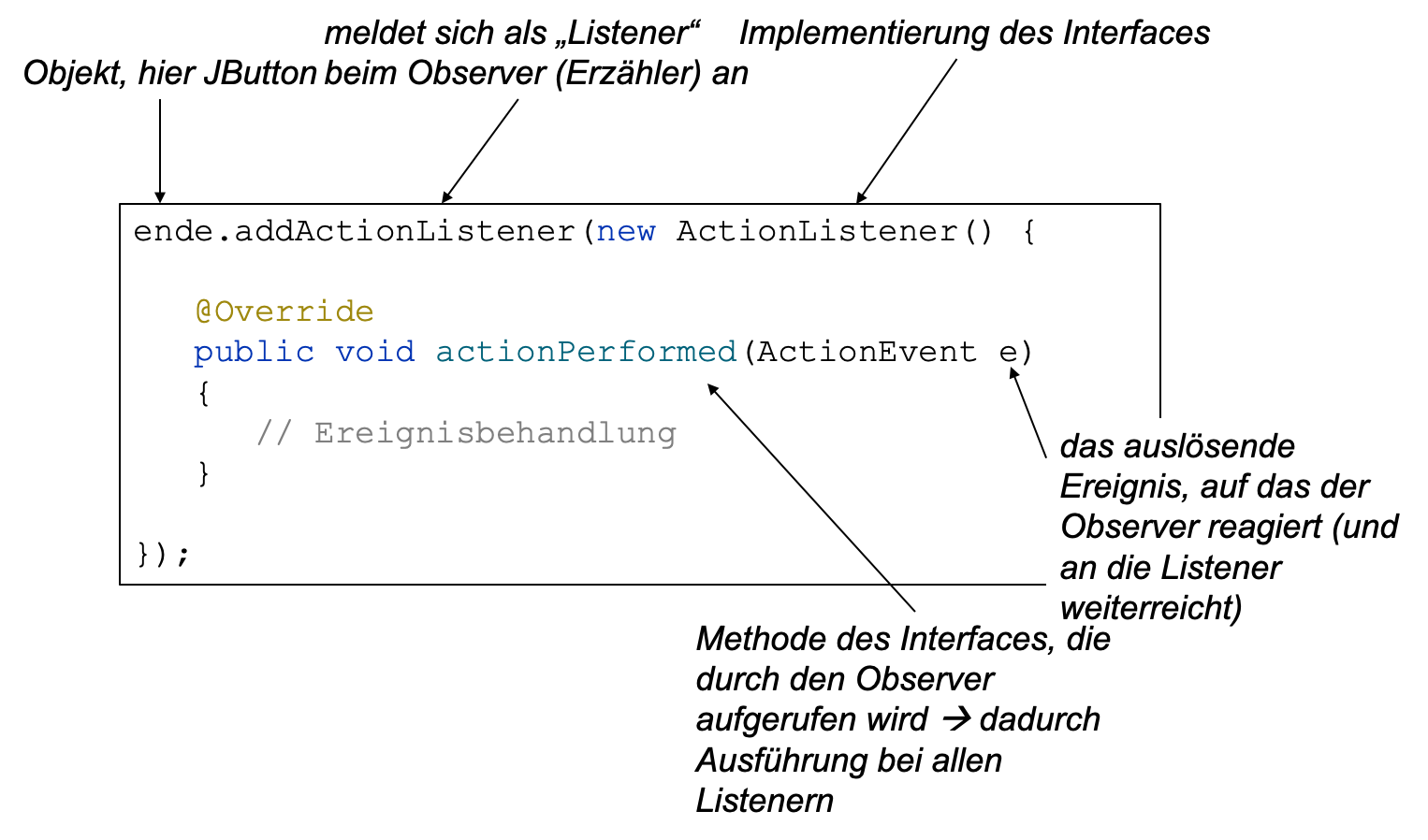


Abbildung 12: aus view/MainWindow.java

Wie hier für den ActionListener eines JButton-Objektes wurden auch ActionListener für JComboBox-Objekte (Datum und Anzahl) und weitere JButton-Objekte (Kaufen und Verkaufen) implementiert. Außerdem wurde in ähnlicher Art auch der ItemListener (itemStateChanged(ItemEvent e)) implementiert.

### Facade Pattern

Die beiden Formulare für das Kaufen und das Verkaufen von Aktien sehen gleich aus und unterscheiden sich nur im Titel und den Labels. Es wurde deshalb entschieden, eine Klasse TradingPanelFactory.java zu erstellen, die die Erzeugung eines solchen Formulars (als JPanel) zur Aufgabe hat. Dem Konstruktor dieser Klasse müssen dann nur noch die entsprechenden Werte für die Label-Beschriftungen und den Titel übergeben werden.

Man hätte anstelle des Konstruktors auch eine *Fabrikmethode* verwenden können, die das entsprechende JPanel-Objekt erzeugt. Allerdings wird hier mittels *Dependency Injection* ein Objekt von Depot übergeben, welches an die jeweiligen Formulare weitergereicht wird (um beim Kaufen oder Verkaufen von Aktien das Depot zu aktualisieren). Aufgrund der dann entstandenen Vermischung von Klassen- und Objekteigenschaften) wurde sich gegen eine statische Fabrikmethode und für den Konstruktor als Fassade entschieden.

### Adapter Pattern

In dem Statistik-Tab werden mithilfe einer JTable die Kursdaten einer Aktie bzw. aller Aktien zu einem Datum angezeigt. Die Verwendung eines Models (das von der abstrakten Klasse AbstractTableModel, die das Interface TableModel implementiert, abgeleitet ist) für die Tabelle entspricht einem Adapter. Das Interface TableModel enthält einige wenige Methoden, mit denen Tabellen generiert werden können, wie z.B. getColumnCount(), getRowCount(), getColumnName(), getValueAt(row, col). Zwischen der Darstellung der Tabelle und den eigentlichen Daten (Spaltennamen und Daten) bedarf es nun nur noch eines Adapters, der beides zusammenbringt. Das ist in meinem Code die Klasse MyTableModel. Einem Objekt dieser Klasse werden dann nur noch die tatsächlichen Daten (data) und die Überschriften (titles) übergeben und das MyTableModel-Objekt wird der Tabelle übergeben:

this.model = new MyTableModel(data, titles);  
this.table = new JTable( this.model );

# UML Klassendiagramm

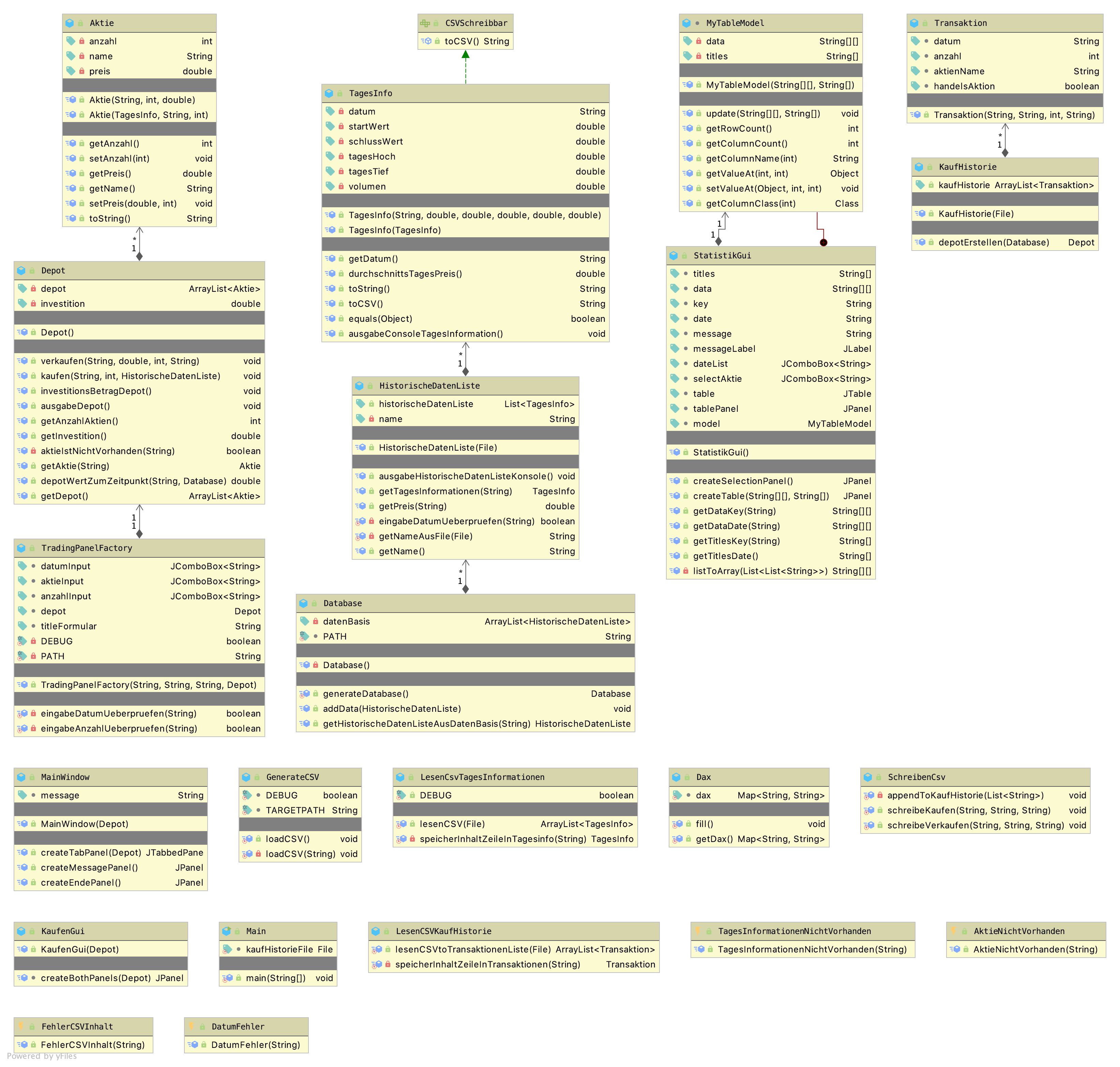


Abbildung UML Klassendiagramm

In Abbildung 13 wird das UML Klassendiagramm zum Aktien Projekt dargestellt. Die einzelnen Klassen werden mit ihrem Namen, ihren Klassen Attributen und ihren Methoden dargestellt. Die Klassen Attribute sind wenn möglich privat, um das Geheimhaltungsprinzip zu gewährleisten. Aus dem Klassendiagramm lassen sich die Assoziationen ablesen. In diesem Projekt stehen die meisten Klassen in Form einer Komposition in Beziehung zueinander. Das bedeutet zum Beispiel, das ein Objekt der Klasse TagesInformation nur so lange existiert wie ein Objekt der Klasse HistorischeDatenListe. Die meisten Assoziationen zwischen den Klassen im package model weisen eine Kardinalität von ***1 zu n*** auf. Zum Beispiel kann ein Objekt der Klasse HistorischenDatenListe mit beliebig vielen Objekten von der Klasse TagesInformationen in Beziehung stehen.