Hausarbeit im Modul Softwaredesign

„Lagerverwaltung“

Marius Mamsch

Gescherweg 165

48161 Münster

Niklas Devenish

Töpferstraße 91

48165 Münster

Jonas Elfering

Hagen 3

48624 Schöppingen

Hochschule Weserbergland

Studiengang: Wirtschaftsinformatik

Studiengruppe: WI44/14

Betreuender Dozent: Prof. Dr. Robert Mertens

Ausbildungsbetrieb:

Fiducia und GAD IT AG

GAD-Straße 2-6

48163 Münster

**Sperrvermerk**

Die vorliegende Arbeit beinhaltet Informationen über betriebsbezogene Prozesse sowie vertrauliche Daten des Unternehmens.

Sie darf nicht ohne ausdrückliche Genehmigung der

GAD eG

GAD-Straße 2-6

48163 Münster

veröffentlicht oder Dritten zugänglich gemacht werden.

1. Inhaltsverzeichnis

[I Inhaltsverzeichnis I](#_Toc437989163)

[II Abkürzungsverzeichnis II](#_Toc437989164)

[III III](#_Toc437989165)

[IV Abbildungsverzeichnis III](#_Toc437989166)

[V Tabellenverzeichnis IV](#_Toc437989167)

[1 Verwendete Patterns 1](#_Toc437989168)

[1.1 Model-View-Controller 1](#_Toc437989169)

[1.2 Oberserver 1](#_Toc437989170)

[1.3 Command 2](#_Toc437989171)

[1.4 Strategy 2](#_Toc437989172)

[2 Besonderheiten 2](#_Toc437989173)

[3 Prototypen und Mockups 2](#_Toc437989174)

[4 Zusammenfassung 4](#_Toc437989175)

[5 Quellenverzeichnis 5](#_Toc437989176)

[VI Anhangsverzeichnis V](#_Toc396480339)

[VII Anhang A-1](#_Toc396480340)

1. Abkürzungsverzeichnis

|  |  |
| --- | --- |
| Abkürzung | Beschreibung |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

2. Abbildungsverzeichnis

[Abbildung 1: Koala **Fehler! Textmarke nicht definiert.**](file:///C:\Users\xgadmmh\Documents\Vorlage%20Praxisbericht.docx#_Toc396481379)

[Abbildung 2: Pinguine **Fehler! Textmarke nicht definiert.**](file:///C:\Users\xgadmmh\Documents\Vorlage%20Praxisbericht.docx#_Toc396481380)

[Abbildung 3: Dumme Steine **Fehler! Textmarke nicht definiert.**](file:///C:\Users\xgadmmh\Documents\Vorlage%20Praxisbericht.docx#_Toc396481381)

[Abbildung 4: Blumen 0-1](file:///C:\Users\xgadmmh\Documents\Vorlage%20Praxisbericht.docx#_Toc396485718)

1. Tabellenverzeichnis

[Tabelle 1: 3 Spalten **Fehler! Textmarke nicht definiert.**](#_Toc396485575)

[Tabelle 2: Tabelle Anhang **Fehler! Textmarke nicht definiert.**-**Fehler! Textmarke nicht definiert.**](#_Toc396485650)

# Verwendete Patterns

## Model-View-Controller

Das grundlegend verwendete Entwurfsmuster bei der Erstellung der Lagerverwaltung ist das Model-View-Controller-Entwurfsmuster (MVC). Dieses Entwurfsmuster ist für diese Anwendung geeignet, da es möglich ist, die fachliche Logik getrennt von der sichtbaren Oberfläche zu bearbeiten. Zudem wird durch das MVC-Entwurfsmuster ermöglicht, einen Zustand des Systems in einer Datei abzuspeichern und zu einem späteren Zeitpunkt neu zu laden.

Die fachliche Logik der Lagerverwaltung wird in verschiedenen Models abgebildet. Insgesamt werden in diesem Fall sechs Models eingesetzt. Das LagerVerwaltungsModel enthält alle notwendigen Methoden, um die Gesamtheit aller Buchungen zu verwalten. Zudem wird die Initialbefüllung der Lager in diesem Model vorgenommen. In Ergänzung zum LagerVerwaltungsModel existiert das LagerModel, welches dazu dient mit einzelnen Lagern zu interagieren und beispielsweise verwendet wird, um eine Buchung auf einzelnes Lager auszuführen. Des Weiteren existiert ein BuchungsModel. Darin sind sowohl alle Informationen, wie beispielsweise Anteile und Buchungstag, als auch die benötigten Methoden zur Erstellung und Verwaltung von Buchungen enthalten. Das BuchungsModel dient als abstrakte Oberklasse. Die konkrete Implementation der Methoden für eine Zu- oder Abbuchung erfolgt in zwei eigenen Models, dem ZuBuchungsModel und dem AbBuchungsModel. Abschließend existiert ein AnteilModel, welches benötigt wird, um eine Gesamtbuchung als Buchung mehrerer kleiner Anteile auf verschiedene Lager darstellen zu können. In Kombination mit dem Command-Pattern kann so das Speichern der Anteile ermöglicht werden.

Zur graphischen Darstellung der Informationen werden diverse View-Elemente genutzt. Diese View-Elemente kommunizieren über den LagerVerwaltungsController mit den Models. In dem Lagerverwaltungscontroller sind ebenfalls die Methoden für das Speichern, das Laden und einen Undo/Redo-Mechanismus implementiert.   
Das übergeordnete View-Element ist die VerwaltungsView, welche weitere View-Elemente enthält. Ein Beispiel dafür ist die detaillierte Übersicht eines Lagers. Die Informationen, wie viel Kapazität ein Lager besitzt oder welche Buchungen auf dieses lager getätigt wurden erhält diese DetailView vom Lagerverwaltungsmodel und dem LagerModel aufgrund einer Observable/Oberserver-Beziehung. Der LagerVerwaltungsController dient dazu den Models vorzugeben, welche View-Elemente sie benachrichtigen müssen, wenn sich ihr Informationsstand geändert hat. Falls der Benutzer beispielsweise eine neue Buchung tätigt, wird diese Information von der View über den LagerVerwaltungsController an die Models weitergegeben, dort wird der Informationsstand der Models aktualisiert und die neuen Informationen können in der View angezeigt werden.

Ein großer Vorteil bei dieser Vorgehensweise den Controller als Vermittlungsstelle zwischen Benutzeroberfläche und fachlicher Logik zu verwenden, ist die Unabhängigkeit von Logik und Oberfläche voneinander. Es ist möglich ein Model zu verändern, ohne dass die Benutzeroberfläche angepasst werden muss. Dies gilt auch umgekehrt. Des Weiteren ist es möglich den aktuellen Informationsstand der Models zu speichern und zu einem späteren Zeitpunkt neu zu laden, auch wenn sich der Informationsstand der Models verändert hat.

## Observable/Oberserver

In der GUI wurde das Observable/Observer Pattern genutzt um Veränderungen in den Models angemessen in der GUI darzustellen. Das LagerVerwaltungsModel implementiert das Interface Observable und wird von der LagerVerwaltungsView, die den Observer erweitert, observt. Das dient vor allem, dazu Änderungen an der laufenden Buchung im LagerVerwaltungsModell zu erkennen und die GUI hinsichtlich dieser Änderungen zu aktualisieren. Aber auch auf Änderungen der Struktur der Lager und auf Änderungen in der Buchungsliste wird hier geachtet, damit diese immer korrekt angezeigt werden.

Desweiteren werden Observable/Observer bei den LagerModels, die das Interface Observable implementieren, eingesetzt. So erweitert der LagerBaumKnoten den Observer und jeder Knoten überwacht ein Lager, welches er in der Baumstruktur repräsentiert. Hierdurch wird es ermöglicht, dass die Knoten immer den aktuellen Bestand des Lagers anzeigen, ohne immer den kompletten Baum aktualisieren zu müssen. Außerdem überwacht die DetailView als Observer das LagerModel, welches momentan angezeigt wird. Dadurch wird auch ermöglicht, dass die DetailView die aktuellen Informationen zum Lager anzeigt ohne ständig neu erstellt oder komplett aktualisiert zu werden.

Eine weitere Observable/Observer-Beziehung ist zwischen dem Controller und der BuchungsBar zu finden. Der Controller ist in diesem Fall das Observable und die BuchungsBar der Observer. In diesem Fall teilt der Controller seinem Observer mit, wenn sich der Redo- oder der Undo-Stack geändert haben. Aufgrund von diesen Informationen entscheidet die BuchungsBar, ob die Button für den Redo-, bzw. Undo-Mechanismus aktiviert oder deaktiviert werden.

## Command

## Strategy

Das Entwurfsmuster Strategy wurde im Kontext des Sortierers eingesetzt. Der Sortier besitzt eine Sortierstrategie. Die Sortierstrategie ist ein Interface, welches die Methode sortiere() bereitstellt. Dieser Methode müssen zwei BuchungsModel mitgegeben werden, die verglichen werden sollen. Die Methode sortiere() liefert true zurück wenn das zuerst mitgegebene BuchungsModel vor dem zweiten BuchungsModel angezeigt werden muss, ansonsten false. Es gibt mehrere konkrete Implementationen dieses Interface. Für jede mögliche Art die Buchungsliste zu sortieren eine. Diese konkreten Implementationen unterscheiden sich vor allem durch das Attribut nach welchem sie sortieren und durch die Sortierreihenfolge, beides ist am Namen der Implementationen erkennbar. Das Entwurfsmuster Strategy bietet den Vorteil, dass man nur einen Sortieralgorithmus implementieren muss, der mit dem Interface Sortierstrategie arbeitet, um alle Sortiermöglichkeiten abzubilden. Denn die konkrete Implementation der Sortierstrategie kann noch zur Laufzeit geändert werden. Diese Implementation ist sehr viel eleganter, als alternativ für jede Möglichkeit des Sortierens einen kompletten Sortieralgorithmus zu implementieren, denn das würde redundanten Code erzeugen.

# Besonderheiten

# Prototypen und Mockups

## Skizzen

### TreeView und DetailView

C:\Users\Marius\Pictures\Übersicht.tif

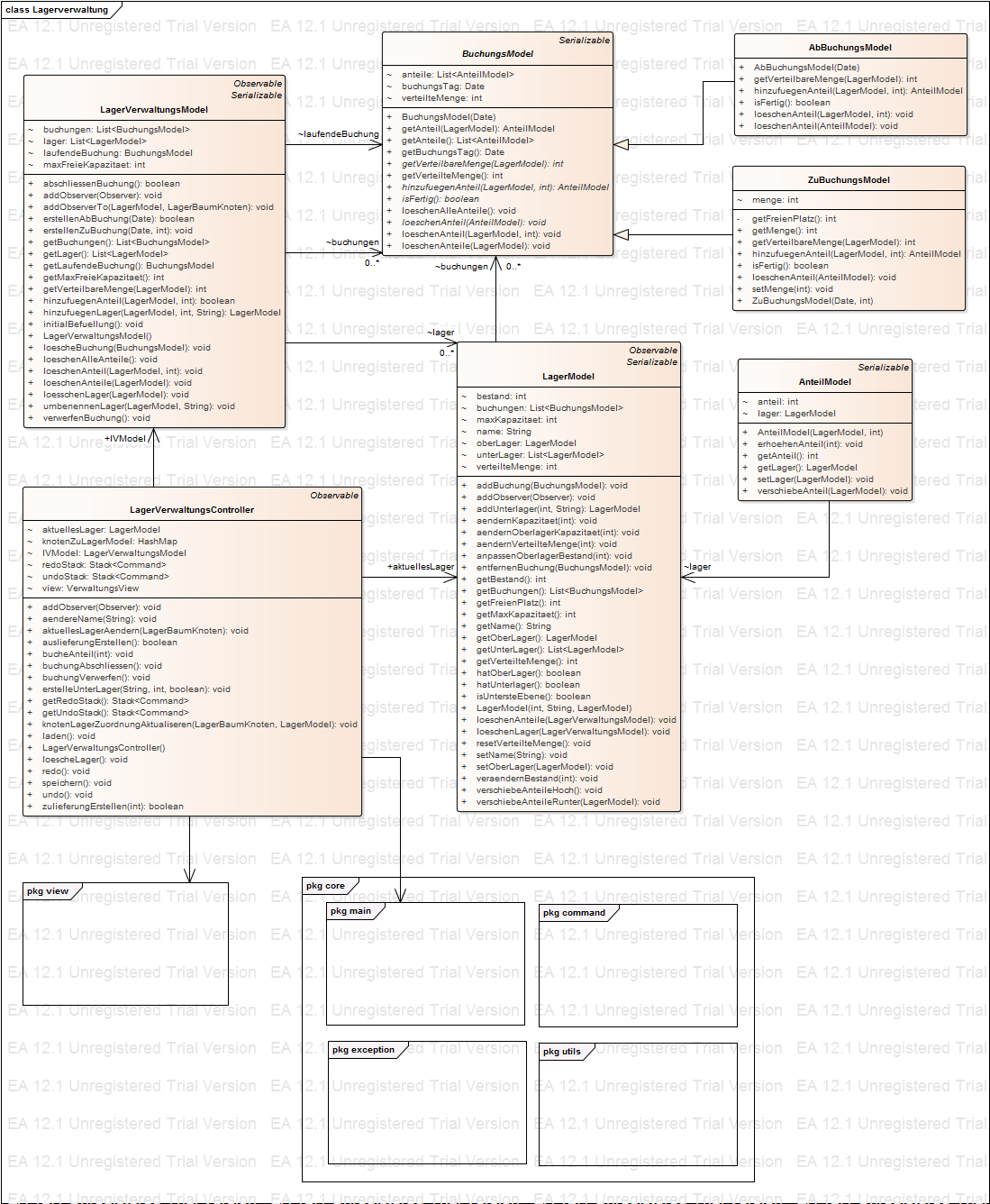
### Übersicht aller Buchungen

C:\Users\Marius\Pictures\alleBuchungen.tif

### Buchungsliste

C:\Users\Marius\Pictures\Buchungsliste.tif

## UML-Modell



# Zusammenfassung

# Quellenverzeichnis

Anhangsverzeichnis

[A1 Abbildungen A-1](#_Toc396481639)

[A1.1 Eigene Grafiken A-1](#_Toc396481640)

[A1.2 Bilder A-1](#_Toc396481641)

[A1.2.1 Aus Internet A-1](#_Toc396481642)

[A1.2.2 Aus Büchern A-1](#_Toc396481643)

[A2 Glossar A-1](#_Toc396481644)

Anhang

1. Abbildungen

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Tabelle : Tabelle Anhang

* 1. Eigene Grafiken
  2. Bilder



Abbildung : Blumen

* + 1. Aus Internet
    2. Aus Büchern

1. Glossar

**Eigenständigkeitserklärung**

„Wir versichern hiermit, dass wir diese Arbeit selbstständig verfasst, keine anderen Quellen und Hilfsmittel als die angegebenen benutzt und die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, in jedem einzelnen Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht haben. Das gleiche gilt auch für eingefügte Zeichnungen und, Kartenskizzen und Darstellungen.“

|  |  |
| --- | --- |
| Hameln, den 17.12.2015 |  |
| Ort, Datum | Unterschrift |

|  |  |
| --- | --- |
| Hameln, den 17.12.2015 |  |
| Ort, Datum | Unterschrift |

|  |  |
| --- | --- |
| Hameln, den 17.12.2015 |  |
| Ort, Datum | Unterschrift |