Dokumentation LagerMaster

Roy Ammerschuber

# Projektbeschrieb

Die Software die ich entwickle heisst «LagerMaster».

Die Software ist dafür da grosse Regallager zu simulieren und deren Inhalt abzuspeichern. Die Software wurde für Computerteilelager designt und optimiert, weswegen man CPUs, Festplatten und sonstige teile verwalten.

Das System kann eine X-beliebige menge an Regalen mit Inhalt speichern. Den Regalen kann man eine höhe und breite angeben sowie auch einen Namen damit man sie besser unterscheiden kann.

Die einzelnen Fächer können mit verschiedenen Teilen befüllt werden. In jedem Fach können mehrere verschiedene Items gelagert werden.

Die Items sind jeweils mit einem Namen, Menge, Gewicht und noch weiteren Teil spezifischen werten versehen.

Unter anderem werden folgende Patterns verwendet:

* Observable
* Singleton
* Factory

Die Regale werden in einer Grafischen Oberfläche dargestellt die sich dank des Observable Pattern immer live mit den Daten aktualisiert werden.

Die Datenbankanbindung und das Model werden über einen Singleton gesteuert. So hat man von überall drauf zugriff und hat nicht das Problem das man mehrere Model erstellt.

Die verschiedenen Items werden mithilfe einer Factory erstellt.

# Wie starte ich das Projekt

Öffnen sie IntelliJ und erstellen sie ein Projekt von Version Control. Wählen sie hier GitHub und laden das Projekt von folgendem Link herunter:

<https://github.com/royAmmerschuber/LagerMaster>

jetzt müssen sie noch in der Klasse java.conroller.MyDatabaseController die statischen MySQL Verbindungsdaten eingeben. Bitte achten sie darauf das die angegebene Datenbank schon erstellt und leer ist.

Bevor sie das Programm jetzt ausführen gehen sie sicher das die Datenbank gestartet ist.

# Anforderungen

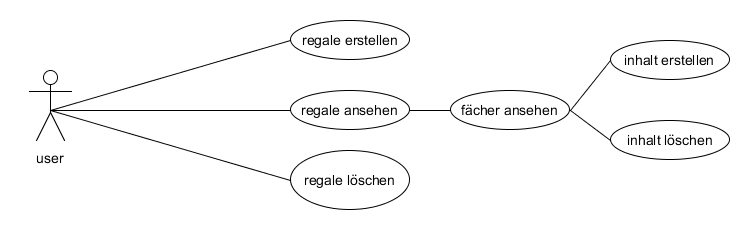
## Funktional

* Regale erstellen
* Regale löschen
* Regale ansehen
* Regalfächer bearbeiten
* Regalfächern Teile hinzufügen
* Aus Regalfächern Teile entfernen
* Error logging

## Nicht Funktional

* Singleton implementieren
* Factory implementieren
* Observable implementieren
* MVC Architektur
* Mindestens 3 abstrakte Klassen
* Mindestens 2 Entitäten in Datenbank
* Wird auf meisten Geräten ordentlich dargestellt

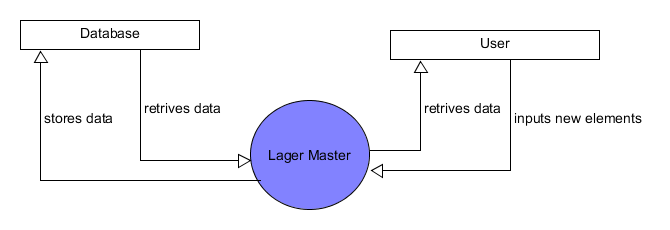
# Use Case Diagramm:



# Klassendiagramm

Das Klassendiagramm ist separat abgelegt.

# Kontextdiagramm



# Verwendete Libraries

* Spring-beans
* Spring-core
* Spring-jcl
* Spring-jdbc
* Spring-tx
* Mysql-connector-java
* junit
* testFx
* hamcrest
* loadui:testfx
* guava

# Verwendete Konstanten

* Controller
  + MyDatabaseController
    - DB\_URL
    - USER
    - PASS
  + DatabaseController
    - DEBUG
  + ErrorLogger
    - PATH
    - NAME
* Model
* View
  + ViewController
    - CELL\_WIDTH
    - CELL\_HEIGHT
  + CellView
    - HEADER\_HEIGHT
    - ROW\_HEIGHT

# Schnittstellenbeschreibung

Die einzige Schnittstelle die bei der Software eingebaut ist, ist die Einbindung der Datenbank diese wird über eine verbindungsklasse MyDatabaseController verwaltet. Diese Klasse implementiert das Singleton Verhaltensmuster und verwaltet die Verbindung, simples querybuilding und komplexere anfragen um direkt die Daten auszulesen.

# Welche Fehlerzustände können vorkommen?

Der einzige wirkliche Fehler der vorkommen kann ist ein Datenbank Verbindungsfehler wenn es versucht die Datenbank zu öffnen obwohl es sie nicht gibt oder der falsche Pfad angegeben wurde.

# Wie werden Fehler geloggt?

Das fehlerlogging wird mit der ErrorLogger Klasse erledigt. In ihren konstanten ist der Datei Pfad und der Dateiname. Im Ordner wird dann bei einer Exception eine neue Datei für den aktuellen tag erstellt die mit einem Datum versehen wird. Hier wird dann die Exception reingeschrieben mit der zeit an der es geschehen ist. Es zeigt einem auch eine Fehlermeldung die einem informiert das etwas schiefgelaufen ist und wo man die fehlermeldung findet.

# Testszenarien

## Modeltest:

Testet die Interaktionen mit dem Model und ob es auch die Datenbank korrekt aktualisiert

|  |  |
| --- | --- |
| Name | newShelf |
| Beschreibung: | Testet die Funktion zur Erstellung von neuen Regalen auf verschiedenen Grössen |
| Resultat: | Erfolgreich |

|  |  |
| --- | --- |
| Name | deleteShelf |
| Beschreibung: | Testet die lösch Funktion für regale. Kontrolliert ob alle daten aktualisiert wurden |
| Resultat: | Erfolgreich |

|  |  |
| --- | --- |
| Name | Reload |
| Beschreibung: | Testet die neu lade Funktion indem es die datenbank direkt bearbeitet. |
| Resultat: | Erfolgreich |

|  |  |
| --- | --- |
| Name | newItem |
| Beschreibung: | Testet die Funktionalität neue Items zu regalen hinzuzufügen. Es Testet dies für alle drei Sorten von Items. |
| Resultat: | Erfolgreich |

|  |  |
| --- | --- |
| Name | removeItem |
| Beschreibung: | Testet die Funktion zur Entfernung von Gegenständen aus dem Regal. |
| Resultat: | Erfolgreich |

|  |  |
| --- | --- |
| Name | removeItems |
| Beschreibung: | Testet die Funktion zum Löschen aller Items einer Regalzelle. |
| Resultat: | Erfolgreich |

## MyDatabaseControllerTest:

Testet die Datenbank Funktionen direkt über die Datenbank klasse.

|  |  |
| --- | --- |
| Name | insertShelf |
| Beschreibung: | Testet die Funktion zum einfügen von Regalen. |
| Resultat: | Success |

|  |  |
| --- | --- |
| Name | getShelfs |
| Beschreibung: | Testet die Funktion zum Holen von der Liste an regalen |
| Resultat: | Success |

|  |  |
| --- | --- |
| Name | deleteShelf |
| Beschreibung: | Testet die Löschfunktion von Regalen |
| Resultat: | Success |

|  |  |
| --- | --- |
| Name | insertItemBasic |
| Beschreibung: | Testet die Funktion zum Einfügen von ItemBasics |
| Resultat: | Success |

|  |  |
| --- | --- |
| Name | InsertItemCPU |
| Beschreibung: | Testet die Funktion zum Einfügen von ItemCPU |
| Resultat: | Success |

|  |  |
| --- | --- |
| Name | InsertItemDrive |
| Beschreibung: | Testet die Funktion zum Einfügen von ItemCPU |
| Resultat: | Success |

|  |  |
| --- | --- |
| Name | getIems |
| Beschreibung: | Testet die Funktion zum holen von Items in einer angegebenen Regalzelle |
| Resultat: | Success |

|  |  |
| --- | --- |
| Name | deleteItem |
| Beschreibung: | Testet die Funktion zum Löschen eines bestimmten Items aus einer Regalzelle |
| Resultat: | Success |

|  |  |
| --- | --- |
| Name | deleteItems |
| Beschreibung: | Testet die Funktion zum löschen aller Items in einer angegebenen regalzelle |
| Resultat: | success |

# Implementierungsentscheidungen

## abstrakte Klassen und Interfaces

### DatabaseController:

Die Datenbank klasse habe ich deswegen abstrakt gemacht, dass ich sie für spätere Projekte wiederverwenden kann.

### Item

Die basisklasse für Items. Speichert die Grunddaten die jedes Item beinhaltet

### ModelObservable:

Grundstruktur für beobachtbare Objekte. Beinhaltet Liste der Observer und eine Benachrichtigungsfunktion für diese.

### ModelObserver

Die Grundklasse für Beobachter des Models. Implementiert eine abstrakte getUpdate Funktion um die Benachrichtigung des Observables zu erhalten.

# Sicherheit

Um die Sicherheit habe ich mich bei diesem Projekt nicht wirklich gekümmert, da mit der Software eh alle Daten die abgespeichert sind erreichbar sind und persönliche daten des Nutzers nicht gespeichert werden.