**Projekt: Mosti**

Software Architektur Spezifikation

(Software Architecture Document)

[Dokumentstruktur basiert auf RUP „Software Architecture Document“]

# Dokumentinformationen

## Änderungsgeschichte

|  |  |  |  |
| --- | --- | --- | --- |
| *Datum* | *Version* | *Änderung* | *Autor* |
| 5.5.2016 | 1.0 | Erstellung der Software Architektur Spezifikation | Team 11 |
| 29.5.2016 | 2.0 | Aktualisierte Packagebeschreibung | Team 11 |
|  |  |  |  |
|  |  |  |  |

## Inhalt

[1 Dokumentinformationen 2](#_Toc452380321)

[1.1 Änderungsgeschichte 2](#_Toc452380322)

[1.2 Inhalt 3](#_Toc452380323)

[2 Einführung (Introduction) 6](#_Toc452380324)

[2.1 Definitionen und Abkürzungen (Definitions, Acronyms, Abbreviations) 6](#_Toc452380325)

[2.2 Referenzen (References) 6](#_Toc452380326)

[2.3 Übersicht (Overview) 6](#_Toc452380327)

[3 Architektonische Darstellung (Architectural Representation) 7](#_Toc452380328)

[4 Architektonische Ziele & Einschränkungen (Architectural Goals and Constraints) 7](#_Toc452380329)

[5 Logische Architektur (Logical View) 8](#_Toc452380330)

[5.1 Übersicht (Overview) 8](#_Toc452380331)

[5.2 Design Pakete (Architecturally Significant Design Packages) 10](#_Toc452380332)

[5.2.1 Package GUI 10](#_Toc452380333)

[5.2.1.1 Beschreibung des Package 10](#_Toc452380334)

[5.2.1.2 Schnittstellen 10](#_Toc452380335)

[5.2.2 Package Domain-Logik 10](#_Toc452380336)

[5.2.2.1 Beschreibung des Package 10](#_Toc452380337)

[5.2.2.2 Schnittstellen 10](#_Toc452380338)

[5.2.2.3 Operationen 10](#_Toc452380339)

[5.2.3 Package Persistenz 10](#_Toc452380340)

[5.2.3.1 Beschreibung des Package 10](#_Toc452380341)

[5.2.3.2 Schnittstellen 10](#_Toc452380342)

[5.3 Java-Pakete und deren Architektur-Design 11](#_Toc452380343)

[5.3.1 Package account 11](#_Toc452380344)

[5.3.1.1 Beschreibung des Packages 11](#_Toc452380345)

[5.3.1.2 Diagramme 11](#_Toc452380346)

[5.3.1.3 Schnittstellen 11](#_Toc452380347)

[5.3.1.4 Aufteilung der Klassen in das Drei-Schichten-Modell: 11](#_Toc452380348)

[5.3.1.5 Operationen 12](#_Toc452380349)

[5.3.2 Package logik.terminplanung 13](#_Toc452380350)

[5.3.2.1 Beschreibung des Packages 13](#_Toc452380351)

[5.3.2.2 Diagramme 13](#_Toc452380352)

[5.3.2.3 Schnittstellen 13](#_Toc452380353)

[5.3.2.4 Operationen 14](#_Toc452380354)

[5.3.3 Package gui.terminplanung 16](#_Toc452380355)

[5.3.3.1 Beschreibung des Packages 16](#_Toc452380356)

[5.3.3.2 Diagramme 16](#_Toc452380357)

[5.3.3.3 Schnittstellen 16](#_Toc452380358)

[5.3.3.4 Operationen 16](#_Toc452380359)

[5.3.4 Package administratorverwaltung 18](#_Toc452380360)

[5.3.4.1 Beschreibung des Packages 18](#_Toc452380361)

[5.3.4.2 Diagramme 18](#_Toc452380362)

[5.3.4.3 Schnittstellen 18](#_Toc452380363)

[5.3.4.4 Aufteilung der Klassen in das Drei-Schichten-Modell: 18](#_Toc452380364)

[5.3.4.5 Operationen 19](#_Toc452380365)

[5.3.5 Package gui.dienstleistungsverwaltung 21](#_Toc452380366)

[5.3.5.1 Beschreibung des Packages 21](#_Toc452380367)

[5.3.5.2 Diagramme 21](#_Toc452380368)

[5.3.5.3 Schnittstellen 21](#_Toc452380369)

[5.3.5.4 Operationen 21](#_Toc452380370)

[5.3.6 Package logik.dienstleistungenverwaltung 23](#_Toc452380371)

[5.3.6.1 Beschreibung des Packages 23](#_Toc452380372)

[5.3.6.2 Diagramme 23](#_Toc452380373)

[5.3.6.3 Schnittstellen 23](#_Toc452380374)

[5.3.6.4 Operationen 23](#_Toc452380375)

[5.3.7 Package gui.produktverwaltung 24](#_Toc452380376)

[5.3.7.1 Beschreibung des Packages 24](#_Toc452380377)

[5.3.7.2 Diagramme 24](#_Toc452380378)

[5.3.7.3 Schnittstellen 24](#_Toc452380379)

[5.3.7.4 Operationen 25](#_Toc452380380)

[5.3.8 Package logik.produktverwaltung 26](#_Toc452380381)

[5.3.8.1 Beschreibung des Packages 26](#_Toc452380382)

[5.3.8.2 Diagramme 26](#_Toc452380383)

[5.3.8.3 Schnittstellen 26](#_Toc452380384)

[5.3.8.4 Operationen 26](#_Toc452380385)

[5.3.9 Package gui.kassenfunktion 28](#_Toc452380386)

[5.3.9.1 Beschreibung des Packages 28](#_Toc452380387)

[5.3.9.2 Diagramme 28](#_Toc452380388)

[5.3.9.3 Schnittstellen 29](#_Toc452380389)

[5.3.9.4 Operationen 29](#_Toc452380390)

[5.3.10 Package kundenverwaltung 30](#_Toc452380391)

[5.3.10.1 Beschreibung des Packages 30](#_Toc452380392)

[5.3.10.2 Diagramme 30](#_Toc452380393)

[5.3.10.3 Schnittstellen 30](#_Toc452380394)

[5.3.10.4 Aufteilung der Klassen in das Drei-Schichten-Modell: 30](#_Toc452380395)

[5.3.10.5 Operationen 31](#_Toc452380396)

[5.3.11 Package mitarbeiterverwaltung 33](#_Toc452380397)

[5.3.11.1 Beschreibung des Packages 33](#_Toc452380398)

[5.3.11.2 Diagramme 33](#_Toc452380399)

[5.3.11.3 Schnittstellen 33](#_Toc452380400)

[5.3.11.4 Aufteilung der Klassen in das Drei-Schichten-Modell: 33](#_Toc452380401)

[5.3.11.5 Operationen 34](#_Toc452380402)

[5.3.12 Package gui.verkauf 36](#_Toc452380403)

[5.3.12.1 Beschreibung des Packages 36](#_Toc452380404)

[5.3.12.2 Diagramme 36](#_Toc452380405)

[5.3.12.3 Schnittstellen 37](#_Toc452380406)

[5.3.12.4 Operationen 37](#_Toc452380407)

[5.3.13 Package logik.verkaufsverwaltung 38](#_Toc452380408)

[5.3.13.1 Beschreibung des Packages 38](#_Toc452380409)

[5.3.13.2 Diagramme 38](#_Toc452380410)

[5.3.13.3 Schnittstellen 38](#_Toc452380411)

[5.3.13.4 Operationen 39](#_Toc452380412)

[5.3.14 Package gui.trester 40](#_Toc452380413)

[5.3.14.1 Beschreibung des Packages 40](#_Toc452380414)

[5.3.14.2 Diagramme 40](#_Toc452380415)

[5.3.14.3 Schnittstellen 40](#_Toc452380416)

[5.3.14.4 Operationen 40](#_Toc452380417)

[5.3.15 Package logik.trester 41](#_Toc452380418)

[5.3.15.1 Beschreibung des Packages 41](#_Toc452380419)

[5.3.15.2 Diagramme 41](#_Toc452380420)

[5.3.15.3 Schnittstellen 41](#_Toc452380421)

[5.3.15.4 Operationen 41](#_Toc452380422)

[5.3.16 Package Persistenz 42](#_Toc452380423)

[5.3.16.1 Beschreibung des Packages 42](#_Toc452380424)

[5.3.16.2 Diagramme 42](#_Toc452380425)

[5.3.16.3 Schnittstellen 42](#_Toc452380426)

[5.3.16.4 Operationen 42](#_Toc452380427)

[5.3.17 Package Main 44](#_Toc452380428)

[5.3.17.1 Beschreibung des Packages 44](#_Toc452380429)

[5.3.17.2 Diagramme 44](#_Toc452380430)

[5.3.17.3 Schnittstellen 44](#_Toc452380431)

[5.3.17.4 Aufteilung der Klassen in das Drei-Schichten-Modell: 44](#_Toc452380432)

[5.3.17.5 Operationen 44](#_Toc452380433)

[6 Datenspeicherung (Data View) 45](#_Toc452380434)

[7 Größen und Leistung (Size and Performance) 46](#_Toc452380435)

# Einführung (Introduction)

## Definitionen und Abkürzungen (Definitions, Acronyms, Abbreviations)

Vgl. separates Glossary-Dokument im Repository, Dateiname: Glossary.docx

## Referenzen (References)

Buch: Craig Larman: UML 2 und Patterns angewendet (2005)

## Übersicht (Overview)

Im folgenden Teil dieses Dokuments gehen wir auf die architektonische Darstellung bei unserer Software ein und beschreiben, welche Ziele und Einschränkungen damit verbunden sind. Außerdem möchten wir insbesondere auf die logische Architektur (logical view) und die Datenspeicherung (data view) eingehen, da diese Views bei unserer Software mehr zum Tragen kommen als andere. Abschließend wird noch beschrieben, welche Größen und Leistungen Mosti umfassen soll.

# Architektonische Darstellung (Architectural Representation)

Ein System lässt sich durch verschiedene architektonische Sichten darstellen. So gibt es beispielsweise Sichten für Logik, Daten, Sicherheit, Implementierung oder auch für Use Cases.

In diesem Dokument möchten wir die Architektur der Mosti-Software mit folgenden Sichten beschreiben:

Logische Sicht (logical view):

Diese Sicht soll die Struktur der Software widerspiegeln. Dies betrifft hier vor allem die Implementierung. So wird beispielsweise beschrieben, wie Packages und Klassen gegliedert bzw. organisiert werden oder auch wie Klassen anhand bestimmter Funktionen interagieren können. Wir werden bei unserer Software das Drei-Schichten-Modell an, welches später genauer erläutert wird.

Daten-Sicht (data view):

Diese Sicht soll einen Überblick darüber geben, wie wir Daten erhalten und verwalten. Bei Mosti setzen wir das mithilfe einer Datenbank um. Die Software hat dabei jederzeit Zugriff auf die Daten, die in der Datenbank gespeichert sind.

*Anmerkung:*

Eine physikalische Sicht ist bei Mosti derzeit nicht relevant, weil die Software so geplant ist, dass sie von einem Anwender nur auf einem bestimmten Rechner benutzt wird und nicht mit anderen Rechnern oder Systemen kommuniziert.

# Architektonische Ziele & Einschränkungen (Architectural Goals and Constraints)

Wir haben uns für eine Drei-Schichten-Architektur entschieden, welche sich aus den Schichten Präsentation (GUI-Schicht), Logik und Datenerhaltung (Persiszenz-Schicht) zusammensetzt. Dabei sollen zwischen den Schichten nur Abhängigkeiten von oben nach unten bestehen. Durch diese Reduktion von Abhängigkeiten können Änderungen in einzelnen Schichten vorgenommen werden, ohne das gesamte System ändern zu müssen. Da wir einen iterativen Software-Entwicklungs-Prozess durchlaufen, können nahezu jederzeit Änderungen in der Implementierung gefordert sein, da sich z. B. nochmals die Anforderungen geändert haben. Es kann auch beispielsweise eine andere Präsentation der Daten gewünscht sein – dann sind nur Änderungen in der sogenannten Präsentations- bzw. GUI-Schicht erforderlich. In den anderen beiden Schichten müssen dabei keinerlei Änderungen vorgenommen werden, da diese sich unterhalb der GUI-Schicht befinden und somit nicht von dieser abhängig sind. Ein weiterer Grund, das Drei Schichten Modell anzuwenden, ist die Tatsache, dass im Mosti-Projekt u. a. Teilfunktionalitäten mit spezifischen GUIs auftauchen, an welchen unterschiedliche Teammitglieder arbeiten. Durch das Schichtenmodell ist eine bestimmte Struktur bzgl. der Implementierung vorgegeben, wodurch andere Teammitglieder einen besseren Überblick und schnelleres Verständnis für Bereiche, die sie nicht selbst implementiert haben, erlangen sollen.

# Logische Architektur (Logical View)

Bei der Entwicklung von Mosti verfolgen wir das Drei-Schichten-Modell. Dabei unterscheiden wir folgende Schichten:

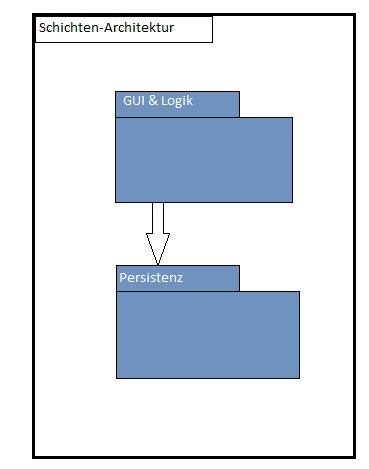
* GUI-Schicht
* Domain-Logik-Schicht
* Persistenz-Schicht (oder auch Datenerhaltungsschicht)

Dieses Modell erläutern wir im folgenden Punkt (5.1 Überblick) noch genauer.

Bezüglich der Package-Struktur streben wir generell an, ähnliche Klassen in einem Package zu gliedern (hohe Kohäsion) und andererseits, dass wenige Abhängigkeiten zwischen den Packages bestehen (geringe Kopplung).

## Übersicht (Overview)

Das Drei-Schichten-Modell für unsere Software lässt sich wie folgt darstellen:



Anmerkung: unsere Software befindet sich gerade im Umbau in das Drei-Schichten-Model.

Teilweise sind Packages so aufgeteilt, wie hier dargestellt, teilweise aber auch in dem Drei-Schichten-Model. Ziel ist die Umstrukturierung aller Packages in da unten beschriebene Drei-Schichten-Model.

Die Schicht GUI umfasst die grafische Benutzeroberfläche, also die Darstellung der Daten für den Nutzer.

Die Domain-Logik-Schicht ist in dieser Darstellung die Schnittstelle zwischen der GUI-Schicht und der Persistenz-Schicht. Darin werden die spezifischen Anforderungen bzw. Funktionalitäten umgesetzt.

In der Persistenz-Schicht wird die Datenerhaltung geregelt.

Aus dem Diagramm ist zu entnehmen, dass die Abhängigkeit zwischen den Schichten von oben nach unten verläuft. So kann die GUI-Schicht nur auf die Domain-Logik-Schicht zugreifen und die Domain-Logik-Schicht wiederum nur auf die Persistenz-Schicht. Somit kann die GUI-Schicht zum einen nicht auf die Persistenz-Schicht zugreifen. Zum anderen können Domain-Logik-Schicht und Persistenz-Schicht nicht auf die GUI-Schicht zugreifen und machen somit eine unabhängige Entwicklung möglich.

Im Folgenden werden die einzelnen Schichten noch spezifischer dargelegt.

## Design Pakete (Architecturally Significant Design Packages)

In diesem Überpunkt gehen wir nochmals auf die drei einzelnen Schichten ein, die wir hier als Design-Pakete bezeichnen. Hierbei ist zu erwähnen, dass es sich dabei nicht um die von uns erstellten Java-Packages handelt. Da es bei Mosti verschiedene Teilbereiche gibt (z. B. Lagerverwaltung oder Kundenverwaltung), wurden diese jeweils zusammenhängenden Klassen in einem Java-Package zusammengefasst und sind innerhalb dieses Teilbereiches/Java-Packages in die Schichten unterteilt. Diese spezifischen Unterteilungen führen wir im Punkt 5.3 auf.

### Package GUI

#### Beschreibung des Package

Das Package GUI umfasst die graphische Benutzeroberfläche und somit u. a. die Präsentation von Daten, die über die Domain-Logik-Schicht zur Verfügung stehen. Bei Mosti arbeitet der Benutzer selbst ausschließlich über die GUI-Schicht, um Daten zu erfassen, abzurufen oder um bestimmte Funktionen auszuführen. Somit ist es wichtig, dass in diesem Package auch Benutzereingaben erkannt und weitergeleitet werden.

#### Schnittstellen

Die GUI-Schicht hat Zugang zur Domain-Logik-Schicht. Jedoch kann von anderen Schichten nicht auf die GUI-Schicht zugegriffen werden.

### Package Domain-Logik

#### Beschreibung des Package

Das Package Domain-Logik umfasst die Logik, welche hinter den Funktionalitäten der Software liegt. Somit ist darin geregelt, wie bestimmte Daten verarbeitet werden. Bezogen auf Mosti ist beispielsweise die Funktionalität, welche die Kassenfunktion betrifft, von hoher Bedeutung.

#### Schnittstellen

Die Domain-Logik-Schicht hat Zugang auf die darunterliegende Persistenz-Schicht. Auf die Domain-Logik-Schicht zugreifen kann hingegen nur die GUI-Schicht.

#### Operationen

### Package Persistenz

#### Beschreibung des Package

Im Package Persistenz wird die Datenerhaltung geregelt. Da wir zur Speicherung von Daten eine Datenbank benutzen, soll dieses Package ermöglichen, dass eine Verbindung zur Datenbank aufgebaut wird, man auf die Daten zugreifen und Datenänderungen tätigen kann.

#### Schnittstellen

Die Schnittstelle zwischen unserer Software und der Datenbank wird innerhalb der Persistenz-Schicht gewährleistet. Zugang auf die Persistenz-Schicht hat die Domain-Logik-Schicht.

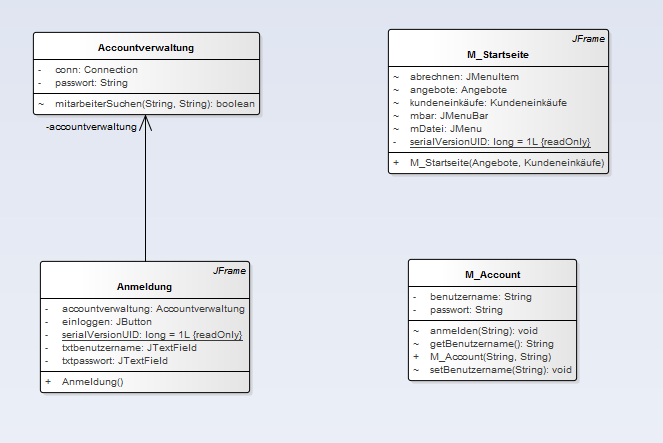
## Java-Pakete und deren Architektur-Design

### Package account

#### Beschreibung des Packages

Das Package *account* umfasst die Aspekte, die den Mitarbeiter-Account betreffen, also dessen Implementierung sowie die Funktion zur Anmeldung mit Passwort inklusive der entsprechenden GUI. Des Weiteren wird in diesem Package in der Persistenzschicht eine Verbindung zur Datenbank aufgebaut, in welcher die Account-Daten abgespeichert sind. Darüber hinaus ist die Startseite-GUI in diesem Package enthalten, da durch die Anmeldung erst Zugriffe auf bestimmte Funktionen auf der Startseite möglich werden.

#### Diagramme

****

#### Schnittstellen

Das Package *account* besitzt Schnittstellen zum Package *mitarbeiterverwaltung*, da mit Anlegen eines Mitarbeiters auch sogleich ein Account für diesen erstellt wird. Des Weiteren besteht derzeit noch eine Schnittstelle zum Package *main*, in welchem die Startseite aufgerufen wird.

#### Aufteilung der Klassen in das Drei-Schichten-Modell:

GUI-Schicht: M\_Startseite, Anmeldung

Logik-Schicht: M\_Account

Persistenz-Schicht: Accountverwaltung

#### Operationen

Accountverwaltung:

* package boolean mitarbeiterSuchen(String,String)

Anmeldung:

* Konstruktor: public Anmeldung()
* Innere Klasse: ActionListener
  + public void actionPerformed(ActionEvent)

M\_Account:

* Konstruktor: public M\_Account(String,String)
* package void anmelden(string)
* package String getBenutzername()
* package void setBenutzername(String)

M\_Startseite:

* Konstruktor: public M\_Startseite(Angebote, Kundeneinkäufe)
* Innere Klasse: ActionListener
  + public void actionPerformed(ActionEvent)
* Innere Klasse: ActionListener
  + public void actionPerformed(ActionEvent)
* Innere Klasse: ActionListener
  + Public void actionPerformed(ActionEvent)
* Innere Klasse: ActionListener
  + Public void actionPerformed(ActionEvent)
* Innere Klasse: ActionListener
  + Public void actionPerformed(ActionEvent)
* Innere Klasse: ActionListener
  + Public void actionPerformed(ActionEvent)
* Innere Klasse: ActionListener
  + Public void actionPerformed(ActionEvent)

### Package logik.terminplanung

#### Beschreibung des Packages

Das Package *terminplanung* umfasst alle Aspekte der Planung und Speicherung von Terminen.Die Hier aufgeführten Klassen managen den Umgang mit den aus der Datenbank herausgelesenen Termindaten und stellen sie den GUI-Klassen zur Anzeige bereit.

#### Diagramme

#### Schnittstellen

Das Package *logik.terminplanung* besitzt Schnittstellen zum Package *kassenfunktion*, von welcher aus die Kundentermine mit den tatsächlichen Termindurchführung verknüpft werden. Eine weitere Schnittstelle existiert zum Package *account*, welches die Startseite der Software beinhaltet und den Benutzer zwischen den einzelnen Modulen navigiert. Zur Verwaltung der Termine existier des Weitern eine Schnittstelle *zu gui.terminplanung* und *persistenz.*

#### Operationen

Intervall:

* public int getEnde()
* public void setEnde(int)
* public int getStart()
* public void setStart()

Konfigurationswerte:

* Konstruktor: public Konfigurationswerte()
* package int getZeitslot()
* package void setZeitslot(int)
* package int getArbeitsbeginn()
* package void setArbeitsbeginn(int)
* package int getArbeitsende()
* package void setArbeitsende(int)
* package int getAufteilung()
* package void setAufteilung(int)
* package int getZeilenanzahlProSeite()
* package void setZeilenanzahlProSeite(int)

TagFrameController:

* Konstruktor: public TagFrameController()
* public int berechneAnzahlZeitslots(int)
* public int getZeile(int, int)
* public int anzahlAlleTermine()
* public boolean istTerminFrei(ArrayList<Termin>)
* public void termineSpeichern(int, int, Date, int)
* public ArrayList<Termin> termineLaden(Date)
* public boolean isFrueherEnabled(int)
* public boolean isSpaeterEnabled(int)

Termin:

* public int getKundenId()
* public void setKundenId(int)
* public int getTerminId()
* public void setTerminId(int)
* public Date getDatum()
* public void setDatum(Date)
* public int getAnzahlZeitslots()
* public void setAnzahlZeitslots(int)
* public int getUhrzeit()
* public void setUhrzeit(int)

TermineTableModel:

* Konstruktor: public TermineTableModel(ArrayList<Termin>, int)
* public String getColumnName(int)
* public int getColumnCount()
* public int getRowCount()
* public Object getValueAt(int,int)
* package ArrayList<Termin> getAlleTermine()
* public void erhoeheAnzeigeseite()
* public void erbiedrigeAnzeigeSeite()

TerminHinzufügenLogik:

* Konstruktor: public TerminHinzufügenLogik()
* public long nächstenTagBerechenen(long)
* public long vorherigenTagBerechnen(long)
* public ArrayList<Termin> freieTermineSuchen(long)
* public int BerechneAnzeigeSeite(int)
* public int berechneTermindauer(String s)
* public long formatieren(String)

TerminLogik:

* Konstruktor: public TerminLogik()
* private ArrayList<Termin> inVolleTerminListe(ArrayList<Termin>)
* private ArryList<Termin> termineSortieren(ArrayList<Termin>)
* package ArrayList<Termin> termineLaden(long)
* package int berechneAnzeigeSeite(int)
* public String termineNachUhrzeit(int)
* public ArrayList<Intervall> berechneLücken(ArrayList<Termin>)
* public int getZeilenAnzahlProSeite()
* public ArrayList<Integer> kundenIdLaden(String)
* public String kundenNamenLaden(int)
* package void termineSpeichern(int, int, Date, int)
* public int getZeitslot()

### Package gui.terminplanung

#### Beschreibung des Packages

Das Package *gui.terminplanung* enthält alle Klassen, die zur Anzeige der Terminplanung benötigt werden.

#### Diagramme

#### Schnittstellen

Das Package *gui.terminplanung* besitzt lediglich eine Schnittstelle zum Package *logik.terminplanung,* da hier die Informationen über die darzustellenden Werte enthalten sind.

#### Operationen

FreieTermineTableModel:

* Konstruktor: public FreieTermineTableModel(ArrayList<Termin>)
* public String getColumnName(int)
* public int getColumnCount()
* public int getRowCount()
* public Object getValueAt(int, int)

KundenNameCellRenderer:

* Konstruktor public KundenNameCellRenderer()
* public Component getTableCellRendererComponent(JTable, Object, boolean, boolean, int, int)

TagFrame:

* Konstruktor TagFrame(Date, int, TerminHinzufügenFrame)
* innere Klasse: MyMouseListener
* public void mousePressed(MouseEvent)
* innerer Klasse: MySpäterHandler
* public void actionperformed(ActionEvent)
* innerer Klasse: MyFüherHandler
* public void actionperformed(ActionEvent)

TermineCellRenderer:

* Konstruktor: public TermineCellRenderer()
* public Component getTableCellRendererComponent(JTable, Object, boolean, boolean, int, int)

TerminErstellenDialog:

* Konstruktor: TerminErstellenDialog(int. Date, ArrayList<Termin>, String)
* innere Klasse: MyOkListener
* public void actionPerformed(ActionEvent)
* innere Klasse: MyKeyListener
* public void keyPressed(KeyEvent)

TerminHinzufügenFrame:

* Konstruktor: public TerminHinzufügenFrame()
* innere Klasse: MyBerechnenHandler
* public void actionPerformed(ActionEvent)
* innere Klasse: MyMouseListener
* public void MousePressed(MouseEvent)
* innere Klasse: MyFrüherHandler
* public void actionPerformed(ActionEvent)
* innere Klasse: MySpäterListener
* public void MousePressed(MouseEvent)

TerminplanungsFrame:

* Konstruktor: public TerminplanungsFrame()
* innere Klasse: ActionListener
* public void actionPerformed(ActionEvent)

### Package administratorverwaltung

#### Beschreibung des Packages

Das Package *administratorverwaltung* enthält die Klassen, die die administratorspezifischen Funktionalitäten zur Verfügung stellen (z.B. Schichtplanung)

#### Diagramme

#### Schnittstellen

Schnittstellen existieren zum momentanen Zeitpunkt keine zu anderen Schnittstellen, jedoch soll später eine Schnittstelle zum Package *account* erstellt werden, um vom der Startseite des Administratorsauf die Schichtplanverwaltung zugreifen zu können.

#### Aufteilung der Klassen in das Drei-Schichten-Modell:

GUI-Schicht:SchichtHinzufügenFrame, SchichtTageFrame, MitarbeiterNameCellRenderer, SchichtTableModel, WochenübersichtTableModel, innere Klassen zur Verarbeitung von Benutzereingaben

Logik-Schicht: SchichtLogik, Konfigurationswerte, Schicht

Persistenz-Schicht: SchichtplanDB

#### Operationen

Konfigurationswerte:

* Konstruktor: public Konfigurationswerte()
* int getMitarbeiterProSchicht()
* void setMitarbeiterProSchicht()
* public int getSchichtenProTag()
* public void setSchichtenProTag()
* public int getSchichtDauer()
* public void setSchichtDauer(int)
* public int getArbeitsbeginn()
* public void setArbeitsbeginn(int)
* public int getArbeitsende()
* public void setArbeitsende(int)

MitarbeiterNameCellRenderer:

* Konstruktor: public MitarbeiterNameCellRenderer()
* public Component getTableCellRenderer(JTable, Object, bolean, boolean, int, int)

Schicht:

* Konstrktor: public Schicht()
* public Integer getMitarbeiterId(int)
* public void addMitarbeiter(int)
* package int getAnzahlMitarbeiter()
* public int getSchichtId()
* public void steSchichtId(int)
* public int getUhrzeit()
* public void setUhrzeit(int)
* public Date getDatum()
* public void setDatum(Date)

SchichtHinzufügenFrame:

* Komstruktor: package SchichtHinzufügenFrame(long)
* innere Klasse: MyKeyListener:
* public void keyTyped(keyEvent)
* innere Klasse: MyOkListener
* public void actionPerformed(ActionEvent e)

SchichtLogik:

* Konstruktor: public SchichtLogik()
* public ArrayList<Schicht> schichtLaden(long)
* package void schichtSpeichern(Date, int, int)
* public ArrayList<Schicht> schichtenSortieren(ArrayList<Schicht>)
* public ArrayList<Schicht> schichtenMergen(ArrayList<Schicht>)
* public ArrayList<Integer> mitarbeiterIdLaden(String)
* public String mitarbeiternameLaden(int)
* package int getMitarbeiterProSchicht()
* package int getSchichtenProTag()
* public int berechneUhrzeit(int)

SchichtplanDb:

* package ArrayList<Schicht> schichtLaden(Date)
* package void schichtSpeichern(Date, int, int)
* package String mitarbeiterNamenLaden(int)
* package ArrayList<Integer> adminwerteLaden()
* package ArrayList<Integer> mitarbeiterIdLaden(String)

SchichtplanungsFrame:

* innere Klassen: ActionListener:
* public void actionPerformed(ActionEvent e)

SchichtTableModel:

* Konstruktor: package SchichtTableModel(ArrayList<Schicht>)
* public String getColumnName(int)
* public int getColumnCount()
* public int getRowCount()
* public Object getValueAt(int, int)

SchichtTagFrame:

* Konstruktor: public SchichtTagFrame(Date)

WochenübersichtTableModel:

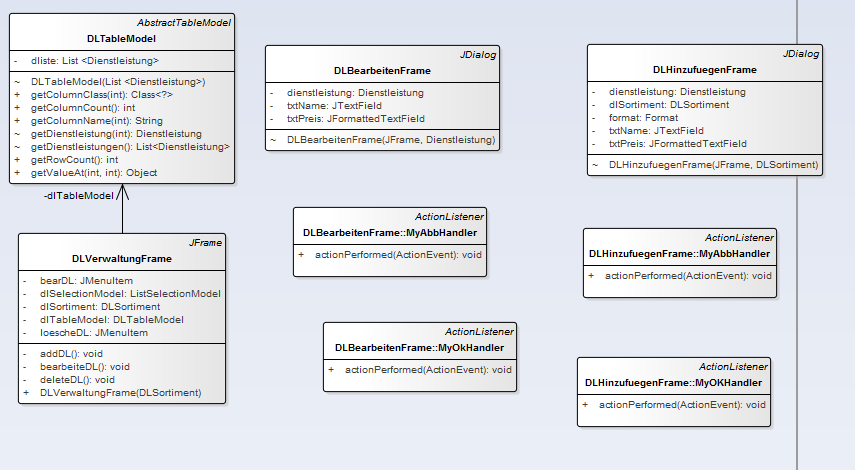
* Konstruktor: public WochenübersichtTableModel(ArrayList<Schicht>)
* public in getColumnCount()
* public int getRowCount()
* public Object getValueAt(int, int)

### Package gui.dienstleistungsverwaltung

#### Beschreibung des Packages

Dieses Package enthält alle Klassem zur Darstellung der Dienstleistungsverwaltung. Die darzustellenden Informationen erhält dieses Package vom Package *logik.dienstleistung.*

#### Diagramme



#### Schnittstellen

Die wichtigste Schnittstelle des *gui.dienstleistung*-Packages besteht zum *logik.dienstleistung,* um die Informationen über die darstellenden Werte zu erhalten.

#### Operationen

DLBearbeitenFrame:

* Konstruktor: package DLBearbeitenFrame(JFrame, Dienstleistung)
* innere Klasse: MyOkHandler
* public void actionPerformed(ActionEvent e)
* innere Klasse: MyAbbHandler
* public void actionPerformed(ActionEvent e)

DLHinzufügenFrame:

* Konstruktor: package DLHinzufügenFrame(JFrame, DLSortiment)
* innere Klasse: MyOkHandler
* public void actionPerformed(ActionEvent e)
* innere Klasse: MyAbbHandler
* public void actionPerformed(ActionEvent e)

DLTableModel:

* Konstruktor: package DLTableModel(List<Diestleistung>)
* public int getColumnCount()
* public int getRowCount()
* public Object getValueAt(int, int)
* public String getColumnName(int)
* public Class<?> getColumnClass(int)
* package Dinestleistung getDienstleistung(int)
* package List<Dienstleistung> getDienstleistungen()

DLVerwaltungsFrame:

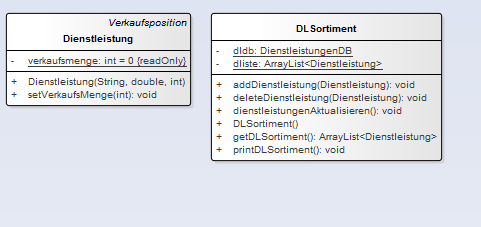
* Konstruktor: public DLVerwaltungFrame(DLSortiment)
* innere Klasse: MyAbbHandler
* public void actionPerformed(ActionEvent e)
* private void addDL()
* private void deleteDL()
* private void bearbeiteDL()

### Package logik.dienstleistungenverwaltung

#### Beschreibung des Packages

Das Package *logik.dienstleistungenverwaltung* umfasst alle Aspekte zur Erstellung und Bearbeitung, sowie zum Löschen von einzelnen Dienstleistungen. Das Package bearbeitet die von der Datenbank geladenen Daten und stellt sie dem Package *gui.dienstleistung* zur Verfügung.

#### Diagramme



#### Schnittstellen

Das Package *logik*.*dienstleistungenverwaltung* besitzt Schnittstellen zum Package *kassenfunktion*, in welcher die vorhandenen Dienstleistungen zum Verkauf präsentiert werden. Eine weitere Schnittstelle existiert zum Package *account*, welches die Startseite der Software beinhaltet. Auf der Startseite kann der Benutzer dann die Verwaltung der Dienstleistungen aufrufen. Außerdem existieren Schnittstellen zu den Packages *gui.dienstleistungsverwaltung*  und *persistenz*, um die Dienstleistunge zu speichern und zu präsentieren.

#### Operationen

Dienstleistung:

* Konstruktor: public Dienstleistung(String, double, int)
* public void setVerkaufsMenge(int)

DLSortiment:

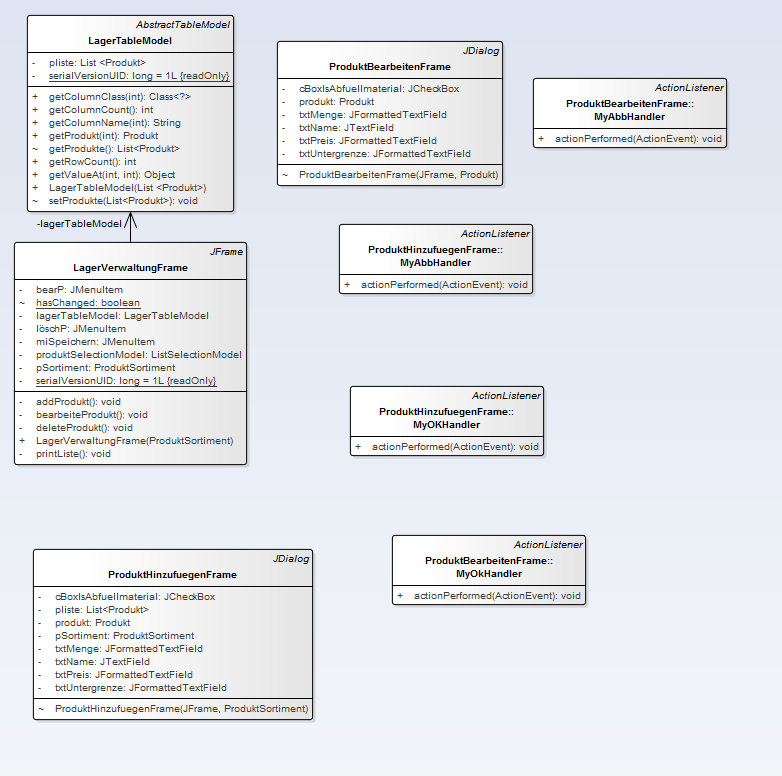
* Konstruktor: public DLSortiment()
* public void addDienstleistung(Dienstleistung)
* publich void dienstleistungenAktualisieren()
* public void deleteDienstleistung(Dienstleistung)
* public ArrayList< Dienstleistung> getDLSortiment()
* public void printDLSortiment()

### Package gui.produktverwaltung

#### Beschreibung des Packages

Das Package *gui.produktverwaltung* umfasst alle Aspekte zur Präsentation von Lager und angelegter Produkt sowie Frames zum Hinzufügen bzw. Bearbeiten von Produkten.

#### Diagramme



#### Schnittstellen

Das Package *gui.produktverwaltung* hat die Schnittstellen zu *logik.produktverwaltung\*, kundenverwaltung.NullableFormatter*.

#### Operationen

ProduktHinzufuegenFrame:

* Konstruktor: package ProduktHinzufuegenFrame(JFrame, List<Produkt>,List<Produkt>)
* innere Klasse: MyOKHandler
  + public void actionPerformed(ActionEvent)
* innere Klasse: MyAbbHandler
  + public void actionPerformed(ActionEvent)

ProduktBearbeitenFrame:

* Konstruktor: package ProduktBearbeitenFrame(JFrame, Produkt)
* innere Klasse: MyOKHandler
  + public void actionPerformed(ActionEvent)
* innere Klasse: MyAbbHandler
  + public void actionPerformed(ActionEvent)

LagerVerwaltungFrame:

* Konstruktor: public LagerVerwaltungFrame(List<Produkt>, List<Produkt>)
* innere Klasse: ActionListenerActionListener()
* public void actionPerformed(ActionEvent)
* innere Klasse: SelectionListener(ListSelectionListener())
  + public void valueChanged(ListSelectionEvent)
* innere Klasse: addMouseListener(MouseAdapter())
  + public void MousePressed(MouseEvent)
* private void addProdukt()
* private void deleteProdukt()
* private void bearbeiteProdukt()
* private void printListe()

LagerTableModel:

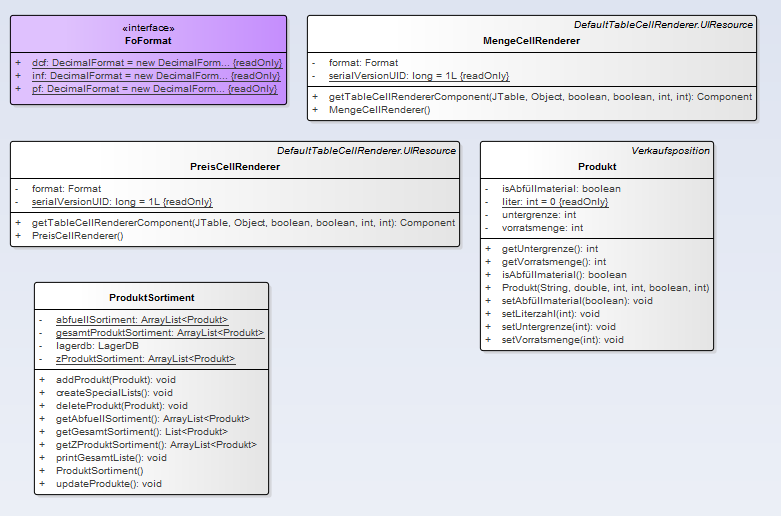
* Konstruktor: package LagerTableModel(List<Produkt>, List<Produkt>)
* public int getColumnCount()
* public String getColumnName(int)
* public int getRowCount()
* public Object getValueAt(int, int)
* public Class<?> getColumnClass(int)
* public Produkt getProdukt(int)
* public List<Produkt> getZusatzProdukte()
* public void setZusatzProdukte(List<Produkt>)
* public void setAbfuellProdukte(List<Produkt>)

### Package logik.produktverwaltung

#### Beschreibung des Packages

Das Package logik.produktverwaltung umfasst alle Aspekte der Logischen Ausführungen.

#### Diagramme



#### Schnittstellen

Das Package logik.produktverwaltung besitzt Schnittstellen zu logik.verkaufsverwlatung.Verkaufsposition und persistenz.LagerDB

#### Operationen

Interface FoFormat

MengeCellRenderer:

* Konstruktor: public MengeCellRenderer()
* public Component getTableCellRendererComponent(JTable, Object, boolean, boolean, int, int)

PreisCellRenderer:

* Konstruktor: public PreisCellRenderer()
* Public Component getTableCellRendererComponent(JTable, Object, boolean, boolean, int, int)

Produkt:

* Konstruktor: public Produkt(String, double, int, int, boolean, int)
* public int getVorratsmenge()
* public void setVorratsmenge(int)
* public int getUntergrenze()
* public void setUntergrenze(int)
* public boolean isAbfüllmaterial()
* public void setAbfüllmaterial(Boolean)

ProduktSortiment:

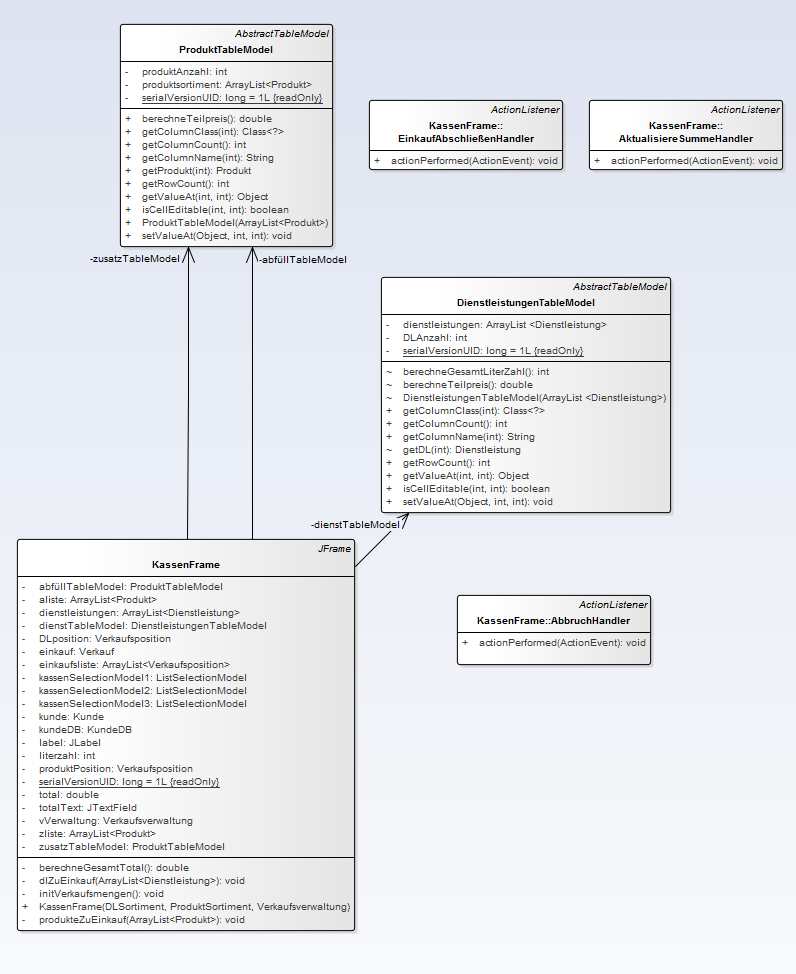
* Konstruktor: public ProduktSortiment()
* Public void createSpecialLists()
* Public void addProdukt(Produkt p)
* Public void deleteProdukt(Produkt p)
* Public List<Produkt> getGesamtSortiment()
* Public ArrayList<Prudikt> getZProduktSortiment()
* Public ArrayList<Produkt> getAbfuellSortiment()
* Public void printGesamtListe()

### Package gui.kassenfunktion

#### Beschreibung des Packages

Das Package *gui.kassenfunktion* umfasst alle Aspekte zur Präsentation von der Kasse, des Produkts und der Dienstleistung

#### Diagramme



#### Schnittstellen

Das Package gui.kassenfunktion hat die Schnittstellen logik.dienstleistung\*, kundenverwaltung.Kunde/KundeDB und logik.produktverwaltung.Produkt.

#### Operationen

DienstleistungenTableModel:

* Konstruktor: DienstleistungenTableModel(ArrayList <Dienstleistung>)
* Public String getColumName(int)
* Public int getRowCount()
* Public int getColumnCount
* Public Object getValueAt(int, int)
* Public void setValueAt(object,int,int)
* Public class<?> getColumnClass(int
* Public Boolean isCellEditable(int,int)
* Double berechneTeilpreis()
* Int berechneGesamtLiterZahl()

KasseFrame:

* Konstruktor: KasseFrame(DLSortiment, ProduktSortiment, Verkaufsverwaltung)
* Private class AbbruchHandler
* Innere Klasse: public void actionPerformed(ActionEvent e)
* Private classAktualisiereSummeHandler
* Innere Klasse: public void actionPerformed(ActionEvent e)
* Private double berechneGesamtTotal()
* Private void initVerkaufsmengen()
* Private class EinkaufsAbschließenHandler
* Innere Klasse: public void actionPerformed(ActionEvent)
* Private void produkteZuEinkauf(ArrayList<Produkt>)
* Private void dlZuEinkauf(ArrayList<Dienstleistung>)

ProduktTableModel:

* Konstruktor: ProduktTableModel(ArrayList<Produkt>)
* Public class<?> getColumnClass(int)
* Public boolean isCellEditable(int, int)
* Public String getColumnName(int)
* Public int getRowCount()
* Publick int getColumnCount()
* Public Object getValueAt(int, int)
* Public void setValueAt(Object, int, int)
* Public Produkt getProdukt(int)
* Public double berechneTeilpreis()

### Package kundenverwaltung

#### Beschreibung des Packages

Das Package k*undenverwaltung* beinhaltet alle Klassen zur Verwaltung eines Kunden. Es enthält Klassen, die zur Ansicht der Kundenliste dienen.

#### Diagramme

#### Schnittstellen

Um Kunden mit einzelnen Termin verknüpfen zu können und um die Einkäufe mit den Kunden zu verbinden, besitzt das Package *kundenverwaltung*  eine Schnittstelle sowohl zu *terminplanung*  als auch zu *kassenfunktion.*

#### Aufteilung der Klassen in das Drei-Schichten-Modell:

GUI-Schicht:KundeBearbeitenDialog, Kundenverwaltung

Logik-Schicht: KundeTableModel, Kunde, NullableFormatter, Formats, Geschlecht,

Persistenz-Schicht: KundenDB

#### Operationen

Kunde:

* Konstruktor: public Kunde(String, String, String, String, String, String, int)
* Konstruktor: public Kunde()
* package String getNachname()
* package void setNachname(String)
* package String getVorname()
* package void setVorname(String)
* package String getStrasse()
* package vois setStrasse(String)
* package String getPlz()
* package void setPlz(String)
* package String getWohnort()
* package void setWohnort(String)
* package String getTel()
* package void setTel(String)
* package int getKundenID()
* package void setKundenID(int)

KundeBearbeitenDialog:

* Konstruktor: public KundeBearbeitenDialog(JFrame, Kunde)
* innere Klasse: MyOkHandler
* public void actionPerformed(ActionEvent)
* private String beautify(String)
* innere Klasse: MyCancelHandler
* public void actionPerformed(ActionEvent)

KundenDB:

* package ArrayList<Kunde> kundenLaden()
* package void kundenSpeichern(ArrayList<Kunde>)
* package void kundeEinfügen(Kunde)
* package void kundeLöschen(int)

KundenHinzufügenFrame:

* Konstruktor: KundeHinzufügenFrame(Jframe,List<Kunde>)
* Innere Klasse: ActionListener
* public void actionPerformed(ActionEvent)

Kundenverwaltung:

* Konstruktor: public KundenVerwaltung()
* innere Klasse: ActionListener
* public void actionPerformed(ActionEvent)
* innere Klasse: ListSelectionListener
* public void valueChanged(ListSelectionEvent)
* innere Klasse: MouseAdapter
* public void mousePressed(MouseEvent)
* private void editKunde()

KundenTableModel:

* public Class<?> getColumnClass(int)
* public String getColumnName(int)
* public int getRowCount()
* public int getColumnCount()
* public Object getValueAt(int,int)
* package ArrayList<Kunde> getKunden()
* package void setKunden(ArrayList<Kunde>)
* package Kunde getKunde(int)
* package deletKunde(Kunde)

NulltableFormatter

* Konstruktor: public NullableFormatter(JFormattedField.AbstractFormatter)
* public void install(JFormattedTextField)
* public void uninstall()
* public Object stringToValue(String)
* public String valueToString(Object)

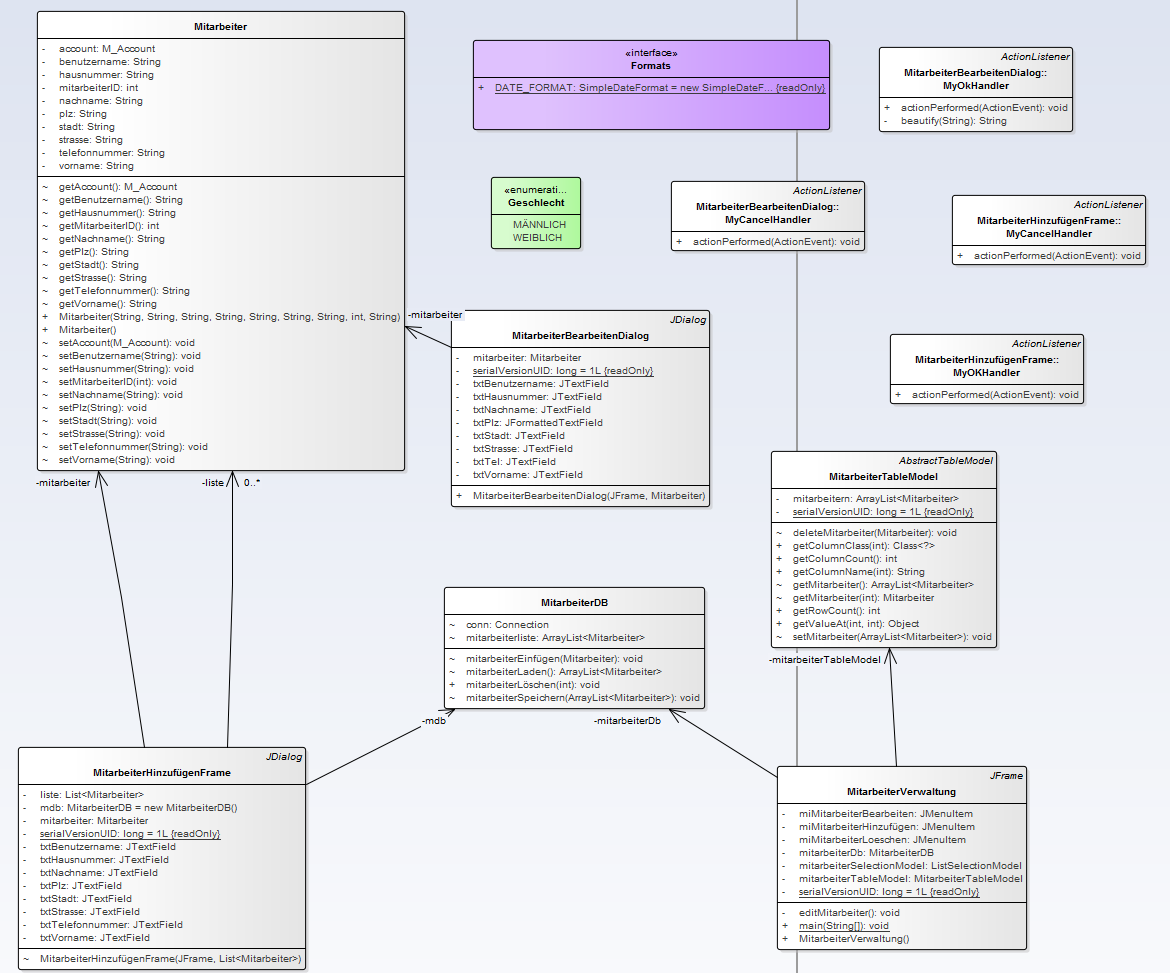
### Package mitarbeiterverwaltung

#### Beschreibung des Packages

Das Package Mitarbeiterverwaltung beinhaltet alle Verwaltungsmöglichkeiten des Mitarbeiters. Diese wären den Mitarbeiter hinzufügen, bearbeiten oder löschen. Das Package enthält ebenfalls Klassen, die zur Ansicht der Mitarbeiterliste dienen und auch Klassen zum Management des Datenbankzugriffs.

.

#### Diagramme

****

#### Schnittstellen

Eine Schnittstelle der *mitarbeiterverwaltung* besteht zum *account,* da jeder Account zu einem Mitarbeiter gehören muss.

#### Aufteilung der Klassen in das Drei-Schichten-Modell:

GUI-Schicht: MitarbeiterBearbeitenDialog, MitarbeiterVerwaltung

Logik-Schicht: MitarbeiterTableModel, Mitarbeiter, Formats, Geschlecht

Persistenz-Schicht:MitarbeiterDB

#### Operationen

Mitarbeiter:

* Konstruktor: public Mitarbeiter(String, String, String, String, String, String, String, int, String)
* Konstruktor: public Mitarbeiter()
* package String getNachname()
* package void setNachname(String)
* package String getVorname()
* package void setVorname(String)
* package String getStrasse()
* package vois setStrasse(String)
* package String getHausnummer()
* package void setHausnummer(String)
* package String getPlz()
* package void setPlz(String)
* package String getStadt()
* package void setStadt(String)
* package String getTelefonnummer()
* package void setTelefonnummer(String)
* package int getKundenID()
* package void setKundenID(int)
* package String getMitarbeiterID()
* package void setMitarbeiterID(int)
* package String getBenutzername()
* package void setBenuttzername(String)
* package M\_Account getAccout()
* package void setAccount(M\_Account)

MitarbeiterBearbeitenDialog

* Konstruktor: public MitarbeiterBearbeitenDialog(Jframe, Mitarbeiter)
* innere Klasse: MyOkHandler
* public void actionPerformed(ActionEvent)
* private String beautify(String)
* innere Klasse: MyCancelHandler
* public void actionPerformed(ActionEvent)

MitarbeiterDB:

* package ArrayList<Mitarbeiter> mitarbeiterLaden()
* package void mitarbeiterSpeichern(ArrayList<Mitarbeiter>)
* package mitarbeiterEinfügen(Mitarbeiter)
* package mitarbeiterLöschen(int)

MitarbeiterTableModel:

* public String getColumnName(int)
* public Class<?> getColumnClass(int)
* public int getRowCount()
* public int getColumnCount()
* public Object getValueAt(int,int)
* package ArrayList<Mitarbeiter> getMitarbeiter()
* package void setMitarbetier(ArrayList<Mitarbeiter>)
* package Mitarbeiter getMitarbeiter(int)
* package deleteMitarbeiter(Mitarbeiter)

MitarbeiterHinzufügenFrame:

* Konstruktor: MitarbeiterHinzufügenFrame(Jframe, List<Mitarbeiter>)
* Innere Klasse: ActionListener
* Public void actionPerformed(ActionEvent)

MitarbeiterVerwaltung:

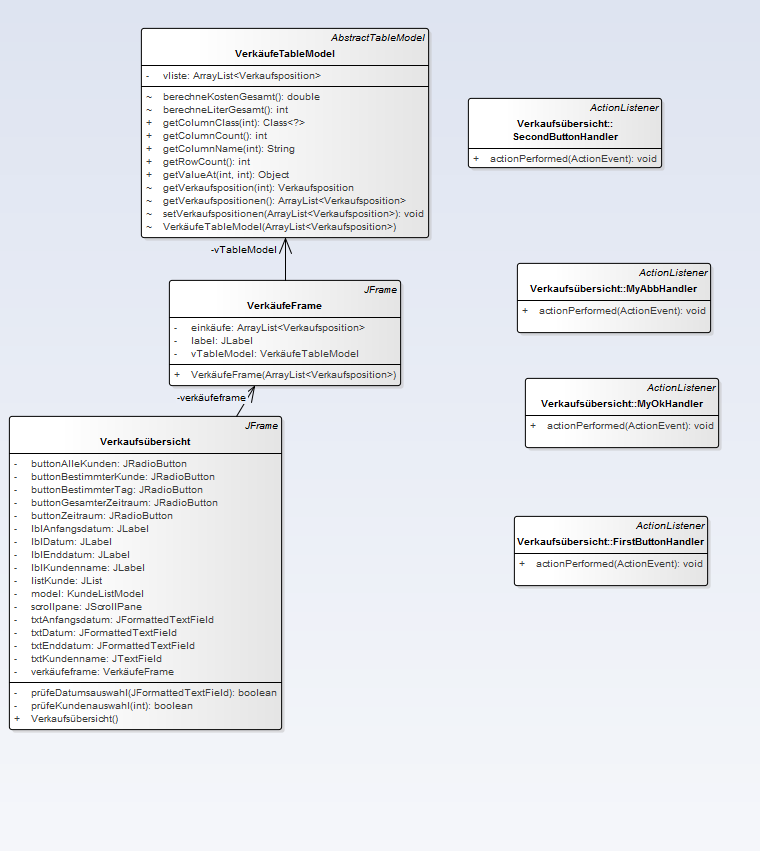
* Konstruktor: public MitarbeiterVerwaltung()
* innere Klasse: ActionListener
* public void actionPerformed(ActionEvent)
* innere Klasse: ListSelectionListener
* public void valueChanged(List SelectionEvent)
* innere Klasse: MouseAdapter
* public void mousePressed(MouseEvent)
* private void editMitarbeiter()

### Package gui.verkauf

#### Beschreibung des Packages

Das Package *gui.verkauf* enthält Klassen zur Darstellung von Verkäufen, welche sich jeweils aus einzelnen Verkaufspositionen zusammensetzen. Außerdem ist eine Klasse vorhanden, mit welcher der Benutzer diverse Bedingungen zu dieser Auflistung festlegen kann (z. B. bestimmter Kunde, bestimmter Zeitraum).

#### Diagramme

****

#### Schnittstellen

Das Package *gui.verkauf* besitzt Schnittstellen zum Package logik.*verkaufsverwaltung,* in welchem beispielsweise definiert ist, was eine Verkaufsposition ist.

#### Operationen

VerkäufeFrame:

* Konstruktor: public VerkäufeFrame(ArrayList<Verkaufsposition>)

VerkäufeTableModel:

* Konstruktor: package VerkäufeTableModel(ArrayList<Verkaufsposition>)
* public String getColumnName(int)
* public int getColumnCount()
* public int getRowCount()
* public Object getValueAt(int,int)
* public Class<?> getColumnClass(int)
* package Verkaufsposition getVerkaufsposition(int)
* package ArrayList<Verkaufsposition> getVerkaufspositionen()
* package double berechneKostenGesamt()
* package int berechneLiterGesamt()

Verkaufsübersicht:

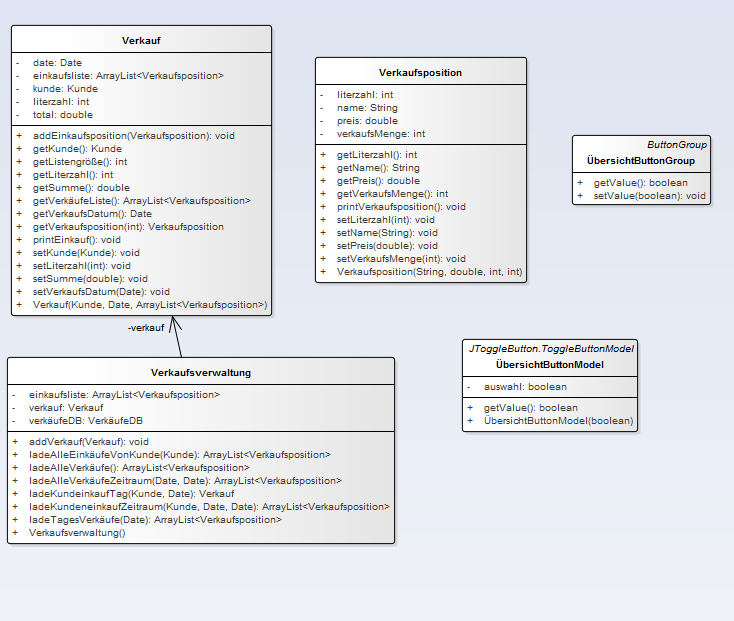
* Konstruktor: public Verkaufsübersicht()
* private boolean prüfeKundenauswahl(int)
* private boolean prüfeDatumsauswahl(int)
* innere Klasse: private class FirstButtonHandler
* public void actionPerformed(ActionEvent)
* innere Klasse: private class SecondButtonHandler
* public void actionPerformed(ActionEvent)
* innere Klasse: private class MyOkHandler
* public void actionPerformed(ActionEvent)
* innere Klasse: private class MyAbbHandler
* public void actionPerformed(ActionEvent)

### Package logik.verkaufsverwaltung

#### Beschreibung des Packages

Das Package *logik.verkaufsverwaltung* enthält Klassen, um einen Verkauf zu definieren, welcher aus u. a. aus mehreren Verkaufspositionen aufgebaut ist. Außerdem besitzt es eine Klasse, welche Funktionen der Datenbank, welche die Verkäufe verwaltet, aufruft.

#### Diagramme



#### Schnittstellen

Das Package *logik.verkaufsverwaltung* besitzt Schnittstellen zum Package *Persistenz,* in welchem der Zugriff auf die Datenbank, in welcher die Verkäufe verwaltet werden, geregelt ist. Außerdem besteht eine Schnittstellen zum Package *kundenverwaltung*.

#### Operationen

Verkaufsposition:

* Konstruktor: public Verkaufsposition(String, double, int, int)
* public String getName()
* public void setName(String)
* public double getPreis()
* public void setPreis(double)
* public int getVerkaufsMenge()
* public void setVerkaufsMenge(int)
* public int getLiterzahl()
* public void setLiterzahl(int)

Verkauf:

* Konstruktor: public Verkauf (Kunde, Date, ArrayList<Verkaufsposition>)
* public void addEinkaufsposition(Verkaufsposition)
* public getVerkaufsposition(int)
* public ArrayList<Verkaufsposition> getVerkäufeListe()
* public int getListengröße()
* public int getLiterzahl()
* public void setLiterzahl()
* public Date getVerkaufsDatum()
* public void setVerkaufsDatum(Date)
* public Kunde getKunde()
* public void setKunde(Kunde)

Verkaufsverwaltung:

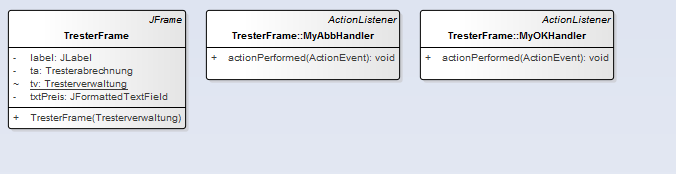
* Konstruktor: public Verkaufsverwaltung()
* public void addVerkauf(Verkauf)
* public Verkauf ladeKundeinkaufTag(Kunde, Date)
* public ArrayList<Verkaufsposition> ladeKundeneinkaufZeitraum(Kunde, Date, Date)
* public ArrayList<Verkaufsposition> ladeAlleEinkäufeVonKunde(Kunde)
* public ArrayList<Verkaufsposition> ladeTagesVerkäufe(Date)
* public ArrayList<Verkaufsposition> ladeAlleVerkäufeZeitraum(Date, Date)
* public ArrayList<Verkaufsposition> ladeAlleVerkäufe()

### Package gui.trester

#### Beschreibung des Packages

Das Package *gui.trester* enthält bisher eine Klasse, in welcher der Benutzer den Preis für den Trester einsehen und verwaltet kann.

#### Diagramme



#### Schnittstellen

Es besteht eine Schnittstelle zum Package *logik.trester*, in welcher die Logik der Tresterverwaltung bzw. –abrechnung enthält.

#### Operationen

TresterFrame:

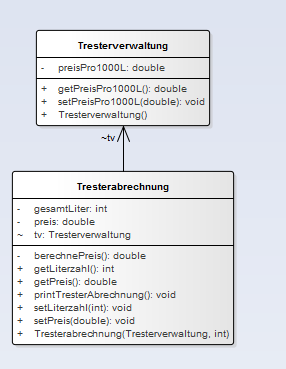
* public TresterFrame(Tresterverwaltung){
* innere Klasse: private class MyOKHandler
  + public void actionPerformed(ActionEvent)
* innere Klasse: private class MyAbbHandler
  + public void actionPerformed(ActionEvent)

### Package logik.trester

#### Beschreibung des Packages

Das Package *logik.trester* enthält die Logik zur Verwaltung des Tresterpreises, sowie zur Erstellung einer Tresterabrechnung.

#### Diagramme



#### Schnittstellen

Es bestehen bisher noch keine Schnittstellen zu anderen Packages. Jedoch muss noch eine Verbindung zur Datenbank aufgebaut werden.

#### Operationen

Tresterverwaltung:

* Konstruktor: public Tresterverwaltung ()
* public double getPreisPro1000L()
* public void setPreisPro1000L(double)

Tresterabrechnung:

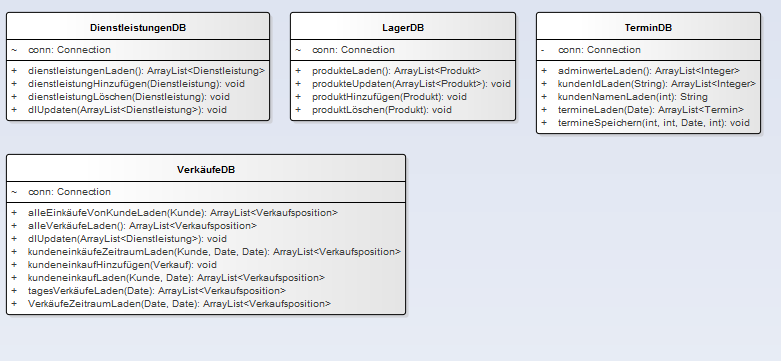
* Konstruktor: public Tresterabrechnung(Tresterverwaltung, int)
* public double getPreis()
* public void setPreis(double)
* private double berechnePreis()
* public void setLiterzahl(int)
* public int getLiterzahl()

### Package Persistenz

#### Beschreibung des Packages

Das Package *persistenz* enthält alle Klassen, die sich um das Lade und Speichern von Daten in die Datenbank kümmern.

#### Diagramme



#### Schnittstellen

Schnittstellen bestehen zu verschiedenen Logikschichten der anderen Module, z.B. *logik.dienstleistungsverwaltung, logik.terminplanung, logik.produktverwaltung*

#### Operationen

DienstleistungenDB:

* Konstruktor: public DienstleistungenDB()
* public ArrayList<DienstLeistung> dienstleistungenLaden()
* public void dlUpdate(<DienstLeistung>)
* public void dienstleistungLöschen(Dienstleistung)
* public void dienstleistungHinzufügen(Dienstleistung )

LagerDB:

* Konstruktor: public LagerDB()
* public ArrayList<Produkt> produkteLaden()
* public void produkteUpdaten(ArrayList<Produkt>)
* public void produktLöschen(Produkt)
* public void produktHinzufügen(Produkt)

TerminDB:

* Konstruktor: public TerminDB()
* public ArrayList<Termin> termineLaden(Date datum)
* public void termineSpeichern(int, int, Date, int)
* public String kundenNamenLaden(int)
* public ArrayList<Integer> adminwerteLaden()
* public ArrayList<Integer> kundenIdLaden(String)

VerkäufeDB:

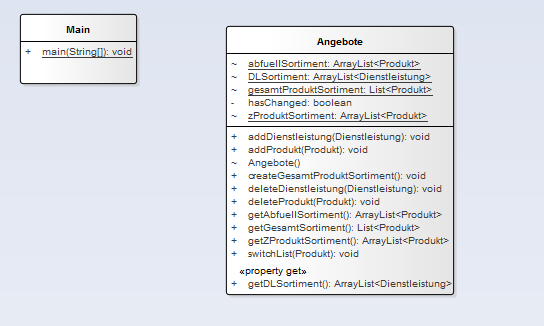
* Konstruktor: public VerkäufeDB()
* public ArrayList<Verkaufsposition> kundenEinkäufeladen(Kunde, Date)
* public ArrayList<Verkaufsposition> alleEinkäufeVonKundeLaden(Kunde kunde)
* public void kundeneinkaufHinzufügen(Verkauf)
* public ArrayList<Verkaufsposition> tagesVerkäufeLaden(Date)
* public ArrayList<Verkaufsposition> kundeneinkäufeZeitraumLaden(Kunde, Date, Date)
* public ArrayList<Verkaufsposition> VerkäufeZeitraumLaden(Date, Date)
* public ArrayList<Verkaufsposition> alleVerkäufeLaden()
* public void dlUpdaten(ArrayList<Dienstleistung>)

### Package Main

#### Beschreibung des Packages

Das Package *main* enthält die Hauptfunktion, von der aus die Software gestartet wird.

#### Diagramme



#### Schnittstellen

Die wichtigste Schnittstelle des *main*-Packages besteht zum *account,* über welchen die Software durch Anmeldung gestartet werden kann.

#### Aufteilung der Klassen in das Drei-Schichten-Modell:

GUI-Schicht:

Logik-Schicht: Angebote, Main

Persistenz-Schicht:

#### Operationen

Angebote:

* Konstruktor: package Angebote()
* public void addProdukt(Produkt)
* public void deleteProdukt(Produkt){
* public void addDienstleistung(Dienstleistung)
* public void deleteDienstleistung(Dienstleistung)
* public void createGesamtProduktSortiment()
* public List<Produkt> getGesamtSortiment()
* public ArrayList<Dienstleistung> getDLSortiment()
* public ArrayList<Produkt> getZProduktSortiment()
* public ArrayList<Produkt> getAbfuellSortiment()
* public void switchList(Produkt)

Main:

* public static void main(String[])

# Datenspeicherung (Data View)

Bei Mosti werden die erfassten Daten in einer Datenbank gespeichert. Die Verbindung wird mithilfe eines JDBC-Treibers aufgebaut, der innerhalb der Open-Source Implementierung UCanAccess auf eine Microsoft Access-Datenbank zugreift.

Aufbau der Datenbank:

*kunden:*

Enthält die Informationen, die über den Kunden abgespeichert werden.

Tabellenspalten: ID, Vorname, Nachname, Straße, PLZ, Wohnort, Telefonnummer,

*termine:*

Enthält die jährlichen Termine und ihre Belegung mit KundenId

TabellenSpalten: Tag1, Tag2, Tag3, ………, Tag365, Tag366

*mitarbeiter:*

Enthält alle Informationen, die über den Mitarbeiter und seinen Account abgespeichert werden

Tabellenspalten: ID, Vorname, Nachname, Strasse, Hausnummer, PLZ, Stadt, Telefonnummer, Benutzername, Passwort

*adminwerte:*

Enthält Konfigurationswerte, die der Mosterei-Inhaber bei Bedarf ändern kann

Tabellenspalten:Id, AnzeigeAufteilung, Zeitslotlänge, Arbeitsbeginn, Arbeitsende,

terminplanung kundenverwaltung mitarbeiterverwaltung

TerminDB KundenDB MitarbeiterDB

**Datenbank**

Adminwerte termine kunden mitarbeiter

# Größen und Leistung (Size and Performance)

Die Mosti-Software soll bis zu 5000 Kundeneinträge verwalten können.

Die hohe Anzahl an Datensätzen soll die Leistung der Software nicht merklich dezimieren, sodass alle Funktionen der Software innerhalb von 10 Sekunden ausgeführt werden.