Entwurfsklassendiagramm

# Einleitung

Um eine bessere Übersichtlichkeit zu gewähren wurden bei Parameters sofern der Datentyp (z.B. File) schon genug Beschreibung ist, der Parametername weggelassen. Allerdings bei z.B. „String“ wurde meist ein Name (z.B. URI) verwendet.

Außerdem wurde für eine bessere Lesbarkeit bei Parametern anstatt „AbstactElement“ nur „Element“ verwendet.

Außerdem werden ebenfalls aus Übersichtszwechen alle „throws“-Deklarationen nur der Beschreibung genannt.

# Package util

Das Package „util“ enthält alle Hilfsklassen, die von der Applikation gebraucht werden.

## Klasse I18nHelper

Diese Klasse verwaltet die übersetzten Texte innerhalb der Applikation. Es ist geplant alle Texte, die dem Nutzer gezeigt werden, übersetzbar zu machen. Daher wird für jeden Text eine ID vergeben. Während der Laufzeit, in der die Sprache feststeht und diese dieser Klasse bekannt ist, wird die Methode „getTextFor(String textID)“ aufgerufen mit der Text ID als Parameter und liefert den entsprechenden Text in der gewünschten Sprache als Rückgabewert. Außerdem besitzt diese Klasse noch eine Methode, in der die globale Sprache gesetzt werden kann, also die Sprache in der die Texte von den oben beschriebenen zurückgegeben werden.

## Klasse EMailSender

Diese Klasse bietet nur eine Methode öffentlich an, mit der eine E-Mail versendet werden kann. Dabei übergibt man den Body, also den darzustellenden Text als String, und als String-Array die E-Mail Adressen der Empfänger. Intern wird dann eine Verbindung zu dem SMTP-Server des Servers hergestellt, der den Mailversand dann übernehmen wird. Sofern keine Internetverbindung besteht, werden die E-Mails lokal zwischengespeichert und gesendet, sobald wieder Internetempfang besteht.

## Klasse PdfExporter

Diese Klasse ist auf das Exportieren zu pdf-Dateien optimiert. Allerdings ist dies keine allgemeingültige Klasse, sondern ist nur für die beiden Anwendungsfälle in denen ein Export zu pdf-Dateien stattfindet zuständig. Dafür bietet sie zwei Export-Methoden an, einmal für die Gästeliste, und einmal für den Speiseplan. Beide Male wird die derzeitige Hochzeitsveranstaltung als Parameter erwartet, da darüber leicht die entsprechenden Daten geladen werden können. Außerdem kann auch beides Mal ein File mitgegeben werden, der die Datei repräsentiert, die die pdf-Datei werden wird.

## Klasse SyncProvider

Diese Klasse ist für die Synchronisation mit dem Server zuständig. Normalerweise sollten keine Konflikte bestehen. Allerdings wenn eine Person offline gearbeitet hat, müssen diese Daten mit dem Server synchronisiert werden. Dafür ist die Methode „sync“ zuständig. Sie vergleicht die Änderungen mit der derzeitigen Datenbasis, sucht nach Konflikten und versucht, sofern welche existieren sollten diese zu lösen. Allerdings kann dies unter Umständen manchmal nicht automatisch gelöst werden. Deswegen wird bei einem solchen Konflikt ein Dialog erstellt, welcher dem Nutzer dies mitteilt. Der Nutzer bekommt dann verschiedene Optionen angezeigt, wie er den Konflikt lösen kann.

Die Methode „sammleOfflineÄnderungen“ ist ihrerseits für das Sammeln aller Operationen im Offline-Betrieb zuständig. Sie wird von Datenbankklasse aufgerufen, sofern ein Verbindungsfehler zum Server geschieht.

## Klasse iCalHelper

Der „iCalHelper“ nimmt in seiner Export-Funktion ein Aktions-Array entgegen, aus dem er dann eine iCal-Datei generiert. Diese Datei entspricht dem Ablaufplan der Aktionen. Außerdem nimmt die Methode noch einen File entgegen, der spezifiziert wo und mit welchem Namen die iCal-Datei auf dem lokalen System abgespeichert werden soll.

## Klasse XMLSerialisierer

Um die XML-Objekte in Entitäten umzuwandeln und andersherum, bietet diese Klasse genau zwei Methoden an, die genau dies tun. Die Methode „serialize“ nimmt eine Entität, genauer ein Serializable, um dieses in ein XML-Dokument umzuwandeln. Die gegenorientierte Methode nimmt ein „Document“ als Parameter, mit dem die Java XML-Klasse gemeint ist und wandelt dieses in eine Entität um.

## Klasse XMLTransferer

Diese Klasse fungiert quasi als Wrapper um die Klasse „XMLSerialisierer“. Die import-Funktion erlaubt das importieren von XML-Dokumenten, die danach vom „XMLSerialisierer“ in einen Entität umgewandelt wird. Die Export-Funktion macht genau das Gegenteil, wobei ein weiterer Parameter die Zieldatei bestimmt.

## Klasse EMailHelper

Diese Klasse ist wiederum als Wrapper um die Klasse „EMailSender“ gedacht. Sie hilft die Nachricht zu erstellen und den, beziehungsweise die Empfänger herauszufinden. Hierzu wird der Klasse in ihre einzige Methode eine Aktion übergeben, die geändert werden soll, nachdem die Änderungen an der Klasse vorgenommen worden sind, aber bevor diese in der Datenbank gespeichert wurden. Anhand der Daten sucht sich die Methode die „alten“ Daten aus der Datenbank heraus und kann so eine E-Mail mit den Änderungen erstellen. Mithilfe der Aktion kennt sie auch die Empfänger. Die so nun generierten/herausgesuchten Daten werden dem „EMailSender“ übergeben.

## Klasse ServerFileHandler

Um Daten auf dem Server hochzuladen und herunterzuladen bietet diese Klasse die passenden Methoden an. Für den Upload wird der passenden Methode eine URI übergeben, die dem Pfad auf dem Server entspricht und ein File, der die hochzulandende Datei ist. Der Download funktioniert ähnlich, wobei in diesem Fall nur die URI für die Datei auf dem Server gebraucht wird. Sollte ein Verbindungsfehler auftauchen (also eine ConnectionException geworfen werden), werden die Daten solange in der Applikation gehalten, bis wieder eine Verbindung aufgebaut werden kann.

# Package model

Das Package „model“ enthält das Modell der Applikation, d.h. alle Entitäsklassen und dazugehörigen Enums.

## Enum Priorität

Diese Enumeration enthält die Zahlenwerte der Prioritäten. Es gibt somit die Attributdomaine für das Attribut Priorität fest. Die Domaine ist fest definiert und kann nicht angepasst werden.

## Klasse AbstractElement

Um Entitäten direkt Methoden übergeben zu können, ohne in den Methodenköpfen alle Möglichkeiten einer Entität (also z.B. Aktion, Person, etc.) nennen zu müssen ist eine zentrale Oberklasse die beste Lösung. Sie bietet als einzige Methode „getID“ an, die die ID des Elements zurückgibt. Außerdem implementiert sie das Interface „EventSender“. Dadurch dass diese Klasse als Superklasse aller Entitäten dieses Interface implementiert müssen als Unterklassen die Methoden des Interfaces implementieren.

# Package event

Das Package „event“ enthält alle nötigen Klassen und Interfaces um ein Eventhandling innerhalb der Applikation zu ermöglichen.

## Interface EventListener

Dieses Interface müssen alle Klassen implementieren die innerhalb der Applikation Events empfangen wollen.

## Interface EventSender

Dieses Interface müssen alle Klassen implementieren die innerhalb der Applikation Events senden wollen. Als Methode wird das Senden des Events angeboten, wobei ein beliebiges Event übergeben werden kann.

## Interface ModelEvent

Um wieder ein zentrales Element zu haben, müssen alle Events dieses Interface implementieren. Außerdem sorgt es dafür, dass über eine einheitliche Schnittstelle auf die Event-Daten zugegriffen werden kann.

## Event-Klassen

Es gibt drei unterschiedliche Event-Arten. Jedes davon hat seine eigenes Klassen, die die Methoden des Interfaces „ModelEvent“ implementiert. Es handelt sich um die Klassen „CreateEvent“, „DeleteEvent“ und „UpdateEvent“, wobei die Namen selbsterklärend sind.

# Package exception

Alle Exceptions, die innerhalb der Applikation geworfen werden, sind in diesem Package gesammelt.

## Exception-Klassen

Alle Exception-Klassen haben als Superklasse die Java-Klasse „Exception“. Es gibt die „KeinZugriffException“, die geworfen wird, wenn man keinen Zugriff auf ein Element hat. „ConnectionException“, wird geworfen, wenn zum Server keine Verbindung aufgebaut werden kann und die „AuthorizationException“ taucht auf, wenn der Login fehlschlägt.

# Package datenbank

Das Package Datenbank enthält alle Komponenten, die für das Zusammenspiel mit der Datenbank relevant. Angefangen bei dem bloßen Verbindungsmanagment bis hin zu lokalen Caches. Somit ist es eines der grundlegensten Bestandteile der Software.

## Klasse Datenbank

Diese Klasse ist für das Schreiben und Lesen in/aus der Datenbank verantwortlich. Um die Entsprechenden Daten zu bekommen, implementiert sie das Interface „EventListener“. Sie bietet für das Lesen (get), Löschen (delete), Ändern (update) und das Anlegen (insert) entsprechende Methoden an. Alle Methoden erwarten als Parameters die entsprechende Entität. Des Weiteren können alle Methoden die Exceptions „ConnectionException“ und „KeinZugriffException“ werden. Außerdem besitzt sie eine Backup-Methode, die die Datenbank dazu veranlasst, ein Backup zu erstellen.

## Klasse ConnectionManager

Diese Klasse stellt eine Verbindung zur Datenbank her, dies geschieht mit einem entsprechenden Nutzernamen und Passwort. Sie kann die „AuthorizationException“ oder die „ConnectionException“ werfen.

## Klasse CSVReader & CSVWriter

Diese Klassen sind wichtig, damit offline gearbeitet werden kann. Die Writer-Klasse implementieren geauso wie die Datenbank das Interface „EventListener“ um die offline Daten synchron zur Datenbank zur halten. Beim initialen Starten werden die wichtigen Daten heruntergeladen und als CSV-Datei gespeichert. Die Reader-Klasse ist nun für den Fall da, dass keine Verbindung zum Server besteht. Während nun die Datenbank-Klasse einen Fehler wirft wird die Anfrage an diese Klasse weitergeleitet, die dann aus der lokalen Kopie die Daten liest und zurückgibt.

# Package gui

Dieses Package enthält alle Klassen, die für das Anzeigen der GUI nötig sind. Außerdem ist bezüglich des Eventhandling anzumerken, dass alle GUI-Events wie „onClick“ intern behandelt werden und nicht nach außen über Sender und Listener gegeben werden.

## Klasse GUIHochzeitsmanager

Diese Klasse bildet den Einstieg in graphische Oberfläche dieser Applikation. Sie ist für das Management des Fensters (Frames) und etwaigen Status-und Menüleisten zuständig. Sie bietet eine Start und eine End-Methode an. Außerdem erbt sie von der Java-Klasse „JComponent“.

## Klasse AbstractEventSender

Diese Klasse implementiert das Interface „EventSender“. Hiervon erben alle GUI-Klassen, die nicht schon von „GUIAbstractEntity“ erben. Diese Klasse (also GUIAbstractEntity) erbt auch von AbstactEventSender um eine Polymorphie zu vermeiden, die es in JAVA nicht gibt.

## Klasse GUIAbstactEntity

Alle Klassen, die direkt einer Entität zugehörig sind, wie z.B. „GUIAktionen“, erben von dieser abstrakten Klasse. Dies hat den Vorteil, dass alle Klassen dieselben Standardmethoden haben. Zu diesen gehört es alle Entitäten anzuzeigen, Details zu einer anzuzeigen, Dialog zu löschen, anlegen Exportieren und zum Bearbeiten anzuzeigen mit entsprechenden Parametern. Genau diese Parameter sind generisch und deswegen mit „T“ anstatt einer speziellen Entität gekennzeichnet.

### POC

Um nun die Funktionsweise dieser generischen Vererbung zu veranschaulichen folgt ein kleiner POC:

* + - 1. /\*\*
      2. \* Diese Klasse ist eine abstrakte Beschreibung der möglichen Aktionen,
      3. \* die eine GUI haben wenn sie zu einer bestinmmten Entität (extends
      4. \* AbstactElement)gebunden ist.
      5. \*/
      6. public abstract class GUIAbstractEntity<T extends AbstactElement> {
      7. public abstract void showAll();
      8. public abstract void showDetail(T entity);
      9. public abstract void showLöschenDialog(T entity);
      10. }
      11. /\*\*
      12. \* Diese Klasse ist eine konkrete Implentierung für alle GUI-
      13. \* Operationen an der Entität Aktion
      14. \*/
      15. public class GUIAktionen extends GUIAbstractEntity<Aktion> {
      16. @Override
      17. public void showAll() {}
      18. @Override
      19. public void showDetail(final Aktion entity) {}
      20. @Override
      21. public void showLöschenDialog(final Aktion entity) {}
      22. }
      23. /\*\*
      24. \* Diese Klasse ist die Überklasse aller Entitäten
      25. \*/
      26. public abstract class AbstactElement {
      27. }
      28. /\*\*
      29. \* Entity-Klassen für Aktionen
      30. \*/
      31. public class Aktion extends AbstactElement {
      32. }

Die Klasse „GUIAktionen“ zeigt nun wie das Ganze funktioniert. Sie erweitert die Klasse „GUIAbstractEntity“ generisch mit „Aktion“, wobei „Aktion“ von der Klasse „AbstractElement“ erbt. Die Klasse „GUIAbstractEntity“ nimmt alle generischen Operatoren an, die von „AbstractElement“ erben. Wie man nun in „GUIAktionen“ sieht, sind alle Parameter, die vorher generisch waren nun vom Typ Aktion.

## Alle Entity GUI-Klassen

Alle GUI-Klassen, die von GUIAbstactEntity erweitern diese Klasse mit ihrer Entität generisch. Sie bieten neben den vererbten Methoden noch ein paar weitere an, je nachdem. Die Methodennamen sind selbsterklärend. Folgend nun noch ein paar besondere Methoden:

### GUIHochzeitsveranstaltung

Diese Klasse bietet unter anderem Export-Dialoge für die Gästeliste und den Speiseplan. Beide Male wird der „pdfExporter“ benutzt. Außerdem kann man sich alle Gäste und den Speiseplan nur so anzeigen lassen.

### GUICaterer

In diesem View kann man neues Essen und Trinken anlegen, da eine komplette Verwaltung dieser in separaten Views zu weit gehen würde. Außerdem kann man sich den Caterervergleich anzeigen lassen. Dafür werden die einzelnen Leistungen der Catere inklusive Preis verglichen und gegenüber gestellt. Die Nutzbarkeit ist stark abhängigkeit davon, wie genau die Angebote spezifiziert wurden. Es bietet dem Nutzer somit einen schnellen Überblick über die Angebote.

### GUIAktionen

Diese View bietet eine visuelle Exportfunktion der Aktionen als Ablaufplan. Hierzu wird der „iCalHelper“ benutzt. Außerdem kann man hier auch die Aktion als Template speichern lassen. Hierzu wird auch Hilfe von der Klasse „XMLTransferer“ gebraucht.

## Alle anderen GUI-Klassen

Alle anderen GUI-Klassen erben direkt von „AbstractSender“ und bieten Funktionalität an, die zu keiner Entität direkt gehört. Dazu gehören das Anzeigen des Import-Dialogs, der von überall erreichbar sein soll und die Kostenkontrolle.