DOKUMENTATION:
WHO'S GONNA HELP?!

Hans Böse
Nicolai Jablonski
Jennifer Rydlichowski

Berlin, 2015
FA 35, Gruppe 2

Dokumentation: Who's Gonna Help?!

Inhaltsverzeichnis

1 Projektaufgabe bzw. Projektziel

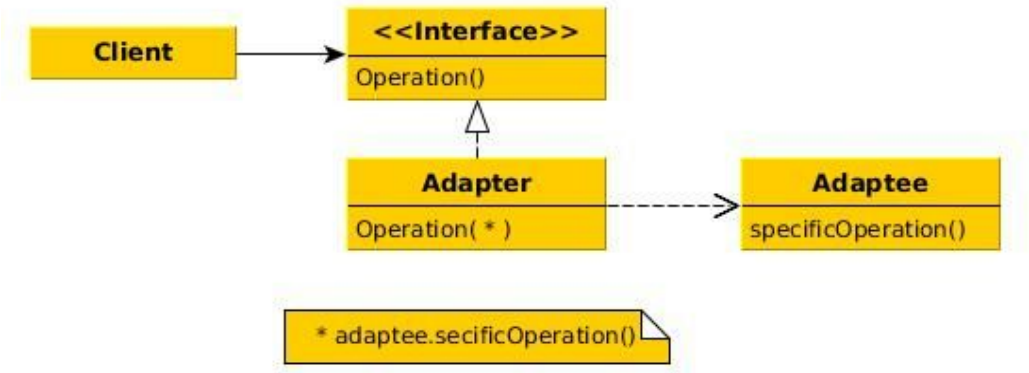
1.1 Allgemein: 3-Schichtenarchitektur

Die Drei Schichten Architektur ist ein Modell, bei dem man das Programm in drei Teile teilt. Üblicherweise in folgende drei Schichten: Benutzeroberfläche, Fachkonzept und Datenhaltung. Diese bauen von unten nach oben aufeinander auf und sollen dafür in die andere Richtung, von oben nach unten, Abhängigkeiten vermeiden.
So ist eine höhere Schicht immer von allen Schichten darunter abhängig, aber nicht anders herum. Die Benutzeroberflächenschicht ist abhängig von der Fachkonzeptschicht und indirekt von der Datenhaltungsschicht, die Fachkonzeptschicht ist nur von der Datenhaltungsschicht abhängig und die Datenhaltungsschicht ist von nichts abhängig. Ändert man eine Schicht, müssen alle darüber gelegten Schichten mit geändert werden.
Bei einer perfekten drei-Schichten-Architektur greift immer nur die obere Schicht auf eine darunter liegende Schicht zu. So ruft die Benutzeroberflächenschicht Methoden aus der Fachkonzeptschicht auf und das Fachkonzept Methoden aus der Datenhaltungsschicht. Das heißt im Umkehrschluss, dass die Benutzeroberflächenschicht ohne weiteres geändert werden kann, diese muss nur wieder die richtigen Methoden aus der Fachkonzeptschicht aufrufen.
Ändert man Methodennamen oder fügt neue Methoden in der Fachkonzeptschicht hinzu, so muss man diese auch in der Benutzeroberflächenschicht ändern bzw. verwenden. Das bedeutet, dass die Benutzeroberflächenschicht abhängig ist von der Fachkonzeptschicht, aber nicht anders herum. Die Fachkonzeptschicht ist es relativ egal, wo ihre Methoden aufgerufen werden und ihre Rückgabewerte ausgegeben werden.
Genauso ist es bei der Datenhaltungsschicht. Werden hier Methoden zugefügt oder Methodennamen geändert, müssen diese in der Fachkonzeptschicht geändert werden. Das heißt, die Fachkonzeptschicht ist abhängig vom Datenhaltungsschicht, aber nicht andersherum. Ergeben sich dadurch neue oder geänderte Methoden in der Fachkonzeptschicht, müssen diese natürlich auch wieder in der Benutzeroberflächenschicht berücksichtigt werden, denn diese ist indirekt über die Fachkonzeptschicht auch abhängig von der Datenschicht.
Daraus ergibt dich das grundlegendes Charakteristikum der Drei-Schichten-Architektur: Weder die Fachkonzeptschicht noch die Datenhaltungsschicht können auf die Benutzeroberflächenschicht zugreifen.
Ein Fazit ist, dass durch eine Schichtenarchitektur die Komplexität der Abhängigkeiten innerhalb des Systems reduziert wird, dafür aber eine Verlangsamung der Ausführungsgeschwindigkeit des Programms durch die Weiterleitung und eventuelle Umwandlung der Daten durch die Schichten auftreten kann.

1.2 Allgemein: Unser Design-Pattern - das Adapter-Pattern

In unserem Projekt wird unter anderem das Adapter-Pattern benutzt, im speziellen der Objektadapter.
Das Adapter-Pattern konvertiert die Schnittstelle (Interface) in die Schnittstelle, die der Aufrufer erwartet. Der Adapter ermöglicht die Zusammenarbeit von Klassen, die ohne nicht zusammenarbeiten könnten, weil sie inkompatible Schnittstellen haben.
Beim Objektadapter wird nur die Zielschnittstelle implementiert und die Anfragen werden vom Adapter zu einem adaptierten Objekt delegiert.

Bei dem Klassenadapter wird eine Unterklasse aus Zielschnittstelle und adaptierten Interface gebildet, während beim Objektadapter Komposition (Klassen sind aus anderen Klassen aufgebaut, d.h. Instanzvariablen sind Referenzen auf andere Klassen.) verwendet wird um Anfragen an das Adaptierte zu übergeben.
Für uns ergibt sich daraus der Vorteil, dass die Adapterklasse nicht unnötig mit Funktionen überfüllt wird, die die Anfragen noch weiter spezifizieren. Diese Methoden sind in der adaptierten Klasse ausgelagert und werden bei Bedarf aufgerufen, dadurch ergibt sich ein sauberer, verständlicherer Code.



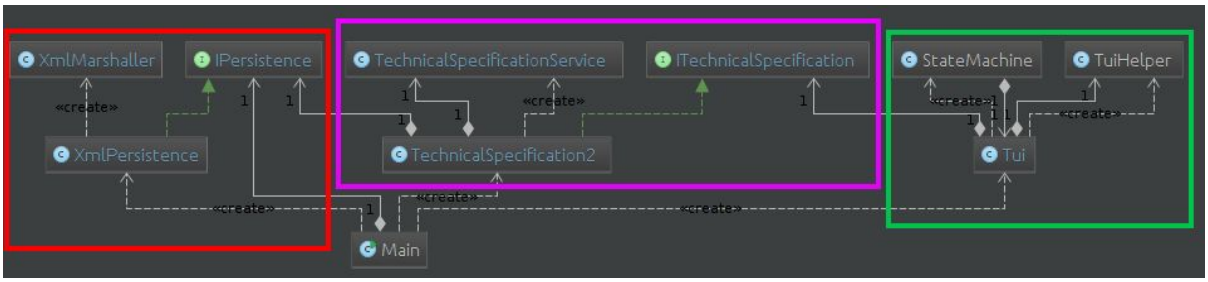
1.3 Unser Anwendungsproblem

Jeder kennt es, man hat bei sich zu Hause ein Problem z.B. eine Wand muss neu gestrichen werden und man weiß nicht wer einem helfen kann und was derjenige gerne als Bezahlung hätte für seinen Freundschaftsdienst.
Daher wollen wir eine Software entwickeln die genau diese Übersicht bietet.

2 Produkt

2.1 Realisierung der 3-Schichtenarchitektur inkl. Klassendiagramm

Die 3-Schichtenarchitektur wurde realisiert wie sehr schön in der Abbildung unterhalb des Textes zu sehen ist.



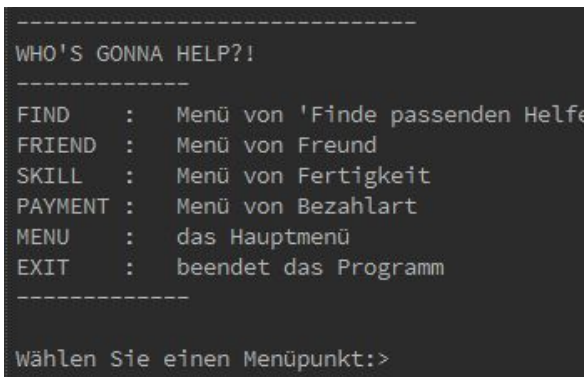
Der rot umrandete Bereich ist die Datenhaltungsschicht (*persistence*), der violette Bereich die Fachkonzeptschicht (*technical specification*) und der grüne Bereich ist die Benutzeroberflächenschicht (*tui*).

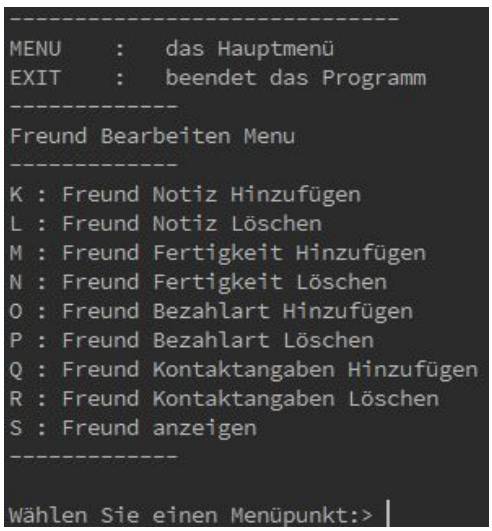
2.2 Beide Benutzeroberflächen inkl. Screenshot

Die Benutzeroberfläche besteht aus 2 Modulen, die gegeneinander frei austauschbar sind, der Tui und der Gui. Beide Module haben als Parameter ein Instanz des Interfaces ITechnicalSpecification, diese Instanz dient zur Kommunikation mit der Fachkonzeptschicht. Die Daten werden dabei in Entity-Objekten gehalten.

2.2 a Unsere Tui

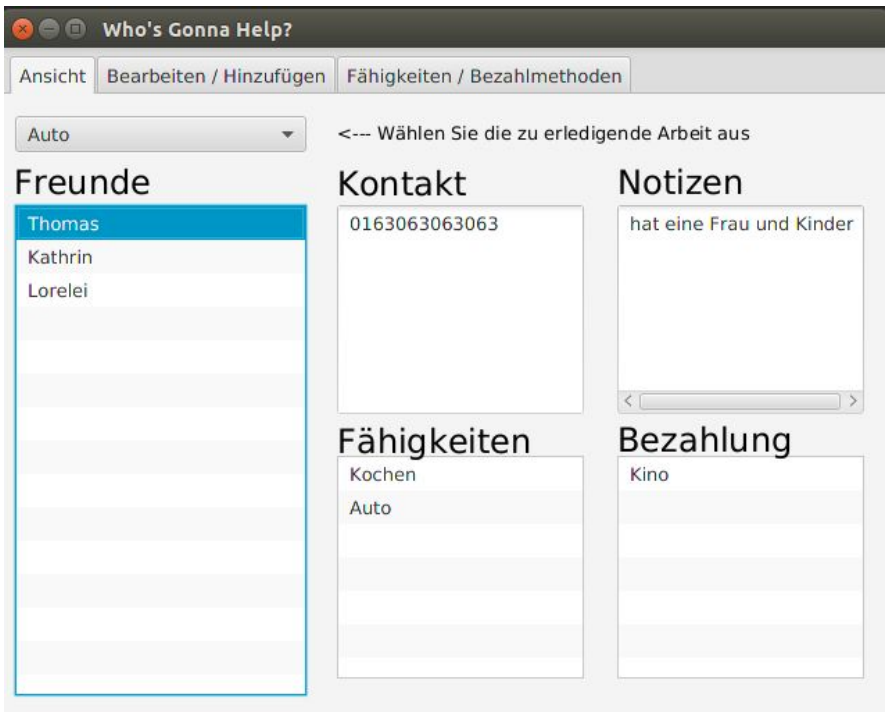
Wir haben die Tui ganz ohne Verwendung einer Bibliothek programmiert. Sie arbeitet mit dem Kommando System.out.println() um Text in die Konsole zu schreiben. Um die Benutzereingaben zu verarbeiten wird der InputStreamReader verwendet.

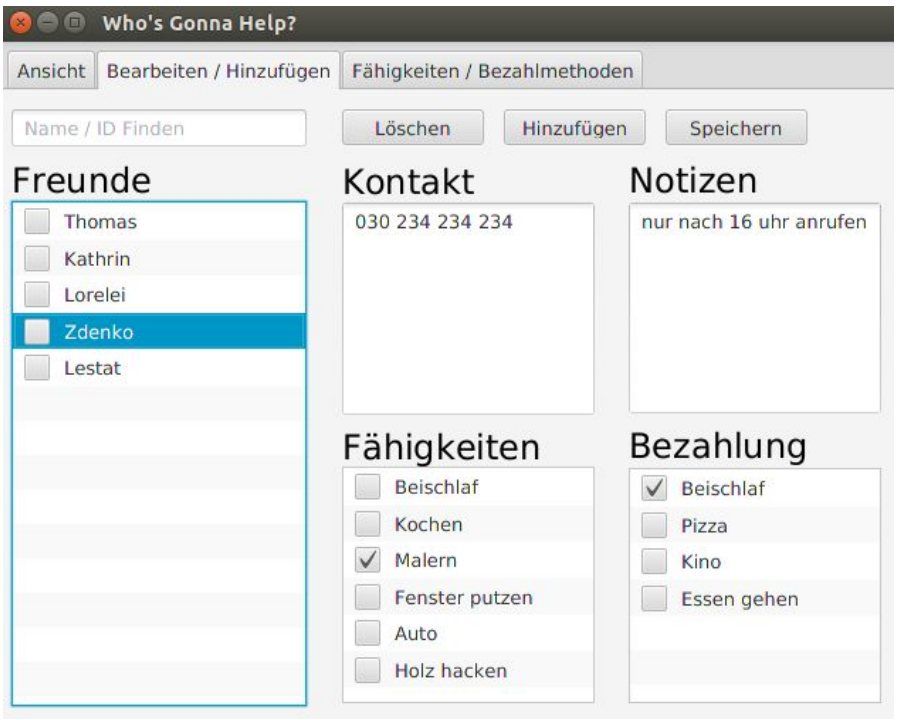




2.2 b Unsere Gui

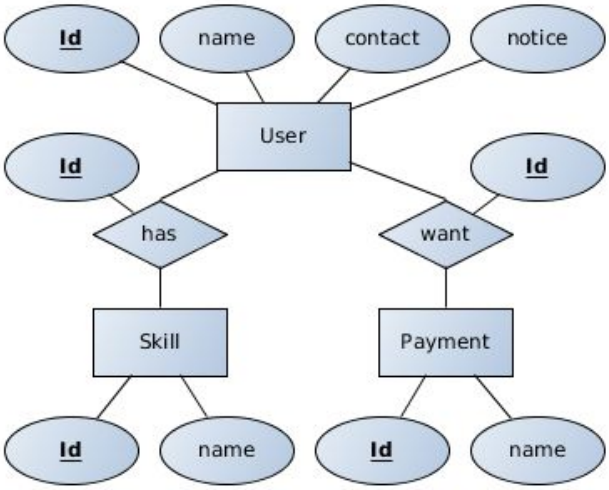
Wir haben uns entschieden die Gui mit JavaFX zu programmieren. JavaFX ist eine Plattform zur Entwicklung von Desktop- und mobilen Anwendungen. Wir haben JavaFX mit einer XML-Sprache names FXML kombiniert, um in der Gui nach Model-View-Controller arbeiten zu können. Die View ist in FXML Dateien definiert, dazu gibt es für jede Funktion einen Controller in Java und als Model arbeitet die Fachkonzeptschicht (technicalSpecification) mit der Gui zusammen.





2.3 Datenhaltungsschicht

Die Datenhaltungsschicht besteht aus 2 Modulen, die gegeneinander frei austauschbar sind, dem XML- und SQLite-Datenbankmodul. Beide Datenbankmodule implementieren das Interface IPersistence, das gleichzeitig auch zur Kommunikation mit der Fachkonzeptschicht dient. Die Daten werden dabei in Entity-Objekten gehalten.



2.3 a XML-Datenbankmodul

Das Grundkonzept des XML-Datenbankmoduls besteht auf dem Marshalling und Unmarshalling von Java-Objekten. Dabei werden die aktuellen Daten eines Objektes in eine XML-Datenstruktur umgeformt (Marshalling) und wieder zurück in ein Objekt übernommen (Unmarshalling).
Das Modul nutzt dazu die in dem Oracle Java Runtime Environment verfügbare DOM-Schnittstelle, die die XML-Datenstruktur mit einem XSD-Schema validiert. Daten, die über die IPersistence-Schnittstelle übergeben werden, werden in die XML-Datenstruktur eingefügt und vollständig auf die Festplatte als XML-Datei geschrieben. Bei dem ersten Aufruf des XML-Datenbankmoduls wird überprüft, ob die XML-Datenstruktur als XML-Datei auf der Festplatte vorhanden ist, und sofern dies zutrifft, wird die XML-Datei geladen - andernfalls wird die XML-Datei auf der Festplatte erstellt.
Die Abfrage von Daten wird direkt aus der geladen XML-Datenstruktur beantwortet, da diese im Normalfall 1:1 der XML-Datei auf der Festplatte entspricht.

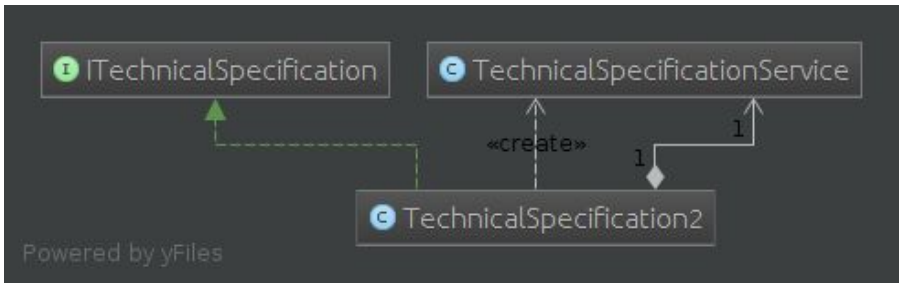
2.3 b SQLite-Datenbankmodul

Das SQLite-Datenbankmodul nutzt hibernate-entitymanger in der Version 4.3.10 um die Attribute und Assoziationen aus den Entity-Objekten in Datenbanktabelllen zu mappen.
Dazu werden Annotationen der Java Persistence API (JPA) 2.1 in den Entity-Klassen genutzt.
Wir nutzen den Datenbank-Treiber sqlite-jdbc in der Version 3.8.11.2 von Taro L. Saito et al für den Zugriff auf die SQLite Datenbank.
Wird haben Hibernate gewählt, da dieses den Datenbank-Zugriff in Form der Hibernate Query Language (HQL) und Criteria Queries abstrahiert, so dass es mit recht geringem Anpassungsaufwand möglich ist ein anderes SQL-Datenbankmanagementsystem zu verwenden, wie z.B. MariaDB, MySQL oder Microsoft SQL Server.

Wir haben uns für SQLite entschieden, da SQLite den Vorteil hat, dass es keinen externen Datenbank-Server benötigt und mit dem Firefox-Addon SQLite Manager einfach administriert werden kann.
SQLite hat allerdings auch den Nachteil, dass es nur von einem Nutzer gleichzeitig genutzt werden kann und nicht offiziell von hibernate unterstützt wird. Eine Alternative zu SQLite wäre die DBMS namens H2 im embedded-Modus gewesen, da die vollständig auf Java basiert und offiziell von hibernate unterstützt wird, aber von Datenbankadministrationsprogrammen schlecht bis gar nicht unterstützt wird.

2.4 Realisierung des Design-Patterns

In der Fachkonzeptschicht wurde das Design Pattern namens Adpater Pattern umgesetzt. Die Klasse TechnicalSpecification2 ist der Adapter, die das Interface ITechnicalSpecification implementiert und die Klasse TechnicalSpecificationService als adaptiertes Objekt instanziiert.



Hier eine Methode aus der Klasse TechnicalSpecification als Beispiel:

```
@Override
public Map<Integer, String> getAllFriendsIdAndName()
{
    List friendEntities = persistence.getAllFriends();

    return technicalSpecificationService.sortedFriendsWithIdAndName(friendEntities);
}
```

Wenn ein Client (bei uns die Tui/Gui) die Methode getAllFriendsIdAndName() der Adapterklasse aufruft, bekommt der Client eine spezifizierte Antwort mit Hilfe der adaptierten Klasse TechnicalSpecificationService zurück ohne dass dieser von der Spezifizierung weiß oder die adaptierte Klasse kennt.

2.5 Kommunikation zwischen den Schichten inkl. Sequenzdiagramm