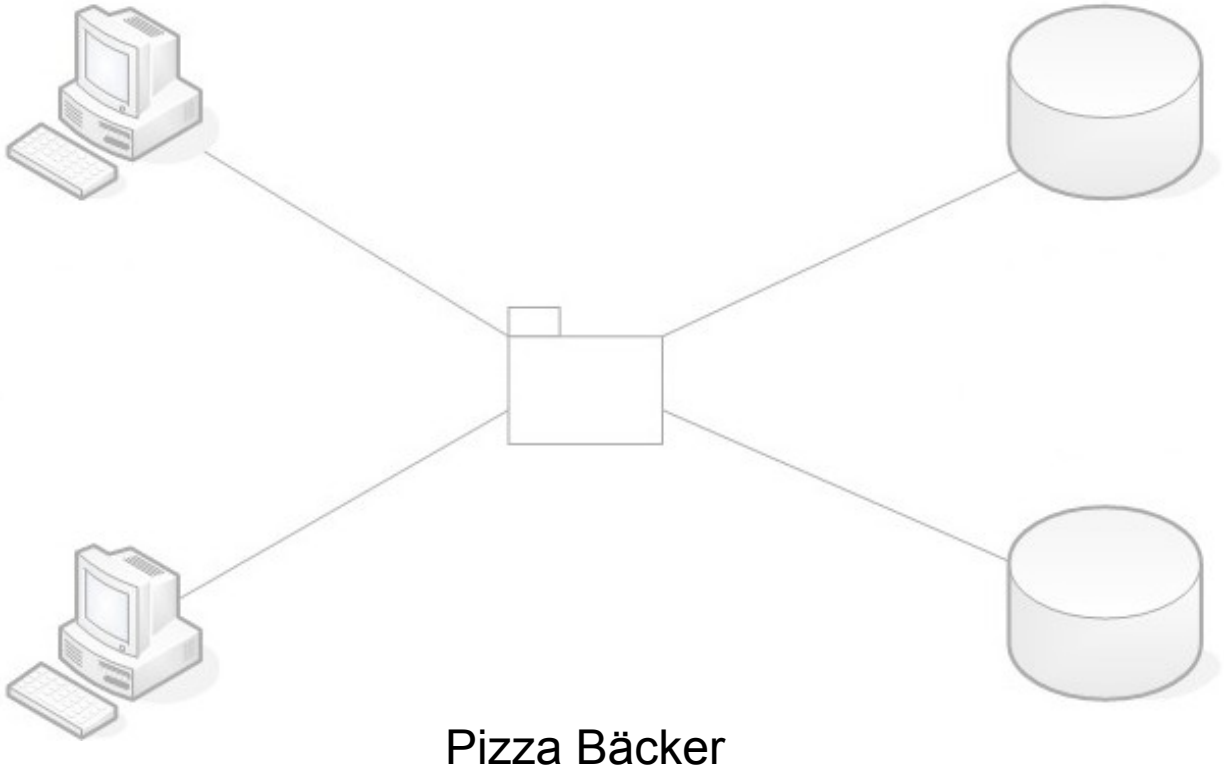
Projektarbeit AS

Drei-Schichten-Architektur



Pizza Bäcker

No Tomatoes Please

21.12.2012

OSZIMT FA06

David Schmidt, Nico Sorger

Inhaltsverzeichnis

1. Projektaufgabe

1.1. Die Drei-Schichten-Architektur

Die dreischichtige Architektur (engl. three- tier architecture) ist eine Architektur in der Softwareentwicklung, bei dem Benutzerschnittstelle (engl. user interface), Logik und Datenhaltung (engl. persistence) getrennt entwickelt und gewartet werden und sich in separaten Modulen und sich unter Umständen sogar auf verschiedenen Systemen befinden.[1][2]
Neben den gewöhnlichen Vorteilen modularer Software, wie austauschbarer Teile, besserer Arbeitsverteilung und Wiederverwendbarkeit, kann man hier ganze Schichten einfach austauschen und unabhängig von den anderen Schichten an neue oder geänderte Technologien anpassen.
Bei einer Verteilung der Schichten auf verschiedene Systeme muss man so zum Beispiel nur die Benutzeroberfläche anpassen oder ersetzen, wenn sich das Betriebssystem der zur Darstellung genutzten Computer ändert.
Die Modularität wird gewährleistet, indem die Kommunikation zwischen den Schichten durch klar definierte Schnittstellen gesteuert und begrenzt wird.

Die drei Schichten Architektur besteht aus den folgenden Schichten:
Darstellungsschicht (engl. presentation tier):
Dies ist die oberste Schicht der Architektur. Die Darstellungsschicht dient nur dazu, die Informationen anzuzeigen. Sie kommuniziert mit der Fachkonzeptschicht um Eingaben zu verarbeiten oder neue Daten zur Darstellung zu erhalten.

Fachkonzeptschicht (engl. application tier):
Die Fachkonzeptschicht kontrolliert nicht nur die Logik, also die Funktionalität des Programms, sie dient auch als Schnittstelle zwischen der Darstellungsschicht und der Persistenzschicht. Alle Daten, die geladen oder gesichert werden, müssen durch diese Schicht.

Persistenzschicht (engl. data tier):
Diese Schicht ist die unterste Schicht der Architektur und kümmert sich nur um die Verwaltung, Speicherung und das Auslesen der Daten. Sie ist neutral und unabhängig von Logik und Darstellung und ihre einzige Funktion ist die Rückgabe von Daten an die verarbeitende und damit auch darstellende Schicht.

1.2. Design-Pattern

Als Entwurfsmuster kommt das sogenannte „Fluent Interface“ zum Einsatz. Dieser Begriff wurde von Martin Fowler sowie Eric Evans geprägt. Fluent Interfaces können einem natürlich-sprachlichen Satz ähneln. Das führt dazu, dass zusätzliche Kommentare im Programmcode überflüssig werden. Die Programmierschnittstelle ist primär dafür entworfen, das der Sourcecode lesbar und fließend ist. Der Entwurf für eine schöne, fließende Schnittstelle benötigt allerdings ein gewisses Maß an Planung und Vorbereitung.[3][4] Darauf wird im Abschnitt „Realisierung des Fluent-Designs“ näher eingegangen.

einfaches 3 teile Beispiel
Stringbuilder append append tostring zb

1.3. Das Pizza-Bäcker-Problem

Unser Kunde ist ein kleiner, aufstrebender Pizzabäcker, der neben seinem kleinen Ristorante Pizzen auf Wunsch auch ausliefert.
Das von unserem Kunden beschriebene Problem gestaltet sich so, dass er bei jeder Bestellung den Preis der Pizza umständlich aus den gewünschten Zutaten selbst berechnen oder in die Kasse eingeben muss um den Endpreis nennen zu können. Dies ist vor allem an geschäftsreichen Abenden, etwa dem Wochenende, und bei Telefonbestellungen sehr unpraktisch.
Der Kunde möchte zur Optimierung der Abläufe ein Programm, in dem er die Pizzen während der Bestellung leicht bedienbar mit einer Hand zusammenstellen kann, um dem Kunden den Preis sofort nennen zu können.
Des weiteren soll das Programm sowohl auf dem modernen Bürorechner als auch dem kleinen, leistungsschwachen Kassencomputer lauffähig sein. Dafür sind zwei verschiedene, angepasste Benutzeroberflächen erforderlich.
Eine dritte, wichtige Anforderung ist die Möglichkeit, neue, saisonale Zutaten auch nachträglich hinzufügen und permanent speichern zu können, damit er dem Kunden immer wieder frische, moderne Ideen anbieten kann. Dazu gehört auch eine mögliche skalierbare Datenhaltung, sollten noch viele Zutaten und Variationen hinzukommen.

2. Produkt

2.1. Realisierung der drei Schichten

Das finale Produkt wurde vollständig in Java realisiert und wurde wie folgt logisch, modular in die Schichten unterteilt:

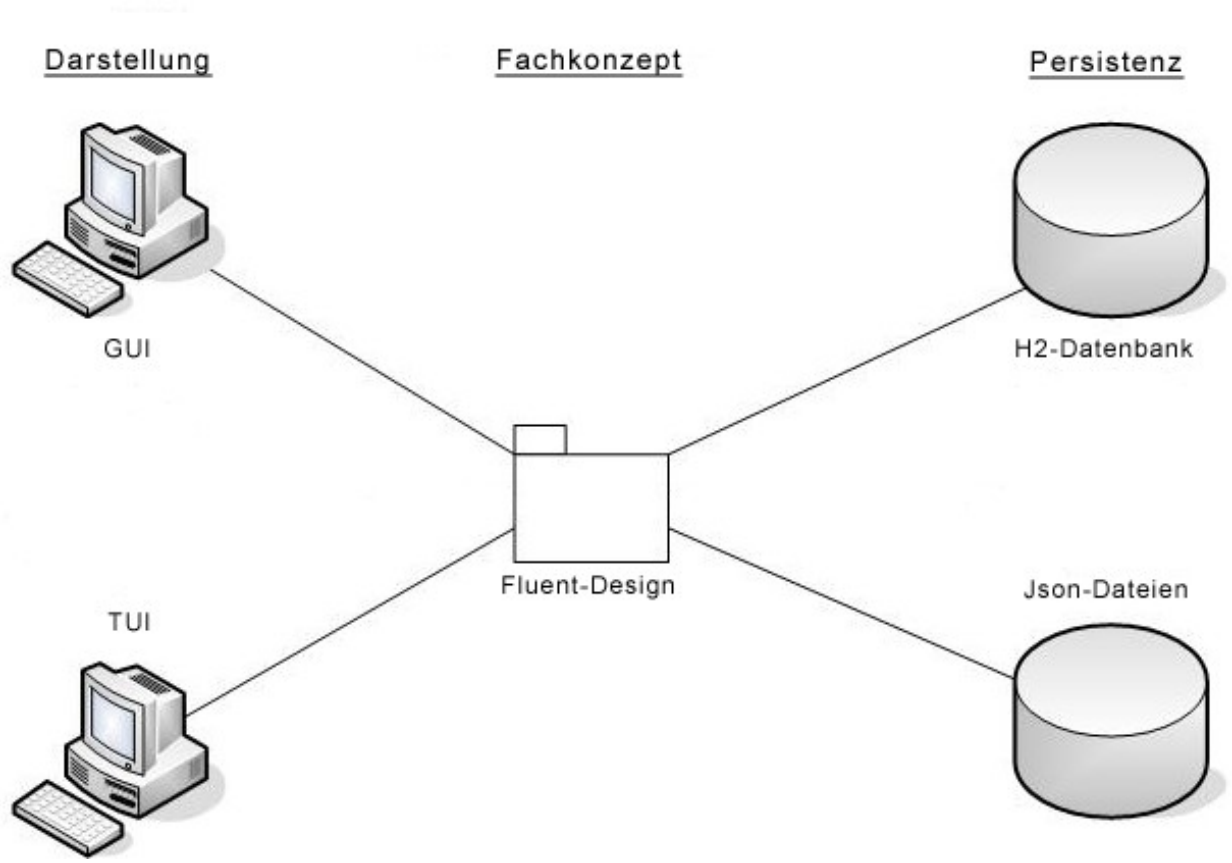


Abbildung 1: Schematische Darstellung der Umsetzung der drei Schichten

Die beiden Module für die grafische Benutzerschnittstelle sowie die Terminalbenutzerschnittstelle sind Module der Darstellungsschicht. Beide sind beliebig gegenseitig austauschbar. Sie enthalten keine Logik oder direkten Zugriff auf die Datenhaltung. Für alle Aktionen, abseits der reinen Darstellung, werden die in dem Logik-Interface definierten Methoden zur Kommunikation mit der Fachkonzeptschicht benutzt. Um ausführbar zu sein, muss eine Benutzerschnittstelle selbst die abstrakte Klasse AbstractUserInterface implementieren.
Die Fachkonzeptschicht wurde durch eine Logik gelöst, welche auf das Entwurfsmuster Fluent Interface beruht. Das Modul wurde ebenfalls so entwickelt, dass man es jederzeit und problemlos durch eine andere Logik ersetzen kann. Dieses Modul erhält weder direkten Zugriff auf die

Datenhaltung, noch besitzt es eine Möglichkeit Daten zu visualisieren. Ein Fachkonzeptmodul muss die abstrakte Klasse AbstractLogic implementieren und kann zur Kommunikation mit der Datenhaltung alle in dem Persistence-Interface definierten Methoden aufrufen.
Das JsonFilePersistence-Modul und das H2Persistence-Modul sind die beiden realisierten Module der Persistenzschicht. Beide Module implementieren das Persistence-Interface. Jedes weitere Datenhaltungsmodul müsste das Interface ebenfalls implementieren.

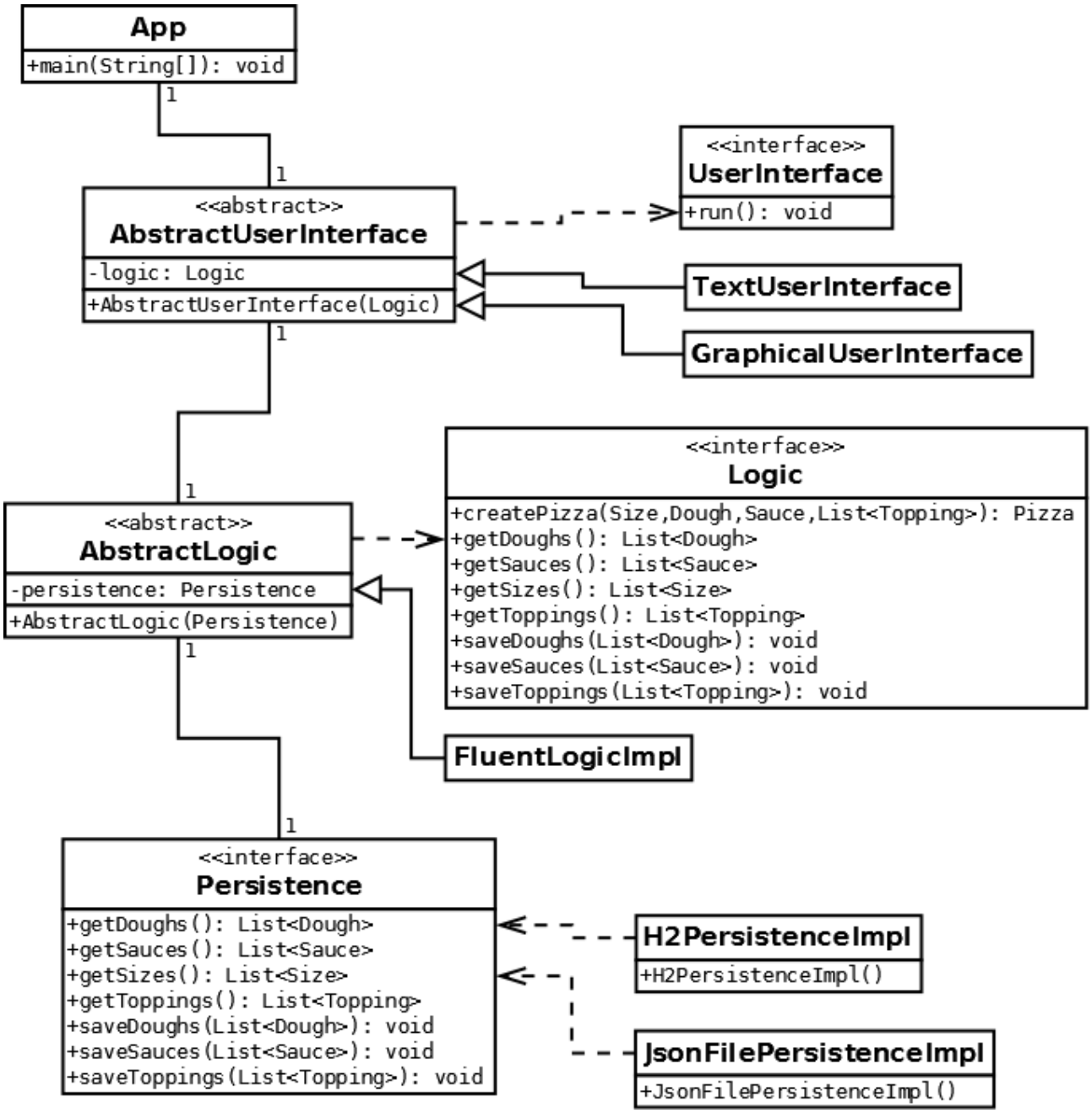


Abbildung 2: Klassendiagramm der elementaren Klassen der drei Schichten

2.2. Benutzeroberflächen

2.2.1. Grafisches User Interface (GUI)

Das grafische User Interface wurde mit Swing realisiert. Swing ist ein Framework für grafische Benutzeroberflächen und ist in der Java Standard Edition bereits enthalten.
Das Design der grafischen Oberfläche hält sich, wie geplant, stark an die Vorlage des Pflichtenheftes.
Die Oberfläche ist in zwei Tabs unterteilt. Der erste Tab enthält alle Elemente die, zur Bearbeitung der verfügbaren Zutaten nötig sind. Für jede der drei möglichen Bestandteile gibt es also eine Übersichtstabelle und Felder um sie hinzuzufügen.
Der zweite Tab stellt alle Funktionen bereit, die zur Erstellung von Pizzen nötig sind. Man wählt einen Teig und eine Sauce und fügt dann die verschiedenen Beläge hinzu.
Ist die gewünschte Pizza zusammengestellt, schließt man diese ab, bekommt den Preis und die benutzten Zutaten angezeigt.

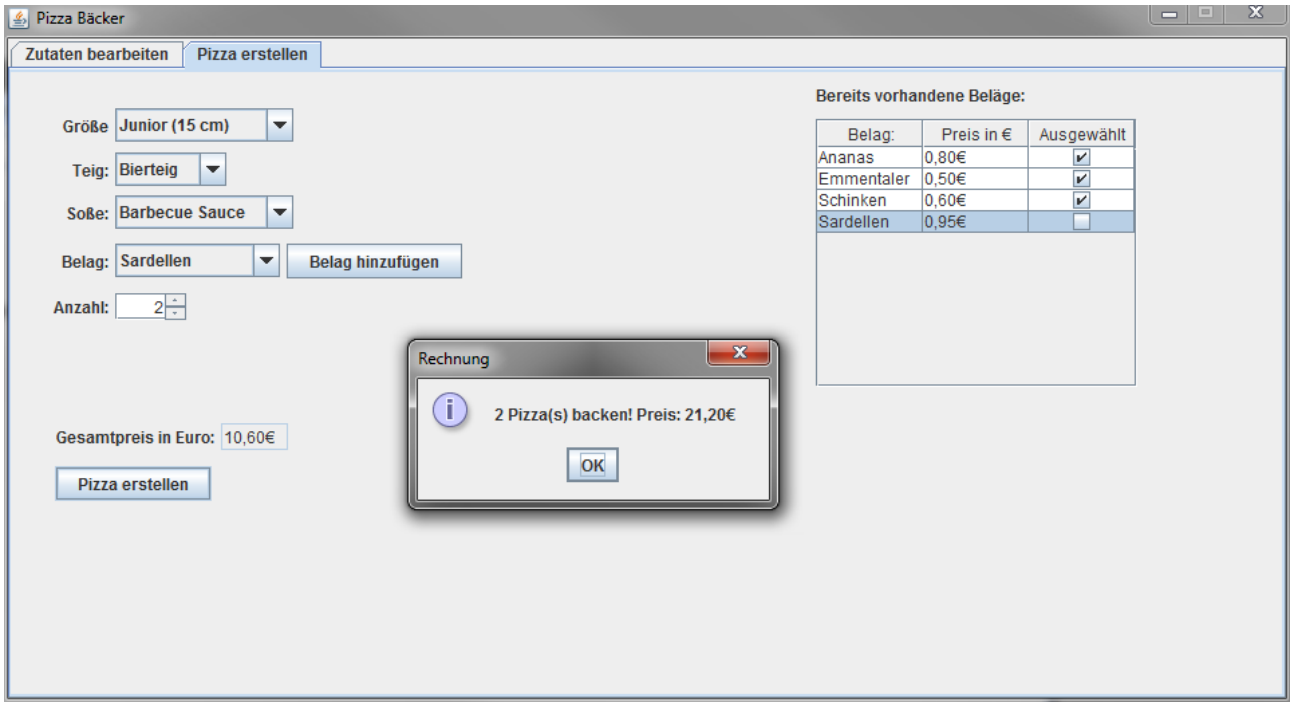


Abbildung 3: Erstellung einer Pizza in der GUI

2.2.2. Text User Interface (TUI)

Für die Umsetzung der Darstellungsschicht im Text-Modus haben wir zu JCurses[5] gegriffen. Dies ist eine Bibliothek, um Textmodus-basierte Anwendungen in Java zu schreiben. Dabei wird von

JCurses eine Fenstertechnik für das Terminal/die Konsole angeboten, die ähnliche wie AWT aufgebaut ist, jedoch auf die „curses“-Bibliothek basiert. JCurses ist unter der GNU Lesser General Public License veröffentlicht und sowohl für Linux als auch für Windows frei verfügbar.
Auf Grund von Beschränkungen von Terminals und JCurses musste das TUI neu designed werden.

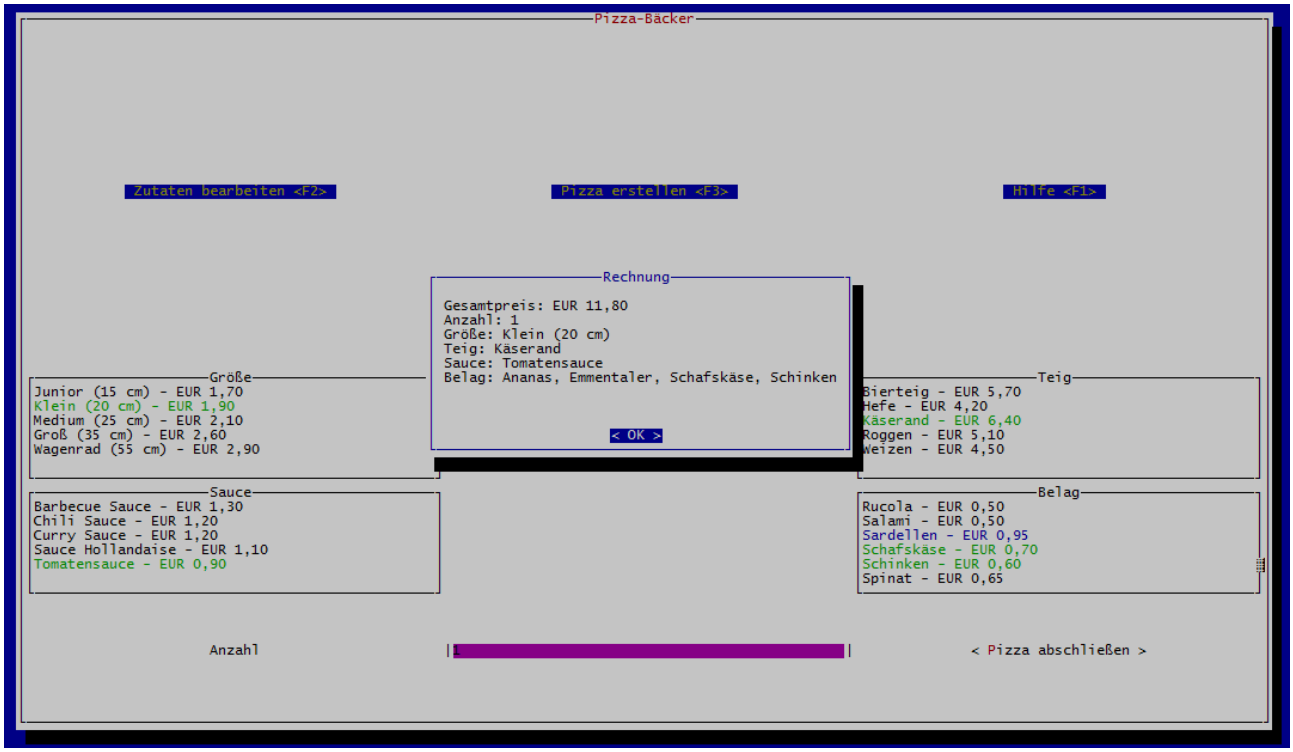


Abbildung 4: Erstellung einer Pizza in der TUI

2.3 Datenhaltung

2.3.1 H2-Datenbank

Als Lösung für das Datenbankmodul haben wir uns für die H2-Datenbank entschieden. Die H2-DB[6] ist eine eingebettete, relationale Datenbank basierend auf dem SQL-Standard. Die Bibliothek ist ein Open-Source Java Projekt und daher verlässlich und einfach zu implementieren.
Die vier verfügbaren Komponenten der Pizzen werden hier in vier Tabellen abgebildet.

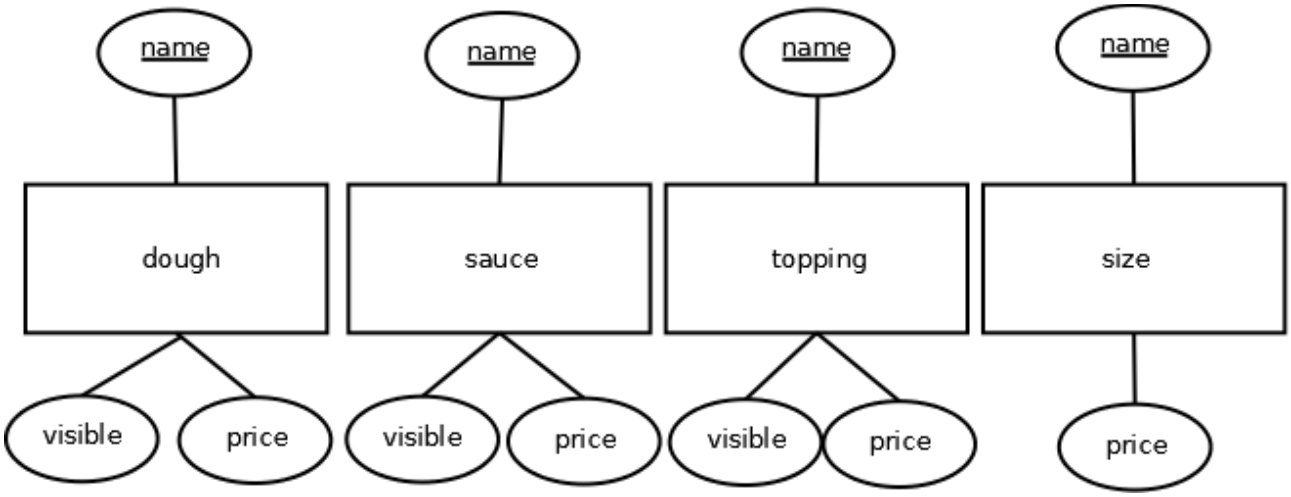


Abbildung 5: ER-Diagramm verwendeter Datenbank-Tabellen

2.3.2 JSON

Bei der zweiten Datenhaltung haben wir uns für JSON[7] (JavaScript Object Notation) entschieden, da es einen sehr einfachen Aufbau hat und daher für Mensch und Computer leicht les- und schreibbar ist (siehe Abbildung 6). Für die Serialisierung und Speicherung sowie dem Lesen und der Deserialisierung der Java-Objekte in und aus den Dateien benutzen wir die Bibliothek GSON[8], die eine simple Schnittstelle bietet. Die serialisierten Daten werden entsprechend den vier Komponenten der möglichen Pizzen in vier Dateien persistiert.

```
[
    {
        "name": "Tomatensauce",
        "price": 90,
        "visible": true
    },
    {
        "name": "Sauce Hollandaise",
        "price": 110,
        "visible": true
    },
    {
        "name": "Barbecue Sauce",
        "price": 130,
        "visible": true
    },
    {
        "name": "Curry Sauce",
        "price": 120,
        "visible": true
    },
    {
        "name": "Chili Sauce",
        "price": 120,
        "visible": true
    }
]
```

Abbildung 6: Inhalt der Sauces.json-Datei

2.4 Realisierung des Entwurfmusters

Abbildung 7: Vorbereitung einer realen Pizza

Um eine fließende, realitätsnahe Programmierschnittstelle zu erhalten, wurde erstmal der logische Ablauf beim Pizza-Backen betrachtet.
Zuerst wird der Teig geknetet. Anschließend wird dieser ausgerollt. Danach wird eine Sauce auf dem zuvor ausgerollten Teig verteilt. Es folgt das Streuen der Zutaten auf die Pizza. Abschließend wird die Pizza in den Ofen geschoben und gebacken. Zum Schluss erhalten wir eine fertige Pizza.

Abbildung 8: Fertige, reale Pizza

Diese Reihenfolge ist logisch strukturiert. Es ist zum Beispiel unsinnig, dass zuerst die Zutaten auf die Pizza gestreut werden und erst danach der Teig ausgerollt wird.

Um diese Reihenfolge in der Programmiersprache umzusetzen, wurde erstmal eine einfache Grammatik entworfen.

Die Grammatik G = (V, S, P, S) ist gegeben mit
V = {S, B, C, D, E},
S = {knead_, rollOut_, spread_, sprinkle_, bake} und
P = {S ? knead_B, B ? rollOut_C, C ? spread_D, D ? sprinkle_E, E ? bake}

und produziert die Sprache L(G) = {knead_rollOut_spread_sprinkle_bake}. Der Zustandsgraph des deterministischen endlichen Automaten (DEA) sieht so aus:

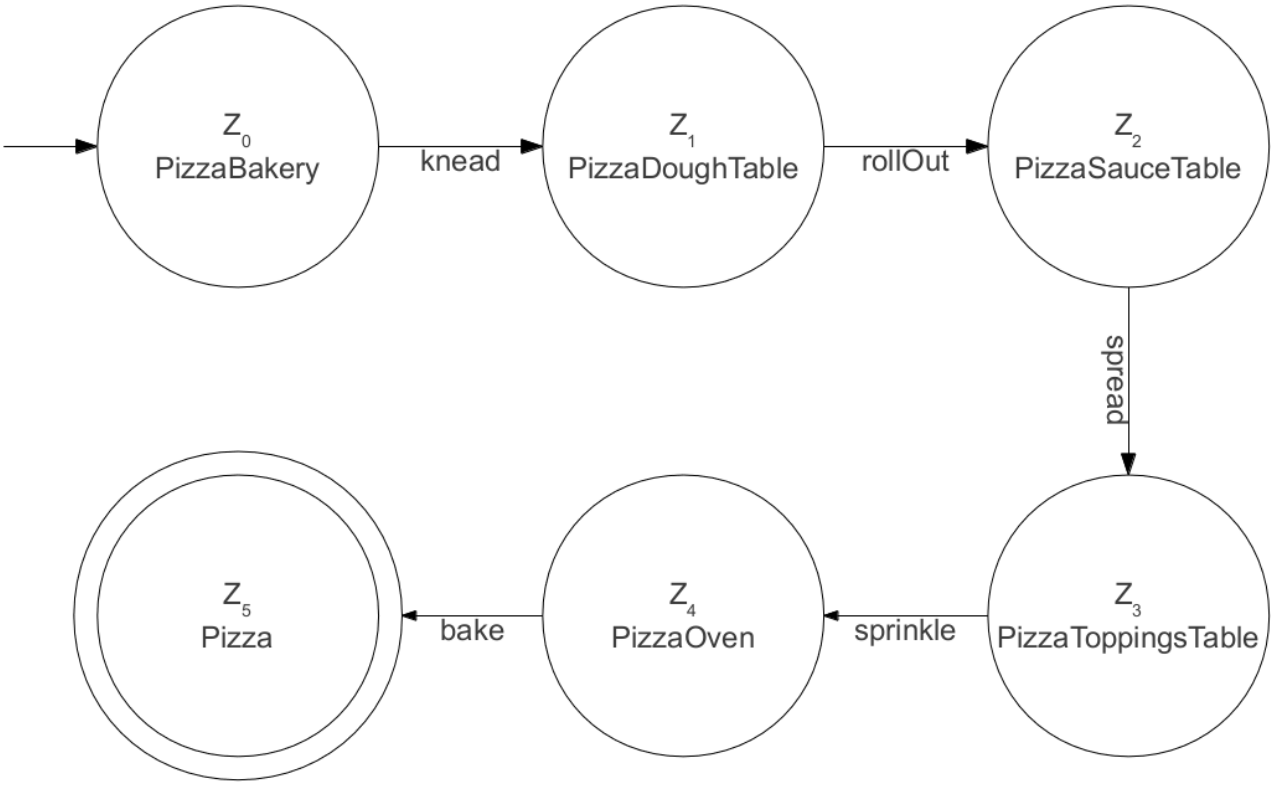


Abbildung 9: DEA der PizzaBakery

Im Quelltext wurden die verschiedenen Zustände mit Hilfe von Klassen realisiert. Die Start-Klasse ist PizzaBakery, die die Methode knead() bereitstellt. Mit dieser Methode gelingt der Übergang zur Klasse PizzaDoughTable. Dort gibt es die Methode rollOut(), mit der es möglich ist, zur nächsten Klasse, PizzaSauceTable, zu kommen. Diese Klasse stellt die Methode spread() zur Verfügung, die den Übergang zur Klasse PizzaToppingsTable ermöglicht. Dort befindet sich die Methode sprinkle(), die zur Klasse PizzaOven führt. PizzaOven gibt dann die fertige Pizza zurück, sobald die Methode bake() aufgerufen wurde.[9]

2.5 Kommunikation zwischen den Schichten

In Abbildung 10 sind, repräsentativ für die Kommunikation zwischen den Schichten, zwei Sequenzen abgebildet.

In dem ersten Beispiel ist abgebildet, was passiert, wenn der User den Button betätigt um einen neuen Belag in das Programm einzutragen.

Die zweite Sequenz verdeutlicht die Abläufe beim Abschließen einer nach Kundenwunsch zusammengestellten Pizza mit anschließender Preisausgabe.

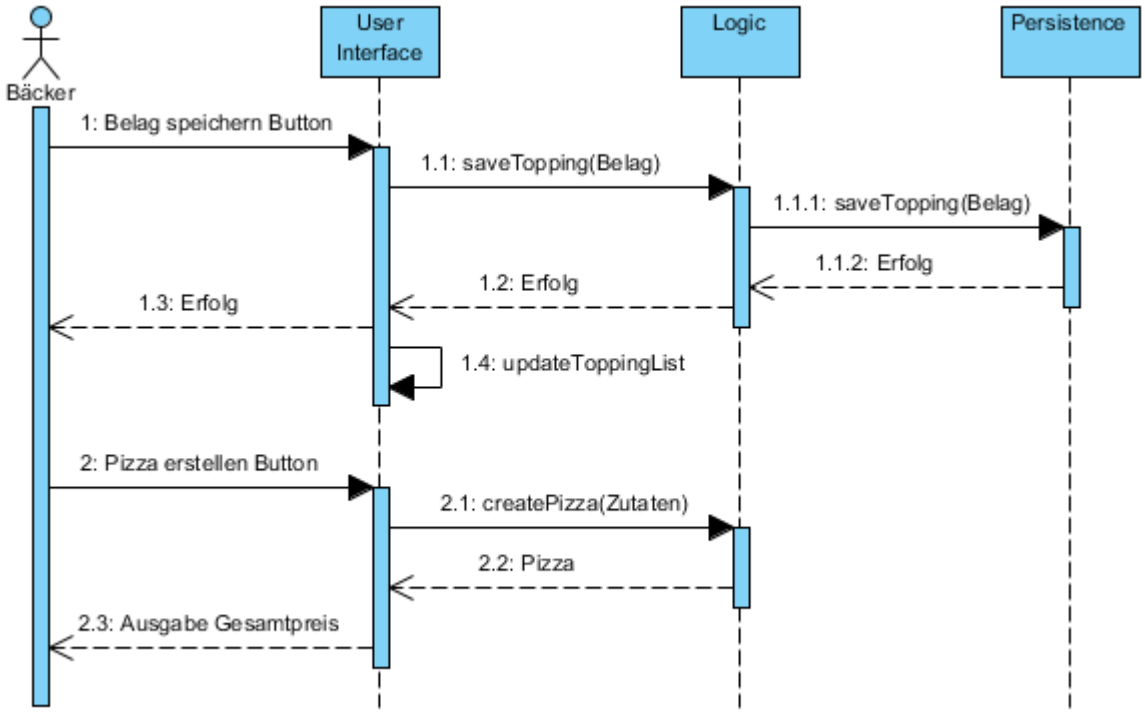


Abbildung 10: Sequenzdiagramm vom Speichern eines neuen Belages und dem Abschließen einer Pizza

I. Quellen

[1] en.wikipedia.org: Three-tier architecture. http://en.wikipedia.org/wiki/Multitier_architecture#Three-tier_architecture (abgerufen am 18. Dezember 2012)

[2] Manfred Rätzmann, Three-Tier-Development. http://www.dfpug.de/konf/konf_1998/09_tier/d_tier/d_tier.htm (abgerufen am 18. Dezember 2012)

[3] Fowler, Martin: Artikel „FluentInterface“ vom 20. Dezember 2005. http://martinfowler.com/bliki/FluentInterface.html (abgerufen am 16. Dezember 2012)

[4] Schiffer, Bernd: Artikel „Flexibel programmieren mit dem Fluent Interface - Flüssiger Erbauer“ vom 04. November 2008. http://www.heise.de/developer/artikel/Flexibel-programmieren-mit-dem-Fluent-Interface-227052.html (abgerufen am 20. Dezember 2012)

[5] Projekt-Seite von JCurses. http://sourceforge.net/projects/javacurses/ (abgerufen am 20. Dezember 2012)

[6] Projekt-Seite der H2-Datenbank. http://www.h2database.com (abgerufen am 16. Dezember 2012)

[7] Projekt-Seite von JSON-Seite. http://json.org/ (abgerufen am 10. November 2012)

[8] Projekt-Seite GSON-Bibliothek. http://code.google.com/p/google-gson/ (abgerufen am 10. November 2012)

[9] Körner, Heiko Prof. Dr.: Skript zur Vorlesung Theoretische Informatik I/II an der Hochschule Karlsruhe vom Oktober 2007, Kapitel 2.1 „Grammatiken und die Chomsky–Hierarchie“, Seite 27 ff. sowie Kapitel 2.2 „Endliche Automaten“, Seite 34 ff.

II. Abbildungen

Titelblatt, Abbildung 1: Modifizierte Grafik, http://de.wikipedia.org/w/index.php?title=Datei:3-Tier.jpg&filetimestamp=20070917103211

Abbildung 2, 5: Erstellt mit DIA, http://dia-installer.de/

Abbildung 3, 4, 6, 9: Screenshots, Eigenanfertigung

Abbildung 7, 8: Screenshots, http://www.youtube.com/watch?v=MjuiazBz5Eo

Abbildung 10: Erstellt mit Visual Paradigm Modeler Edition, http://www.visual-paradigm.com