Softwaretechnik II – Praktikum

Subsystem 4 – Zubereitung

Eine Dokumentation von:

J. Faßbender

J. Gobelet

L. Gobelet

E. Gödel

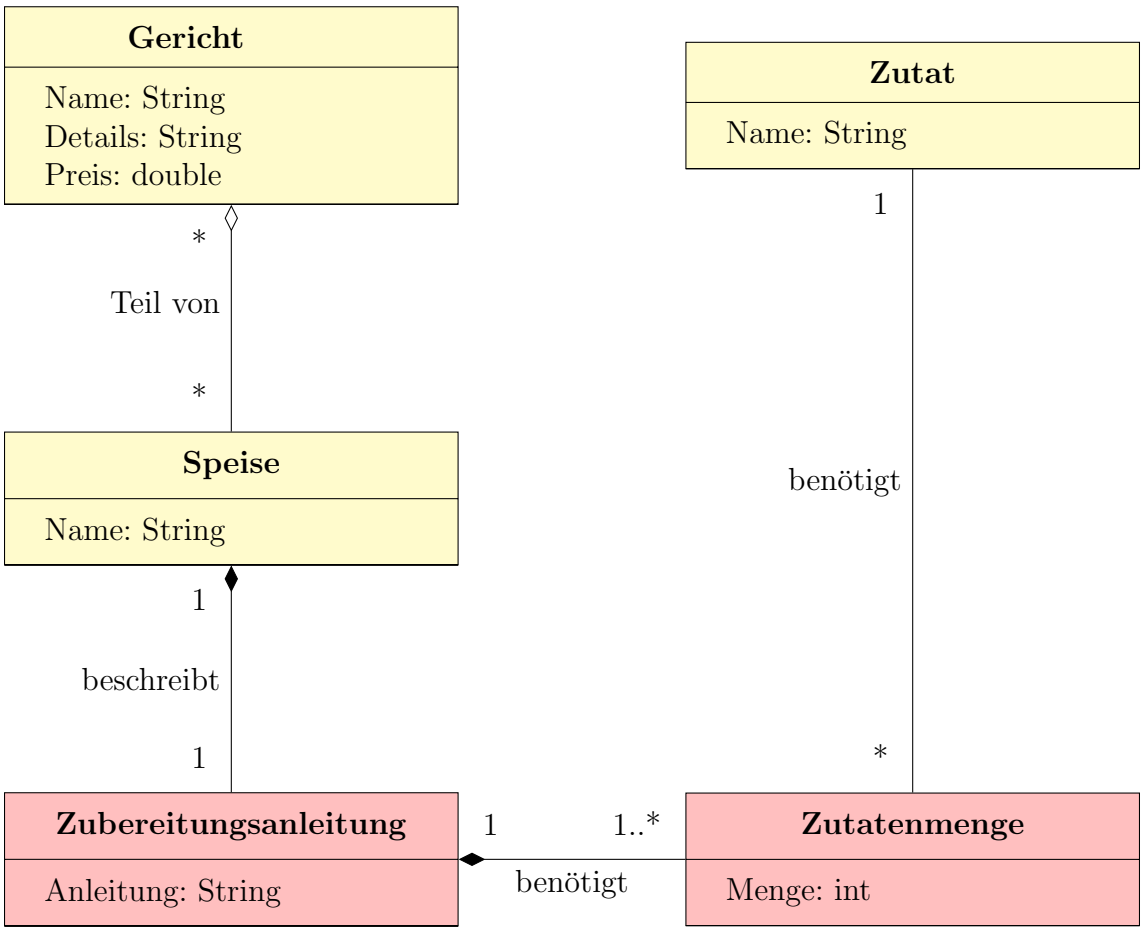
Inhaltsverzeichnis

Abbildungsverzeichnis

1 Meilenstein 1 – Datenzugriffsschicht

1.1 Teilaufgabe 1: Ausschnitt aus Logischem DM mit Entities und Value Objects

1.1.1 Klassendiagramm



: Value Object
: Entity

1.1.2 Fachliches Glossar

Geschäftsobjekt	Attribut	Erklärung
Gericht		Vom Restaurant angebotenes Mahl.
	Name	Gerichtsbezeichnung.
	Details	Wird dem Gast angezeigt. Enthält nähere Angaben zu den Zutaten.
	Preis	Geldbetrag der für das Gericht zu bezahlen ist.
Speise		Teil eines Gerichts. Beispielsweise wäre eine Salatbeilage als Speise zu verstehen.
	Name	Bezeichnung der Speise.
Zubereitungsanleitung		Leitfaden zur Zubereitung einer Speise.
	Anleitung	Erklärender Text, der beschreibt, wie eine Speise zuzubereiten ist.
Zutat		Benötigt für die Zubereitung einer Speise.
	Name	Bezeichnung der Zutat.
Zutatenmenge		Zuordnung zwischen Zutat und Zubereitungsanleitung. Gibt die Menge einer Zutat an, die für die Zubereitung notwendig ist.
	Menge	Die benötigte Menge.

1.1.3 Erweiterungen der Aufgabenstellung

Da es in unserem Logischen Datenmodell keine 1:1-Beziehung gab, haben wir eine zusätzliche redundante Entität eingebaut.

Hierbei handelt es sich um die Entität Speise. Diese Entität hätte genauso gut einfach Teil der Zubereitungsanleitung sein können und ist nur in unser Modell aufgenommen worden, damit wir die für die Aufgabenstellung benötigte 1:1-Beziehung in unserem Diagramm haben.

1.1.4 Erläuterungen

Wir haben Zubereitungsanleitung als Value Object und nicht als Entity deklariert, da hier unserer Meinung nach Sharing nicht sinnvoll ist und ein Zubereitungsanleitungsobjekt deshalb persistent als Teil der zugeordneten Speise in der Datenbank gespeichert werden sollte.

Gleiches gilt für die Zutatenmenge.

1.2 Teilaufgabe 2: Entities und Value Objects mit JPA-Annotierung

1.2.1 Annotationen der Entities und Value Objects

Gericht

```
@Entity
public class Gericht {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private int id;
  private String name;
  private String details;
  private double preis;

  // Ein Gericht besteht aus mehreren Speisen und eine Speise kann
   mehreren Gerichten zugeordnet sein.
  @ManyToMany
  @JoinTable(name = "gericht_speise",
    joinColumns = @JoinColumn(name = "gericht_id"),
    inverseJoinColumns = @JoinColumn(name = "speise_id")
  )
  private Set<Speise> speisen = new HashSet<Speise>();
```

Speise

```
@Entity
public class Speise {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private int id;
  private String name;

  // bidirektionale Beziehung: Gericht kennt zugehoerige Speisen und
    die Speisen kennen zugehoerige Gerichte
  @ManyToMany(mappedBy = "speisen")
  private Set<Gericht> gerichte = new HashSet<Gericht>();
```

Zubereitungsanleitung

```
@Embeddable
public class Zubereitungsanleitung {
  private String anleitung;

  // Die Anleitung enthaelt mehrere Zutatenangaben als Value-Objects
  @ElementCollection (targetClass = Zutatenmenge.class, fetch =
```

```
    FetchType.EAGER)
  @CollectionTable(name = "ZUTATENANGABE")
  private Set<Zutatenmenge> angaben = new HashSet<Zutatenmenge>();
```

Zutat

```
@Entity
public class Zutat {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private int id;
  private String name;
```

Zutatenmenge

```
@Embeddable
public class Zutatenmenge {
  private int menge;

  @ManyToOne
  private Zutat zutat;
```

1.2.2 H2-Console

SELECT * FROM GERICHT;

ID	DETAILS	NAME	PREIS
10	Voll das Oma-Essen!	Kartoffelbrei mit Möhren	7.5
11	Jede Erbse macht einen Knall!	Kartoffelbrei mit Erbsen	8.5

(2 rows, 9 ms)

Abbildung 1: Gerichtstabelle

SELECT * FROM SPEISE;

ID	ANLEITUNG	NAME
7	Möhren und Pfeffer umrühren!	Möhrengemüse
8	Erbsen, Salz und Pfeffer verbrennen lassen!	Erbsengemüse
9	Kartoffeln, Salz und Butter vermatschen!	Kartoffelbrei

(3 rows, 3 ms)

Abbildung 2: Speisentabelle

SELECT * FROM ZUTAT;

ID	NAME
1	Erbse
2	Butter
3	Salz
4	Möhre
5	Pfeffer
6	Kartoffel

(6 rows, 1 ms)

Abbildung 3: Zutatentabelle

SELECT * FROM ZUTATENMENGE;

SPEISE_ID	MENGE	ZUTAT_ID
7	1	5
7	3	4
8	100	1
8	2	3
8	5	5
9	6	6
9	5	3
9	2	2

(8 rows, 8 ms)

Abbildung 4: Zutatenmengentabelle

SELECT * FROM GERICHT_SPEISE;

GERICHT_ID	SPEISE_ID
10	7
10	9
11	8
11	9

(4 rows, 1 ms)

Abbildung 5: Zuordnungstabelle Gericht - Speise

1.3 Teilaufgabe 3: Factories und Repositories

Factory für Erstellung von Gerichten

```
@Component
public class GerichtFactory {

  // Erstelle ein Gericht, das nur aus einer Speise besteht.
  public static Gericht createGerichtWithSpeise(String name, String
   details, double preis, Speise speise) {
```

```
    Gericht gericht = new Gericht(name,details, preis);
    gericht.addSpeise(speise);
    // Rueckreferenz setzen
    speise.addGericht(gericht);
    return gericht;
  }

  // Erstelle ein Gericht, das aus mehreren Speisen besteht.
  public static Gericht createGerichtWithSpeisen(String name, String
    details, double preis, Collection<Speise> speisen) {
    Gericht gericht = new Gericht(name,details, preis);
    gericht.addSpeisen(speisen);
    for(Speise s : speisen) {
      // Rueckreferenz setzen
      s.addGericht(gericht);
    }
    return gericht;
  }
}
```

Hier sieht man gut warum Factories notwendig sind. Bei der Estellung von Gerichten muss zugleich die Rückreferenz von Speise auf Gericht gesetzt werden.

Factory für Erstellung von Gerichten

```
public interface SpeiseRepository extends CrudRepository<Speise,
   Integer> {
  // Die Abfrage ist in JPQL geschrieben - Eine objektorientierte
   Abfragesprache, welche SQL aehnlich ist
  // Findet alle Speisen, die eine bestimmte Zutat enthalten
  @Query("select s from Speise s join s.anleitung a join a.angaben
   ang where ang.zutat = :zutat")
  List<Speise> findByContainsZutat(@Param("zutat")Zutat zutat);
}
```

Ausgabe in der Konsole

```
    // gib alle Speisen aus, die Salz enthalten
    System.out.println("\nSalzige Speisen: ");
    speiseRepository.findByContainsZutat(zutaten.get("Salz")).
   forEach(s -> System.out.println(s.getName()));
```

Folgendes wird dann in der Konsole ausgegeben:

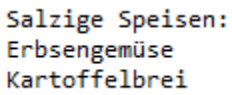


Abbildung 6: Ausgabe in der Konsole