EXKURS DDD:

Persistierung von Value Objects mit JPA

Persistierung von Value Objects mit JPA

Es gibt mehrere Möglichkeiten, VO zu speichern:

- Eingebettet in die Tabelle des „Elternobjekts“
- Serialisierung
- In einer eigenen Tabelle (als DB-Entity)

NICHT als DDD-Entity!

Die Wahl kann individuell nach VO getroffen werden.

VO eingebettet in Elterntabelle speichern

- VO wird in der DB in der selben Tabelle wie das Elternobjekt gespeichert, dem es zugeordnet ist
- in JPA über „@Embeddable / @Embedded“
- Beispiel:
  - Objektsicht: Person mit Adresse
  - Relationale Sicht: Adresse wird in „Person“-Tabelle eingebettet

VO in Elterntabelle speichern am Beispiel einer JPA-Entity

```java
@Entity
public class Person {

    @Id
    @GeneratedValue
    private Long id;

    private String name;

    @Embedded
    private Address address;
}
```

```java
@Embeddable
public class Address {
    private final String strasse;
    private final String plz;
    private final String ort;
    //...
}
```

ID	Name	Straße	PLZ	Ort	…
42	Hugo Müller	Teststr. 1	76131	Karlsruhe	…

DHBW

VO in Elterntabelle speichern

Vorteile:

- Einfache Umsetzung, mit den meisten ORM-Tools problemlos möglich
- Erlaubt Queries über Elemente des VO

Nachteile:

- Ggf. Denormalisierung der DB
  - Im Beispiel: wenn mehrere Personen die selbe Adresse haben
- Funktioniert nur bei 1:1-Beziehungen

VO serialisieren

- Das VO wird in einen einzelnen Wert konvertiert und in einer Spalte der Elterntabelle gespeichert
- Wird häufig in Form einer JSON-Serialisierung verwendet
- In JPA über Converter
- Beispiel:
  - Name einer Person soll in DB als „<Vorname>|<Nachname>“ gespeichert werden

VO serialisieren
am Beispiel einer JPA-Entity

```
@Entity
public class Customer{

    private Long id;

    @Convert(
        converter = NameConverter.class
    )
    private Name name;
}
```

```
public final class Name {

    private final String firstName;
    private final String lastName;

    public Name(String firstName, String lastName) {}
}
```

VO serialisieren am Beispiel einer JPA-Entity

```
@Converter
public class NameConverter implements AttributeConverter<Name, String>{

    private static final String DELIMITER = "|";

    @Override
    public String convertToDatabaseColumn(final Name domainName) {
        return domainName.getFirstName() + DELIMITER + domainName.getLastName();
    }

    @Override
    public Name convertToEntityAttribute(final String dbName) {
        String[] nameComponents = dbName.split(DELIMITER);
        return new Name(nameComponents[0], nameComponents[1]);
    }
}
```

ID	Name	…
42	Max|Muster	…

VO serialisieren

Vorteile:

- Komplexe Objekte können gespeichert werden
- 1:n-Beziehungen möglich (Set, List)

Nachteile:

- DB wird ggf. unlesbar oder schwerer verständlich
- Ggf. Verletzung der 1NF („jedes Attribut einer Relation muss einen atomaren Wertebereich haben“)
- Queries über VO schwierig oder nicht möglich
- Aufwändiger

VO als DB-Entity in eigener Tabelle speichern

- Das VO wird aus Sicht der Persistenz-Schicht wie eine eigenständige Entity behandelt
- ->das VO erhält also eine ID
- Die ID sollte innerhalb der Domäne „versteckt“ werden
- Die ID existiert rein zum Zwecke der Speicherung des VO in der DB
- Finale Felder müssen in parameterlosem Konstruktor initialisiert werden

VO als DB-Entity in eigener Tabelle speichern

```java
@Entity
public final class Address {

    @Id
    @GeneratedValue
    private long id;

    @Column(name = "street")
    private final String street;

    @Column(name = "zip_code")
    private final String zipCode;

    @Column(name = "city")
    private final String city;

    @SuppressWarnings("unused")
    private Address() {
        //Zugeständnis an JPA; JPA erwartet einen parameterlosen Konstruktor, finale Felder müssen aber im
        //Konstruktor initialisiert werden
        street = null;
        zipCode = null;
        city = null;
    }
    // …weiterer code…
}
```

VO als DB-Entity in eigener Tabelle speichern

Vorteile:

- Einfach zu implementieren
- Queries über Attribute des VO möglich
- Ermöglicht 1:n Beziehungen (Set, List)

Nachteile:

- Verschleiert Natur eines VO durch ID
- Gefahr, dass mehrere Entities auf dasselbe VO verweisen