

Bachelor of Science (BSc) in Informatik
Modul Software-Entwicklung 1 (SWEN1)

V3 – Persistenz

SWEN1/PM3 Team:
R. Ferri (feit), D. Liebhart (lieh), K. Bleisch (bles), G. Wyder (wydg)

Ausgabe: HS24

Um was geht es?

- Wie kann ich meine Java Objekte dauerhaft speichern?
- Welche Arten von Datenspeicherung gibt es?
- Welche Design Patterns stehen für die Realisierung von Persistenz in einer Applikation zur Verfügung?
- Wie kann ich mit Hilfe von den Java APIs JDBC (Java Database Connectivity) und JPA (Java Persistence API) meine Objekte dauerhaft in einer Datenbank speichern?



Lernziele LE 12 – Persistenz

- Sie sind in der Lage
 - die Varianten der Datenspeicherung zu nennen,
 - die unterschiedlichen Design Patterns für die Persistenz zu erklären,
 - mit Hilfe des Design Patterns DAO (Data Access Object) und JDBC eine Persistenz in Java umzusetzen,
 - mit JPA ein Objekt-Relationales-Mapping (O/R-Mapping) in Java anzuwenden.

Agenda

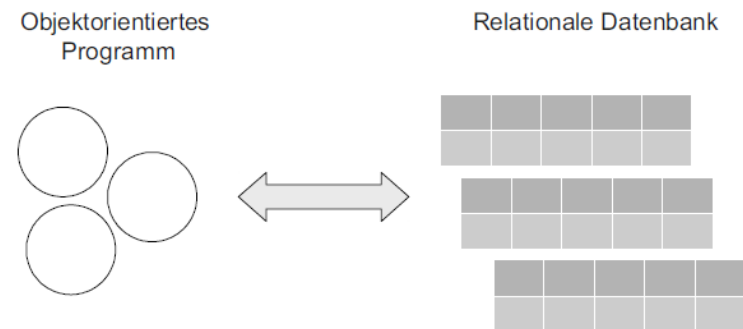
- 1. Einführung in Persistenz**
2. Design-Optionen für Persistenz
3. Persistenz mit JDBC
4. O/R-Mapping mit DAO
5. O/R-Mapping mit JPA
6. Wrap-up und Ausblick

Problemstellung Persistenz (1/2)

- In vielen Applikationen müssen unterschiedliche **Daten** verarbeitet, verwaltet und **dauerhaft**, d.h. **über das Programmende** hinaus gesichert werden.
- Letzteres bezeichnet man als **Persistenz**.
- Die dauerhafte Speicherung erfolgt in **Datenbankmanagementsystemen** (DBMS).
- Übliche Datenbanksysteme sind sogenannte **Relationale Datenbanksysteme (RDBMS)** und sogenannte **NoSQL-Datenbanken**.
- NoSQL-Datenbanken speichern Daten **ohne fixes Schema** und in **verschiedenen Formaten** (Dokument Stores, Key-Value Stores, Graph DB, ...).

Problemstellung Persistenz (2/2)

- Die **Abbildung zwischen Objekten und Datensätzen in Tabellen** einer relationalen Datenbank wird auch **als O/R-Mapping** (Object Relational Mapping, ORM) bezeichnet.
- Verhältnismässig **viel Java-Code** wird benötigt, um die Datensätze des Ergebnisses zu verarbeiten und in **Java-Objekte** zu transformieren.
- Es besteht ein **Strukturbruch** (engl. Impedance Mismatch) aufgrund der unterschiedlichen Repräsentationsformen von Daten (flache Tabellenstruktur – Java-Objekte).



Aufgabe 12.1 (5')

Diskutieren Sie in Murmelgruppen folgende Frage:

- Was ist aktuell die vorherrschende Technologie zum Speichern von Daten im Enterprise-Umfeld?
- Recherchieren Sie dazu unter <https://db-engines.com/en/ranking>.
- Was sind die Gründe für dieses Ranking?

Agenda

1. Einführung in Persistenz
- 2. Design-Optionen für Persistenz**
3. Persistenz mit JDBC
4. O/R-Mapper mit DAO
5. O/R-Mapper mit JPA
6. Wrap-up und Ausblick

Herausforderung: Der O/R-Mismatch (1/2)

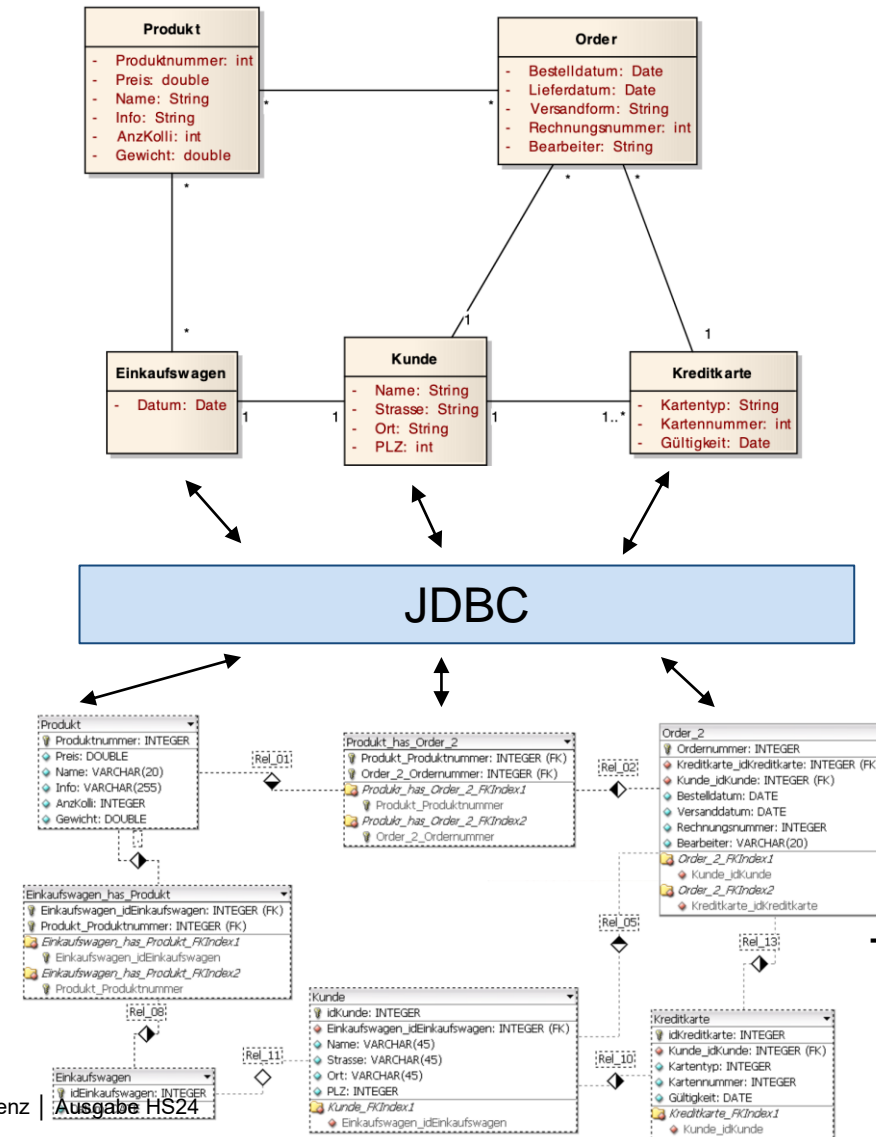
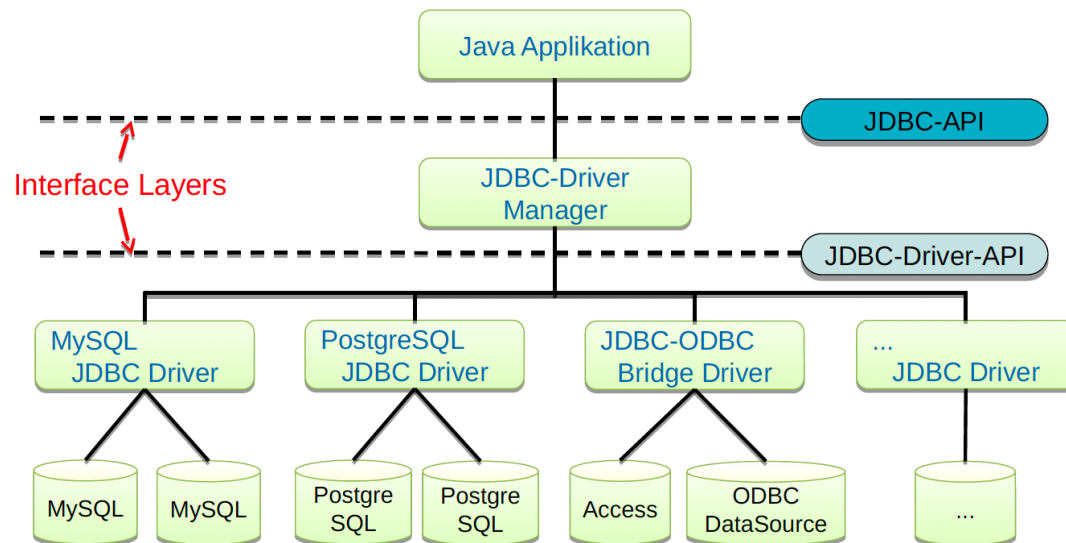
- Der **O/R-Mismatch** ist ein Fakt.
- Der **O/R-Mismatch** folgt aus konzeptionellen Unterschieden der zugrundeliegenden Technologien.
- Es gibt viele verschiedene Möglichkeiten (Patterns) den O/R-Mismatch zu überwinden.
- Active Record, O/R-Mapping resp. O/R-Mapping Frameworks oder Repositories (aus Domain Driven Design, DDD) sind ein möglicher Lösungsansatz.

Herausforderung: Der O/R-Mismatch (2/2)

- Typen-Systeme
 - Null
 - Datum/Zeit
- Abbildung von Beziehungen
 - Richtung
 - Mehrfachbeziehungen
- Vererbung
- Identität
 - Objekte haben eine implizite Identität
 - Relationen haben eine explizite Identität (Primary Key)
- Transaktionen

JDBC: Java Database Connectivity (1997)

- JDBC verbindet die Objektwelt mit der relationalen Datenbankwelt
 - Herausforderung: Objekte vs. Tabellen,
 - Verschiedene Datentypen etc.
 - Die Programmierung ist aufwändig



Objekte

Tabellen

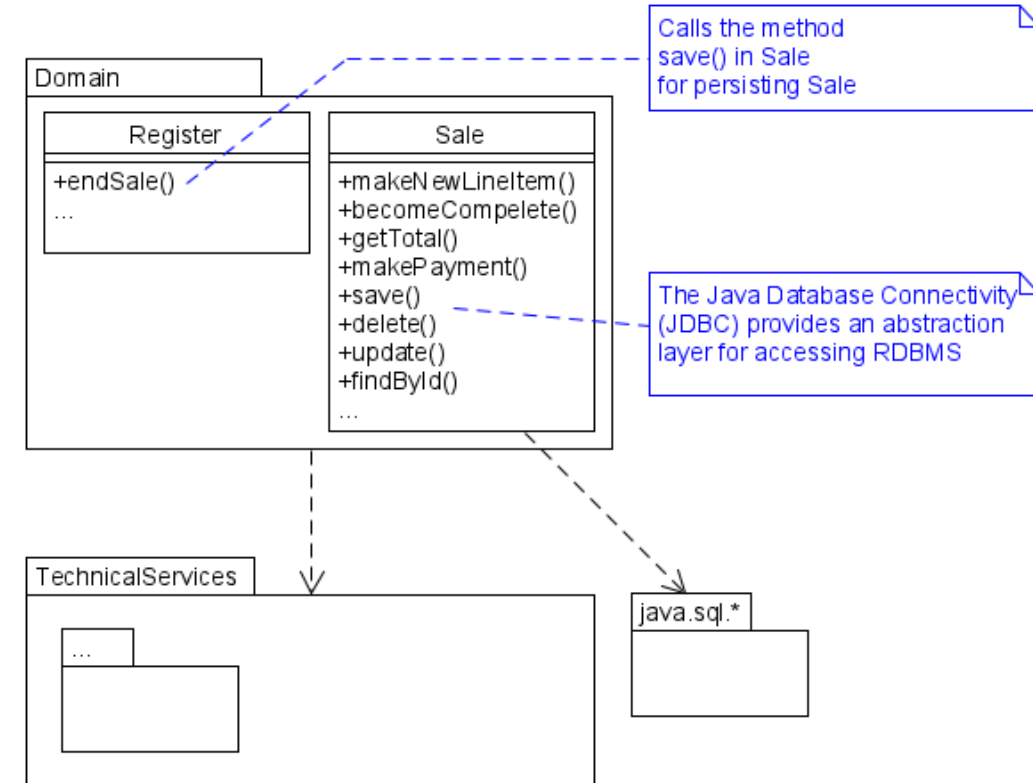
Design Pattern für Persistenz

Für eine Persistenz-Strategie muss eine Entscheidung getroffen werden, wo die Zuordnung (**Mapping**) zwischen Objekten und Tabellen stattfinden soll:

- **Active Record (Anti Pattern)**: Jede Entität ist selber dafür zuständig
- **Data Access Object (DAO)**: Abstrahiert und kapselt den Zugriff auf die Datenquelle
- **O/R Data Mapper**: Separate Klasse für das Mapping oder Einsatz eines ORM

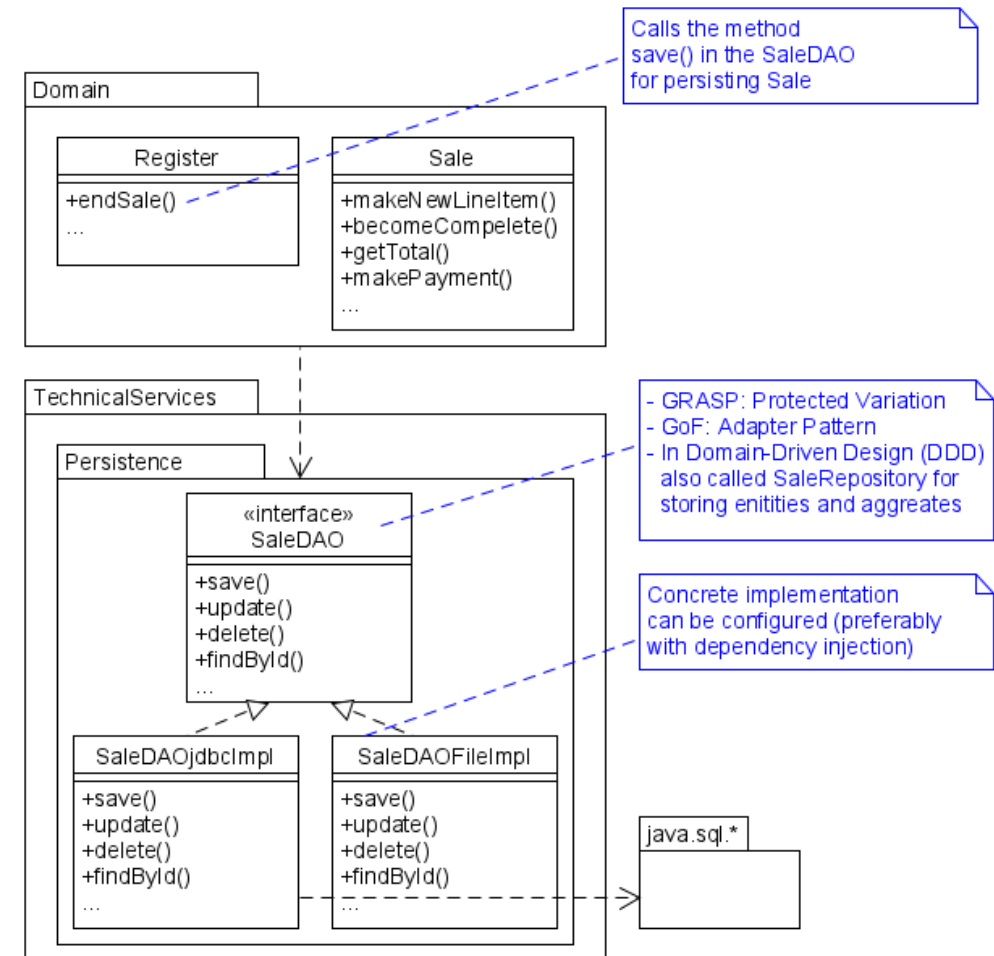
Active Record -> Anti Pattern

- Zugriffscode auf Datenbank ist **in** der **Domänenklasse**
- Wrapper für eine Zeile einer Datenbanktabelle
- Spiegelt die Datenbankstruktur
- Enthält Daten und Verhalten
- Fachlichkeit und Technik alles in einer Klasse (GRASP: Information Expert?)
- **Schlechte** Testbarkeit der Domänenlogik ohne Datenbank
- **Schlechte** Wartbarkeit und Erweiterbarkeit (No separation of concerns!)



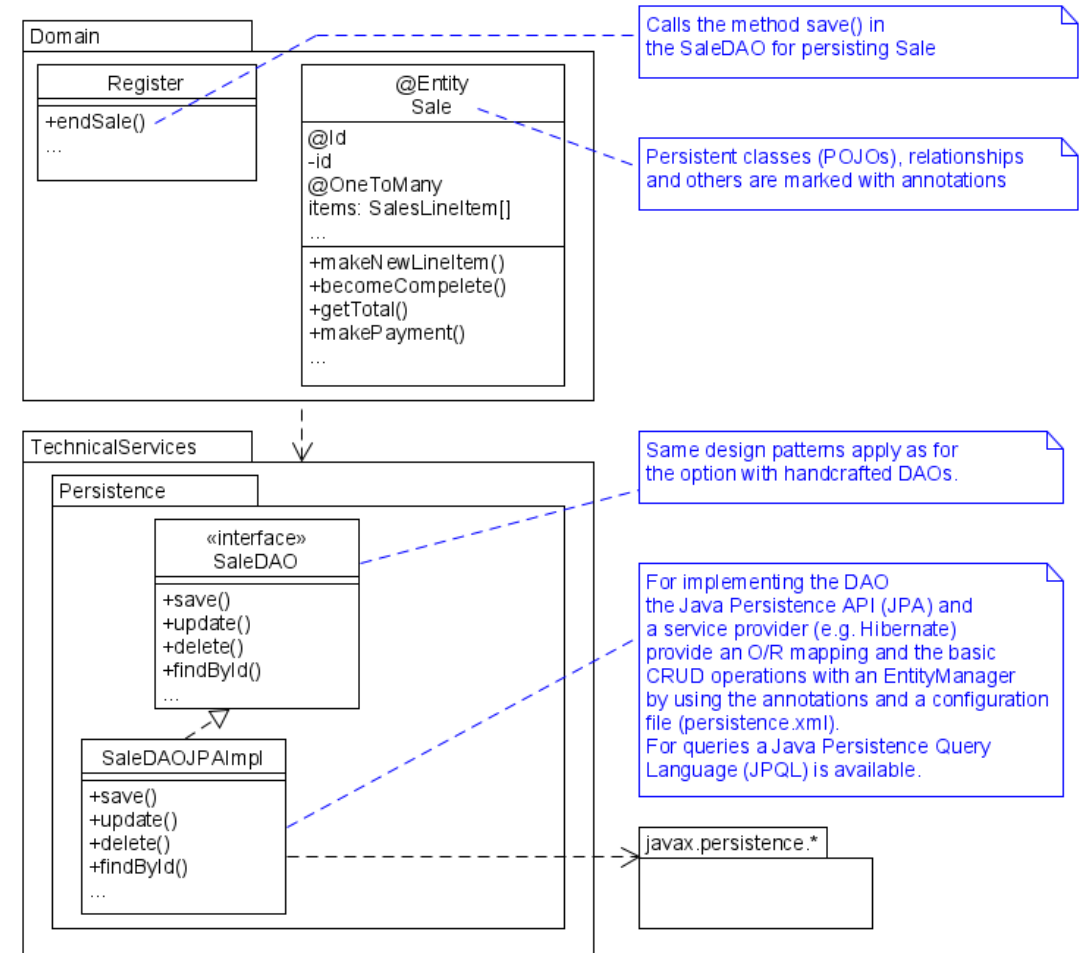
O/R-Mapping von Hand mit Hilfe eines Data Access Objects (DAO)

- Trennung von Fachlichkeit und Technik
(Domänenklasse hat hohe Kohäsion)
- Gute Testbarkeit und Mocking der Persistenz
- Bevorzugtes Design ohne Einsatz eines O/R-Mappers



Verwendung eines O/R-Mappers (JPA mit Hibernate/Eclipselink o.a. Framework)

- Viel weniger Aufwand bzw. Code und standardisierte Schnittstelle
- Trennung von Fachlichkeit und Technik (Domänenklasse hat hohe Kohäsion)
- Gute Testbarkeit und Mocking der Persistenz
- DAO ist auch beim Einsatz von JPA empfehlenswert (Trennung von Fachlichkeit und Technik) - JPA könnte aber auch ohne DAO verwendet werden



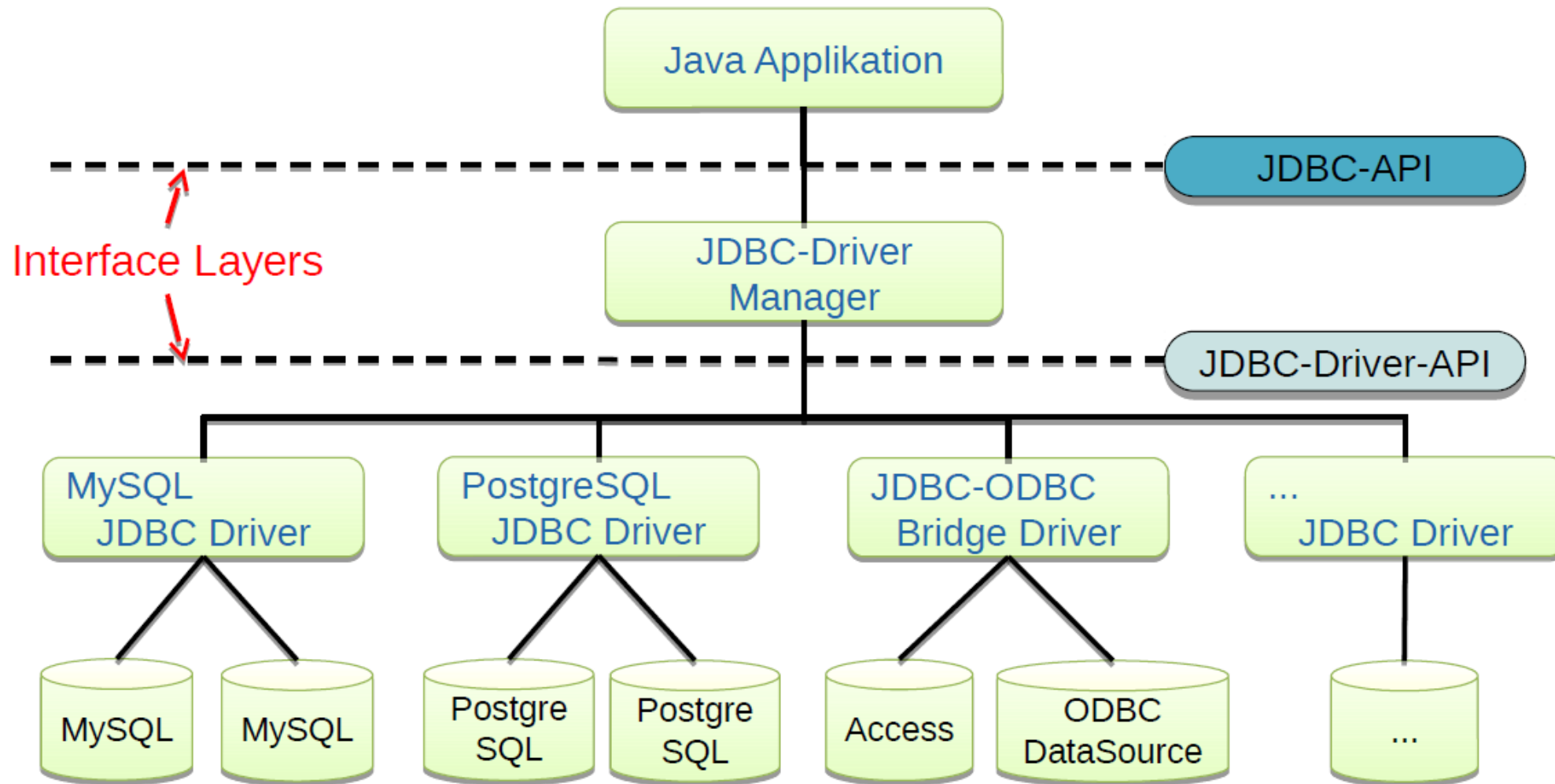
Agenda

1. Einführung in Persistenz
2. Design-Optionen für Persistenz
3. **Persistenz mit JDBC**
4. O/R-Mapper mit DAO
5. O/R-Mapper mit JPA
6. Wrap-up und Ausblick

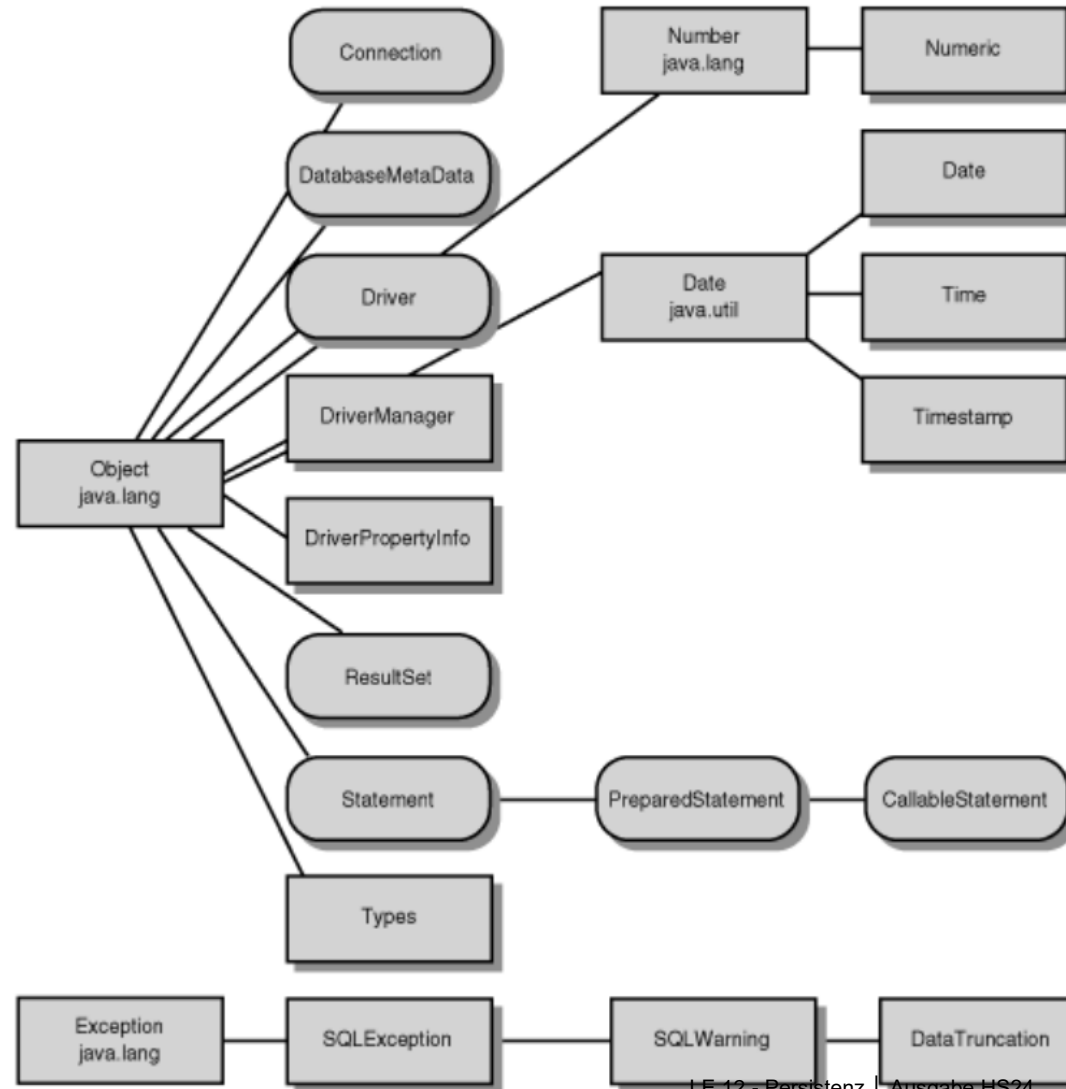
Was genau ist JDBC?

- **JDBC** = Java Data Base Connectivity
- Standardisierte Schnittstelle, um auf relationale Datenbanken mit Hilfe von SQL zuzugreifen
- Cross-Plattform und DB-independent
- JDBC-API ist Teil der Java-Plattform seit JDK1.1 (1997)
- Die aktuelle Version ist 4.2

JDBC API: Basic -Architecture



JDBC-API: Interfaces and Classes



Anwendung von JDBC

Basisanweisungen:

1. Install and load JDBC driver
2. Connect to SQL database
3. Execute SQL statements
4. Process query results
5. Commit or Rollback DB updates
6. Close Connection to database

```
import java.sql.*;

public class DbTest {
    public static void main(String[] args)
        throws ClassNotFoundException, SQLException {
        Connection con = DriverManager.getConnection(
            "jdbc:postgresql://test.zhaw.ch/testdb",
            "user", "password");
        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery(
            "SELECT * FROM test ORDER BY name");
        while (rs.next()) {
            System.out.println(
                "Column 1 contains '" +
                rs.getString(2) + "'");
        }
        con.close();
    }
}
```

Agenda

1. Einführung in Persistenz
2. Design-Optionen für Persistenz
3. Persistenz mit JDBC
4. **O/R-Mapping mit DAO**
5. O/R-Mapping mit JPA
6. Wrap-up und Ausblick

O/R-Mapping Pattern

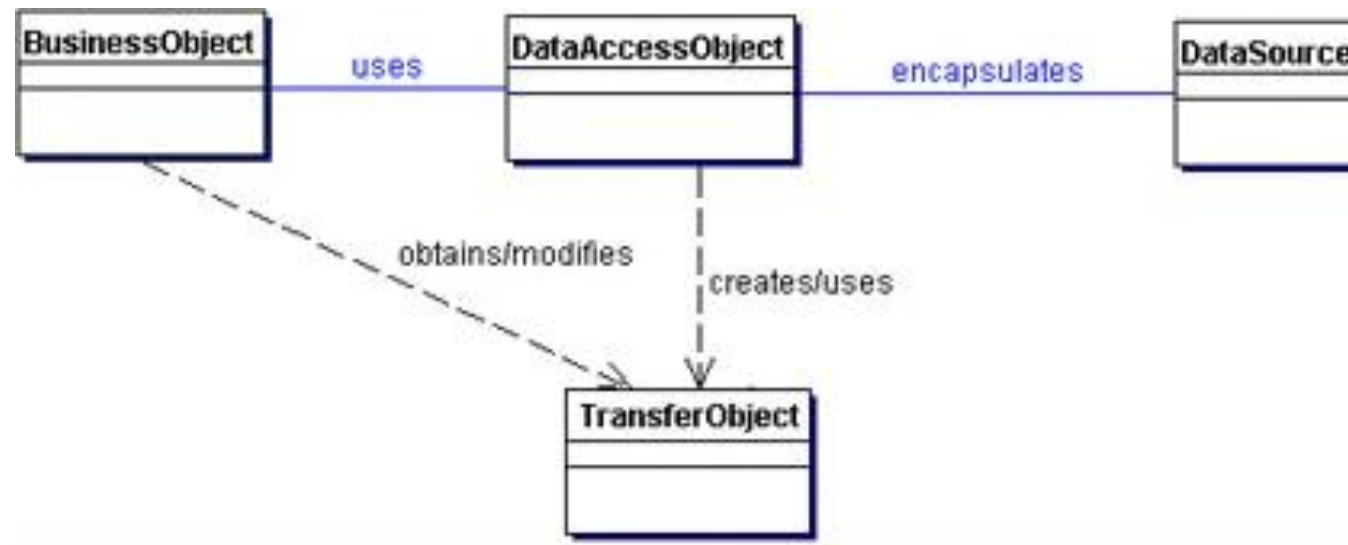
Es sollen beide Varianten des O/R-Mapper Patterns anhand eines praktischen Beispiels betrachtet werden:

- **DAO** (Data Access Object) **ohne** ein ORM (Object Relational Mapper)
- Umsetzung von DAO mit Hilfe von **JPA** (Java Persistence API)

DAO - Data Access Object Pattern

- Das Artikel-Objekt repräsentiert das Domain-Model-Objekt.
- Die Verbindung zur Datenbank wird durch das DAO sichergestellt.
 - Enthält die üblichen CRUD-Methoden wie create, read, update und delete.
 - Kann auch Methoden enthalten wie findAll, findByName, findById um eine Kollektion von Daten aus der Datenbank abzufragen.

DAO - Data Access Object Pattern



Sun Developer Network - Core J2EE Patterns

<http://www.oracle.com/technetwork/java/dataaccessobject-138824.html>

Das TransferObject aka. DTO kann zusätzlich für den Transport der Daten in einem verteilten System verwendet werden.

Beispiel Article und ArticleDAO

Business Object

```
public class Article {  
    private long id;  
    private String name;  
    private float price;  
    public long getId(){  
        return id;  
    }  
    public void setId(long id){  
        this id = id  
    };  
    ...  
}
```

Data Access Object (DAO)

```
//Interface to be implemented by all ArticleDAOs  
public interface ArticleDAO {  
    public void insert(Article item);  
    public void update(Article item);  
    public void delete(Article item);  
    public Article findById(int id);  
    public Collection<Article> findAll();  
    public Collection<Article> findByName (String name);  
    public Collection<Article> findByPrice (float price);  
    ...  
}
```

Agenda

1. Einführung in Persistenz
2. Design-Optionen für Persistenz
3. Persistenz mit JDBC
4. O/R-Mapping mit DAO
- 5. O/R-Mapping mit JPA**
6. Wrap-up und Ausblick

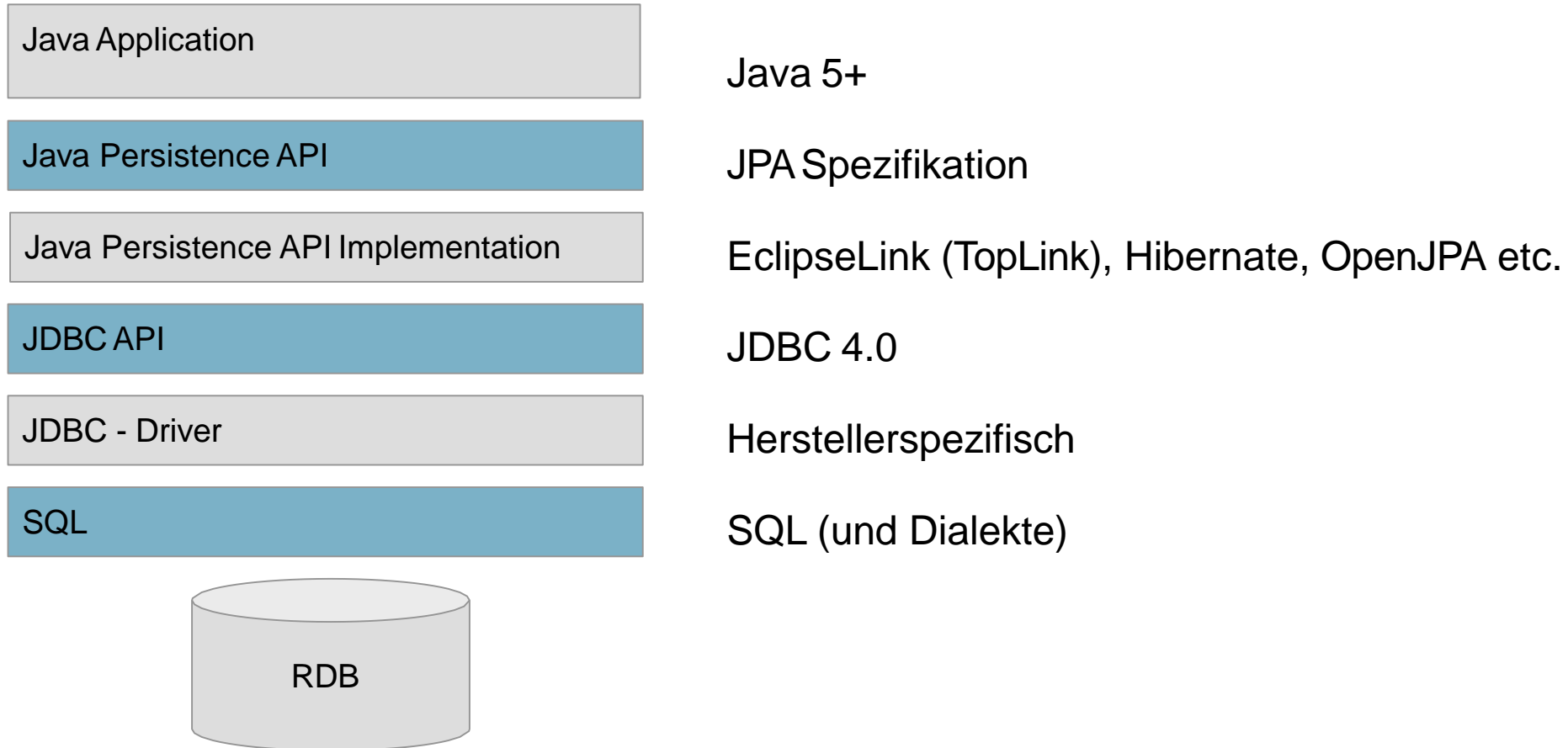
Versprechen von automatischem O/R-Mapping

- Die Applikation wird von der DB entkoppelt
 - Applikationsentwickler muss **kein** SQL beherrschen.
 - Das relationale Modell der Datenbank hat **keinen** Einfluss auf das OO-Design.
- Automatische Persistenz
 - **Automatisierte** Abbildung der Objekte in die relationalen Strukturen.
 - Die Applikationsentwickler muss sich **nicht** um die «low-level»-Details kümmern.
- Transparente Persistenz / Persistence Ignorance
 - Die Klassen des Domain-Models **wissen nicht**, dass sie persistiert und geladen werden können und haben keine Abhängigkeit zur Persistenz-Infrastruktur.
- JPA ist ein Java Standard für O/R-Mapping
 - Verschiedene Implementationen, Hibernate vermutlich die bekannteste

JPA (Java Persistence API) Überblick

- Es folgt eine kurze, unvollständige Auflistung der wichtigsten Konzepte von JPA.
- **Starke** Entkopplung der Anwendungslogik von der (relationalen) Datenbank.
- Die **Domänenklassen** sind ganz **normale** Java Klassen (**POJO**)
 - Ausser Annotationen enthalten Sie keinen JPA spezifischen Code.
- **Referenzen**
 - Werden entweder mit der referenzierenden Klasse (eager loading) oder erst, wenn die Referenz gebraucht wird (lazy loading), geladen.
 - Referenzen können direkt traversiert werden, JPA erledigt das Laden des referenzierten Objekts im Hintergrund.
- **Transaktionshandling** und das Absetzen von **Queries** müssen über JPA spezifische Klassen abgewickelt werden.
 - EntityManagerFactory, EntityManager, EntityTransaction

Technologie-Stack



Entity Metadata

- Kennzeichnung mit Annotation `@Entity` oder Mapping mit XML
- Klasse kann Basisklasse oder abgeleitet sein
- Klasse kann abstrakt oder konkret sein
- Serialisierbarkeit ist bezüglich Persistenz **nicht** erforderlich

Beispiel Entity

- Minimale Anforderung an eine Entity

```
@Entity
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String name;
    private String lastName;
    ..
    ..
}
```

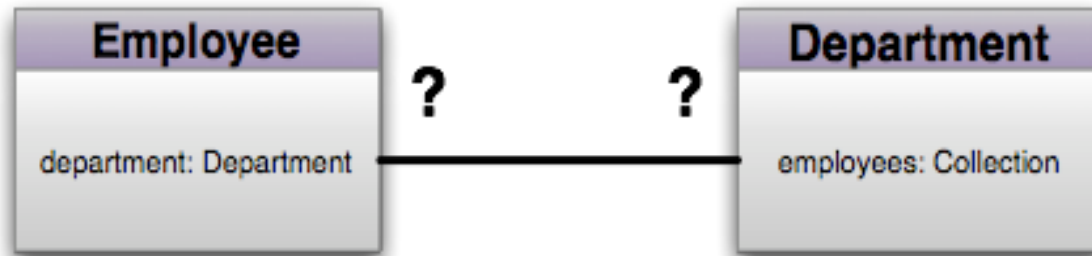
Primärschlüssel

- Primärschlüssel können in Zusammenarbeit mit der Datenbank generiert werden. Strategien sind **Identity**, **Table**, **Sequence** und **Auto**

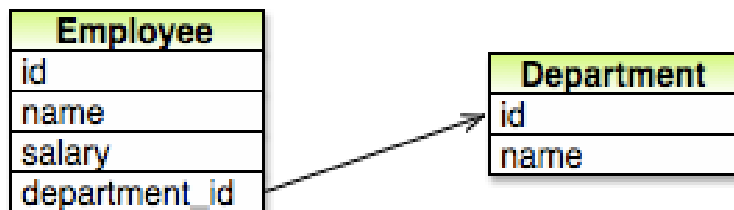
```
@Entity public class Employee {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    public Integer id;  
}
```

```
public class Employee {  
    @TableGenerator(name = "Emp_Gen", table = "ID_GEN", pkColumnName = "GEN_NAME",  
                    valueColumnName = "GEN_VAL")  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.TABLE, generator = "Emp_Gen")  
    private int id;  
  
    ...  
}
```


Parent-Child Beziehung



- Mapping des Klassenmodells auf das DB-Schema mittels JPA: Metadata ist erforderlich.
 - Je nach Klassenmodell wird entweder eine **many-to-one** Beziehung oder eine **one-to-many** Beziehung gemappt.
 - Falls beide Richtungen gemappt werden sollen, so muss definiert werden, dass für beide derselbe **Foreign-Key** zugrunde liegt.

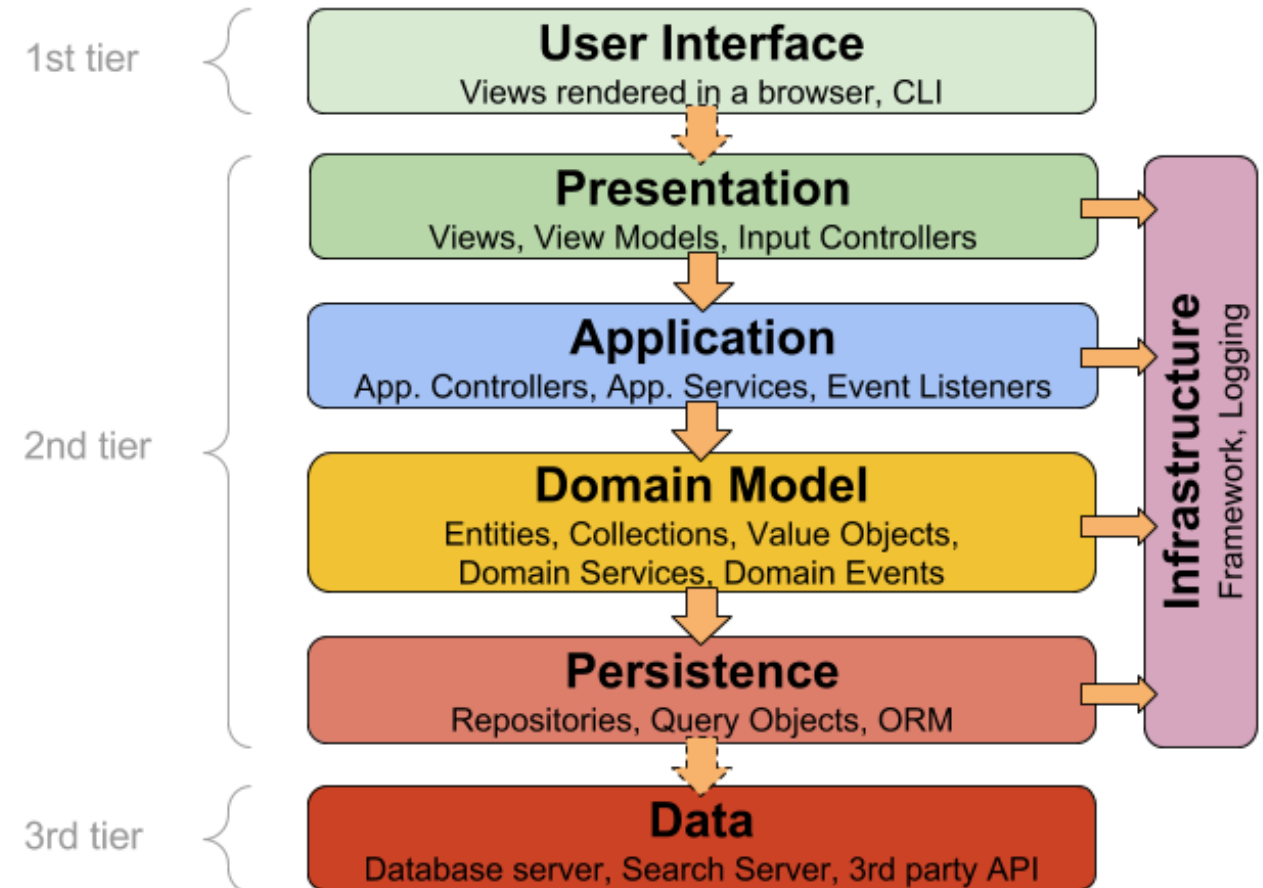


Ausblick Design Pattern Repository

- Ein System mit einem **komplexen Domänen-Model** profitiert wie vorher beschrieben von einer Data-Mapper-Schicht (mit JPA und DAOs), um die Details des Datenbankzugriffcodes zu isolieren.
- Eine **zusätzliche Abstraktionsschicht oberhalb des Data-Mappers** kann helfen um die Konstruktion von Datenbank-Abfragen (Queries) an einem Ort zu konzentrieren.
- Diese zusätzliche Schicht wird um so wichtiger je mehr Domänen-Klassen vorhanden sind, die viele Zugriffe auf die Datenbank vornehmen.
- Die zusätzliche Schicht wird als **Repository** bezeichnet
- Das Konzept stammt aus Domain Driven Design, DDD (Eric Evans).
 - Wird in den Wahlpflichtmodulen ASE1/2 anhand von Spring Data behandelt.

Design Pattern Repository: Schichtenmodell

- 3-Tier Architecture
- Persistenz kann mittels Repositories umgesetzt werden



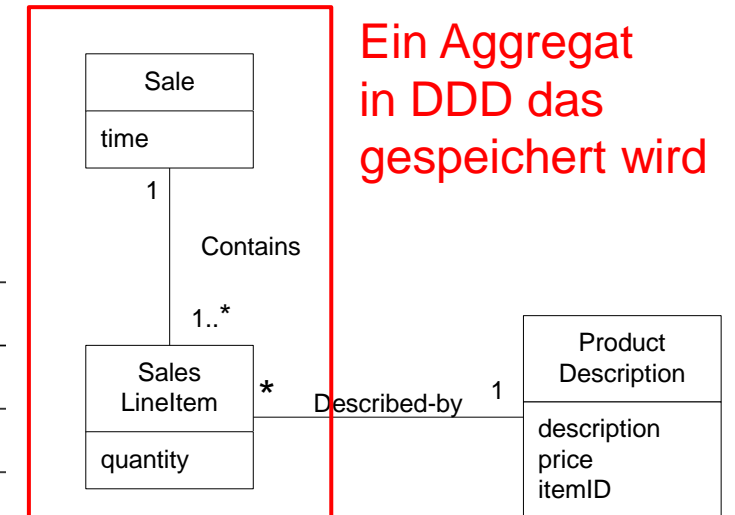
Idee und Beispiel Repository Pattern

- Eine Repository vermittelt zwischen Domänen- und Data-Mapping Schicht

```

9  /**
10 * The GRASP controller for the use case process sale.
11 */
12 public class ProcessSaleHandler {
13     private ProductDescriptionRepository catalog;
14     private SaleRepository saleRepository;
15     private Sale currentSale;
16
17     public ProcessSaleHandler(ProductDescriptionRepository catalog, SaleRepository saleRepository) {
18
19     }
20
21     public void makeNewSale() {
22
23     }
24
25     public void enterItem(String id, int quantity) {
26
27     }
28
29     public Money getTotalOfSale() {
30         @Transactional
31         public void endSale() {
32             assert(currentSale != null && !currentSale.isComplete());
33             this.currentSale.becomeComplete();
34             this.saleRepository.save(currentSale);
35         }
36
37     }
38
39     public Money getTotalWithTaxesOfSale() {
40
41     }
42
43     public void makePayment() {
44
45     }
46
47     }
48
49 }

```



```

1  /**
2   * Repository for Sale.
3   * An implementation of CRUD and common search methods
4   * is automatically generated by Spring Data.
5   */
6   @Repository
7   public interface SaleRepository extends CrudRepository<Sale, String> {
8       public List<Sale> findOrderByDateTime();
9       public List<Sale> findByDateTime(final LocalDateTime dateTime);
10  }

```

Agenda

1. Einführung in Persistenz
2. Design-Optionen für Persistenz
3. Persistenz mit JDBC
4. O/R-Mapping mit DAO
5. O/R-Mapping mit JPA
6. **Wrap-up und Ausblick**

Wrap-up

- Viele Applikationen verlangen, dass **Daten dauerhaft gesichert** werden müssen nach dem Programmende.
- Bei **kleineren Applikationen** kann diese Persistenz auch **selber ausprogrammiert** werden.
- Dabei sollte aber das Design Pattern **Data Access Object (DAO)** oder **Repository** angewendet werden.
- Für **grössere Applikationen** werden heute sogenannte **O/R-Mapper** eingesetzt.
- Java bietet mit dem **Java Persistence API (JPA)** eine **standardisierte Schnittstelle** für das O/R-Mapping, für die es viele Provider gibt (z.B. Hibernate).

Ausblick

- In der nächsten Lerneinheit werden wir:
 - das Thema Design von Frameworks vertiefen.

Quellenverzeichnis

- [1] Oracle: Core J2EE Patterns - Data Access Object,
<https://www.oracle.com/java/technologies/dataaccessobject.html>
- [2] Oracle: Core J2EE Patterns - JPA,
<https://www.oracle.com/java/technologies/persistence-jsp.html>
- [3] Hibernate: Documentation, <https://hibernate.org/orm/documentation/5.4/>
- [4] Ranking Database: <https://db-engines.com/en/ranking>