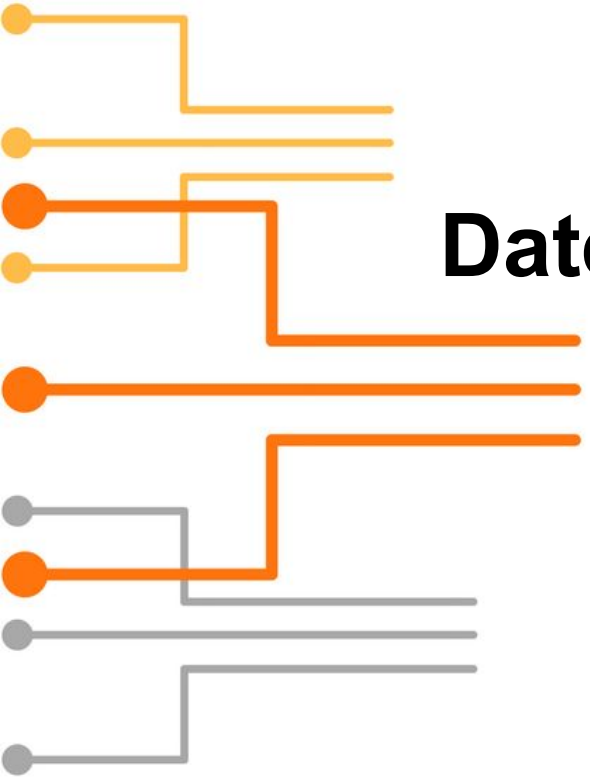


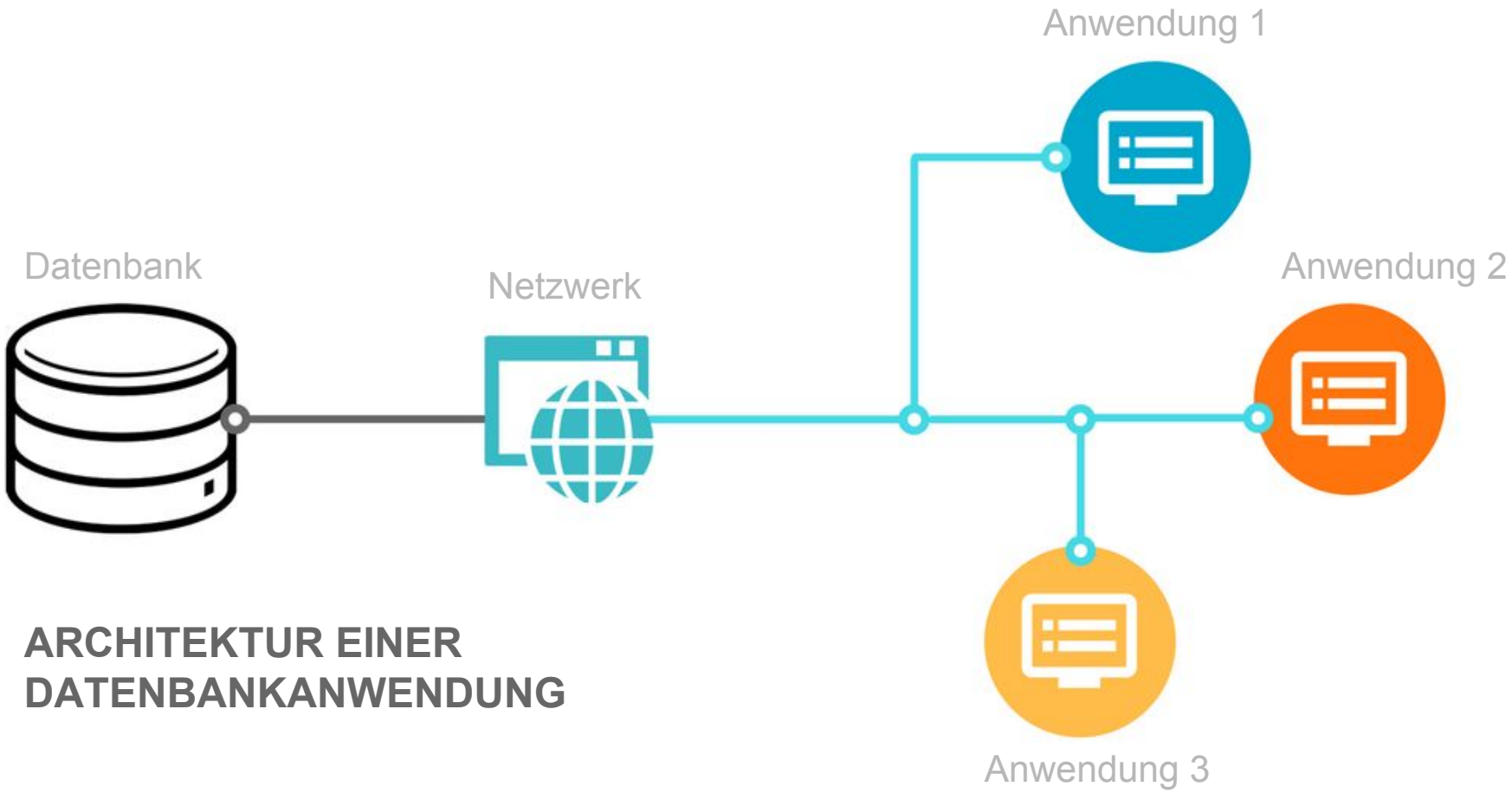
Datenbankanwendungen

mit Java

Architektur
Konzepte
Technologien

Sonja Kopf, MSc

so@datenautomaten.nu

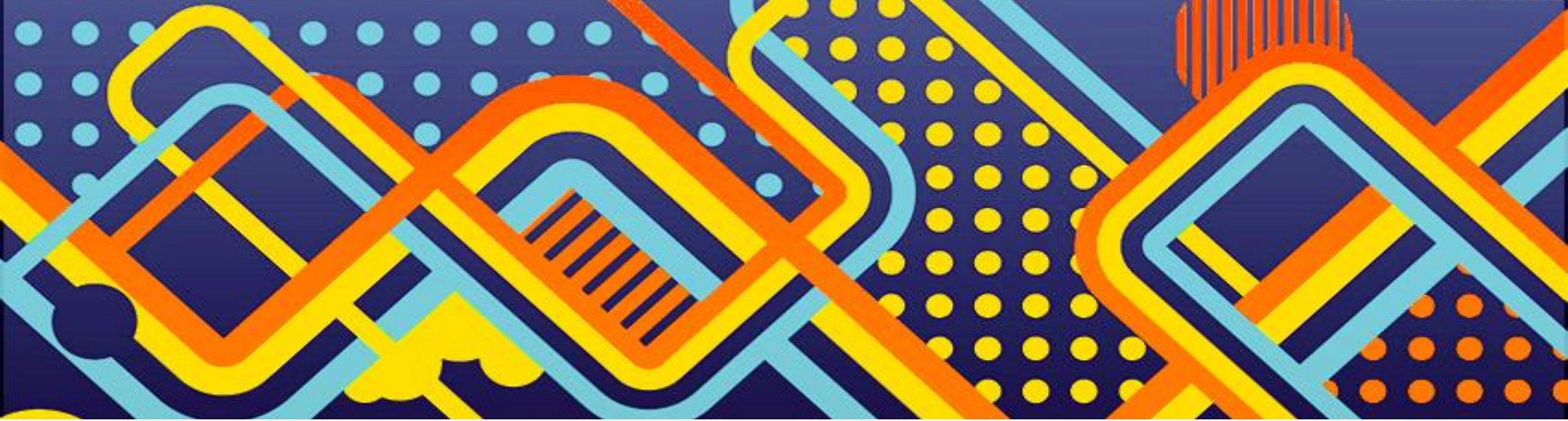


ARCHITEKTUR EINER DATENBANKANWENDUNG

Herausforderungen - was müssen wir tun?

- Verbindung zur Datenbank
- Abfragen erstellen
- Ergebnismengen abfragen
- Ergebnismengen umwandeln in Objekte
- Daten persistieren

- Wartbarer Code - wenig Code
- Abstraktion der Datenbankebene
- Paradigm Mismatch - Klassenmodell <-> Datenbankmodell

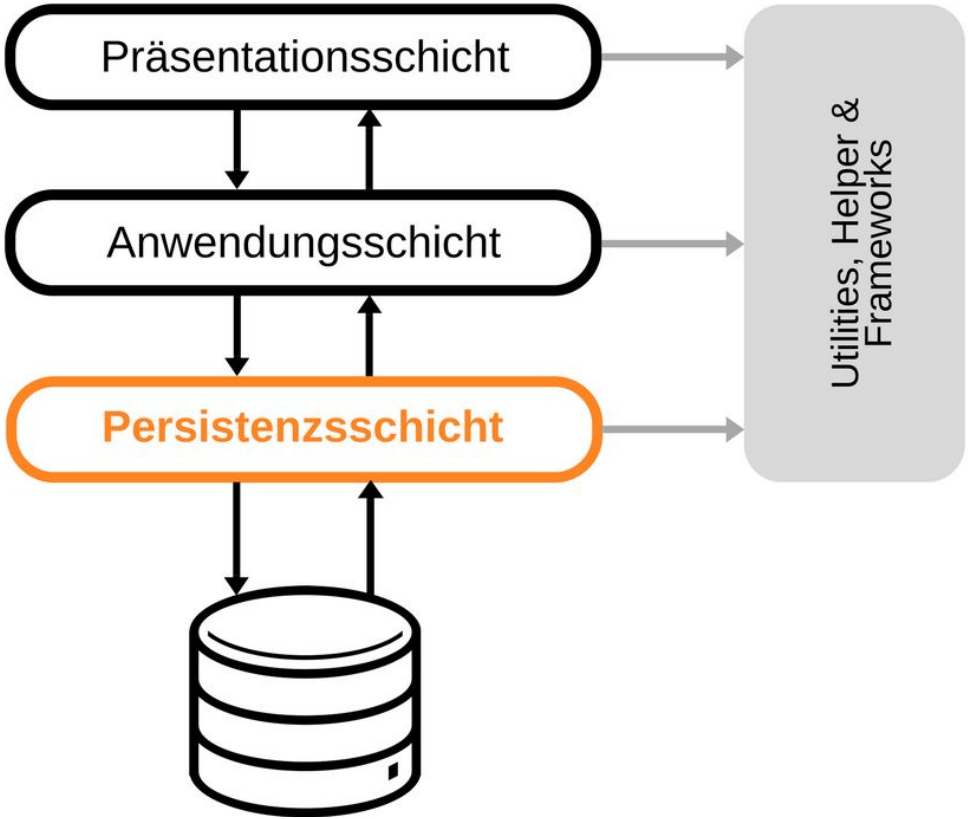


Patterns

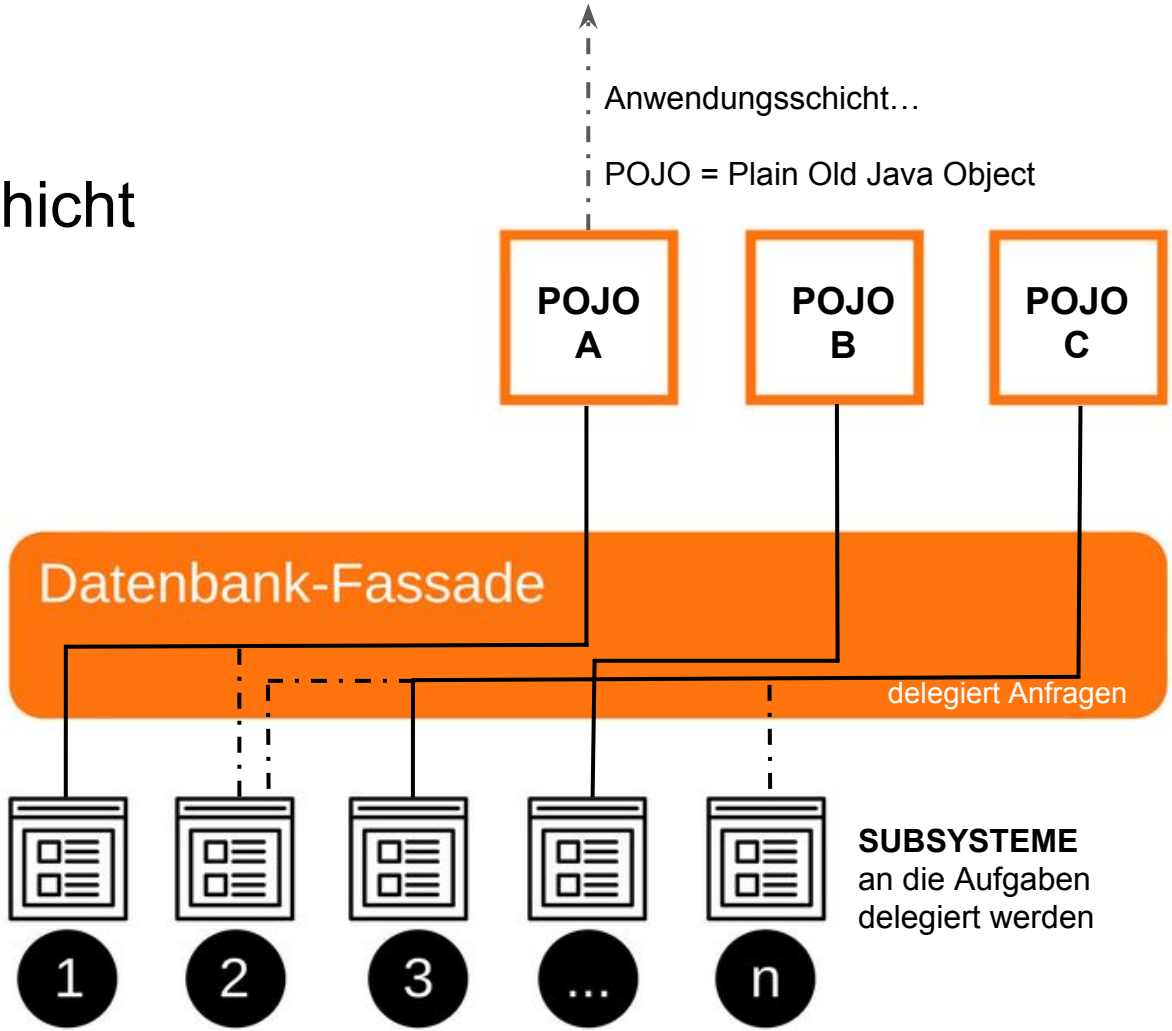
- Codekonstrukte und Konzepte
- Lösungsansätze für bestimmte Problemstellungen
- Wartbare und wieder verwendbare Lösungen
- Standardisierte Code-Modelle und Entwurfsmuster

Schichtenmodell

- Jede Schicht kennt nur ihre direkten Nachbarn
- und kommuniziert nur mit ihren Nachbarn



Persistenzschicht



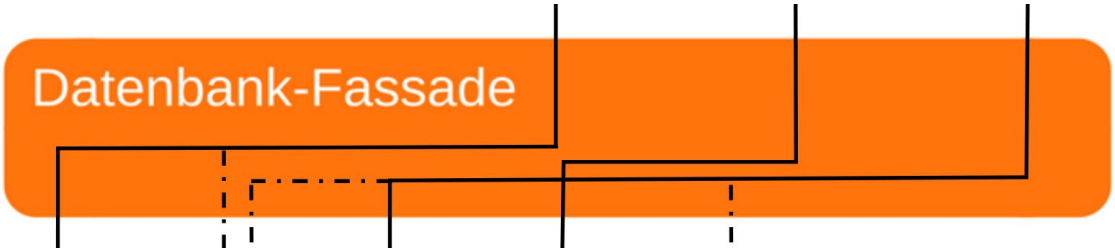
POJO
Plain Old
Java Object

- Datencontainer für Entitäten der DB
- (De-)Materialisierung in der Persistenzschicht
- Keine Anwendungslogik!

- Keine Abhängigkeiten zu anderen Klassen oder Interfaces
- Kann als Basisklasse für Business-Objekt verwendet werden

Beispiel

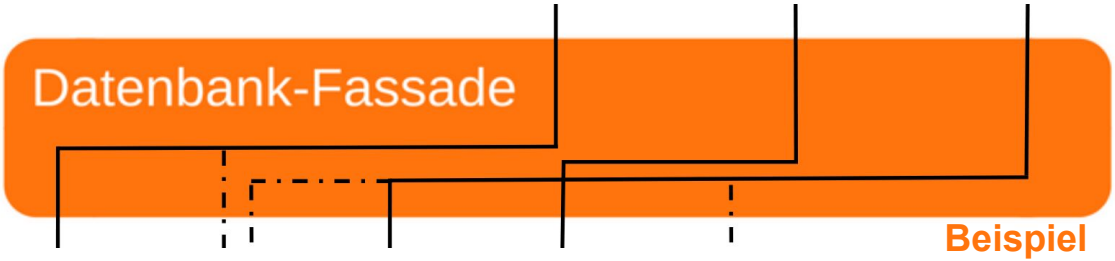
```
public class Book {

	private String name;
	private String author;

	public String getName() { return name; }
	public void setName(String name) { this.name = name; }

	public String getAuthor() { return author; }
	public void setAuthor(String author) { this.author = author; }

}
```



- einfach zu verwendende Schnittstelle zu Subsystem(en)

- verringert die Komplexität
- reduziert direkte Abhängigkeit zu Subsystemen
- organisiert die Kommunikation zwischen Subsystemen

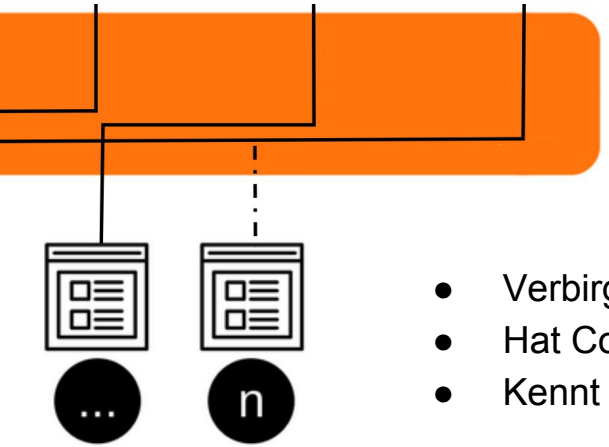
- Fassade weiss welche Subsysteme für welche Arten von Anfragen/Aufgaben zuständig sind
- → delegiert an diese
- Subsysteme wissen nichts von der Fassade
- Fassade *kann* static oder Singleton sein



Beispiel

```
public class ShapeMaker {

      public drawCircle();

      public drawRectangle();

      public drawSquare();

}
```

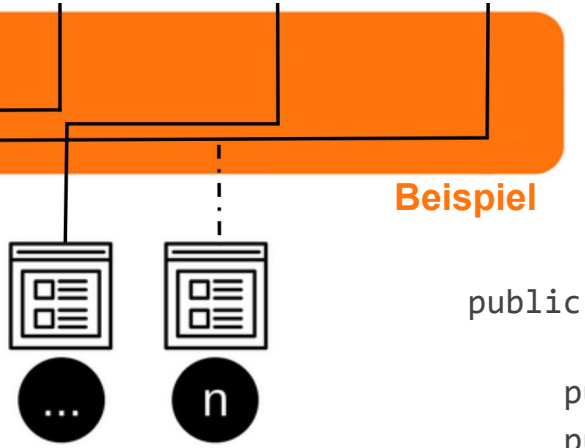
Die Fassade stellt eine einfache Schnittstelle für den Zugriff auf komplexe Systeme zur Verfügung. Sie kapselt wichtiges, versteckt unwichtiges und vereinfacht die Handhabung.



Broker

- Verbirgt sich als Subsystem hinter der Fassade
- Hat Connections
- Kennt Details

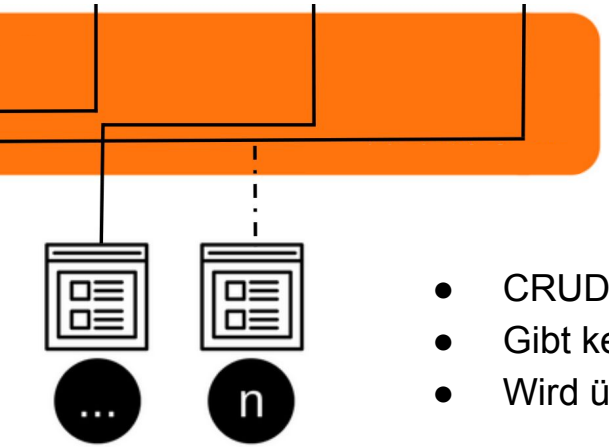
- Ein Broker pro Objekttyp
- Materialisiert und dematerialisiert POJOs
- Kommuniziert ausschließlich mit der Fassade



Broker

Beispiel

```
public class BookBroker {

     public void insert(Book b);
     public void delete(Book b);
     public void update(Book b);
     public Book get(intid);
     public List<Book> getAll();

     public void save(Book b);
     public void validate(Book b);
     public bool exists(Book b);

}
```



DAO

- CRUD-Funktionalität
- Gibt keine Details über Persistenz preis
- Wird über Interface definiert

Ein DAO ist ein Objekt, das für einen bestimmten Objekttyp (des Datenmodells) die Funktionalitäten zum Erstellen, Lesen, Laden und Speichern bietet.

MOJO
Model Java Object

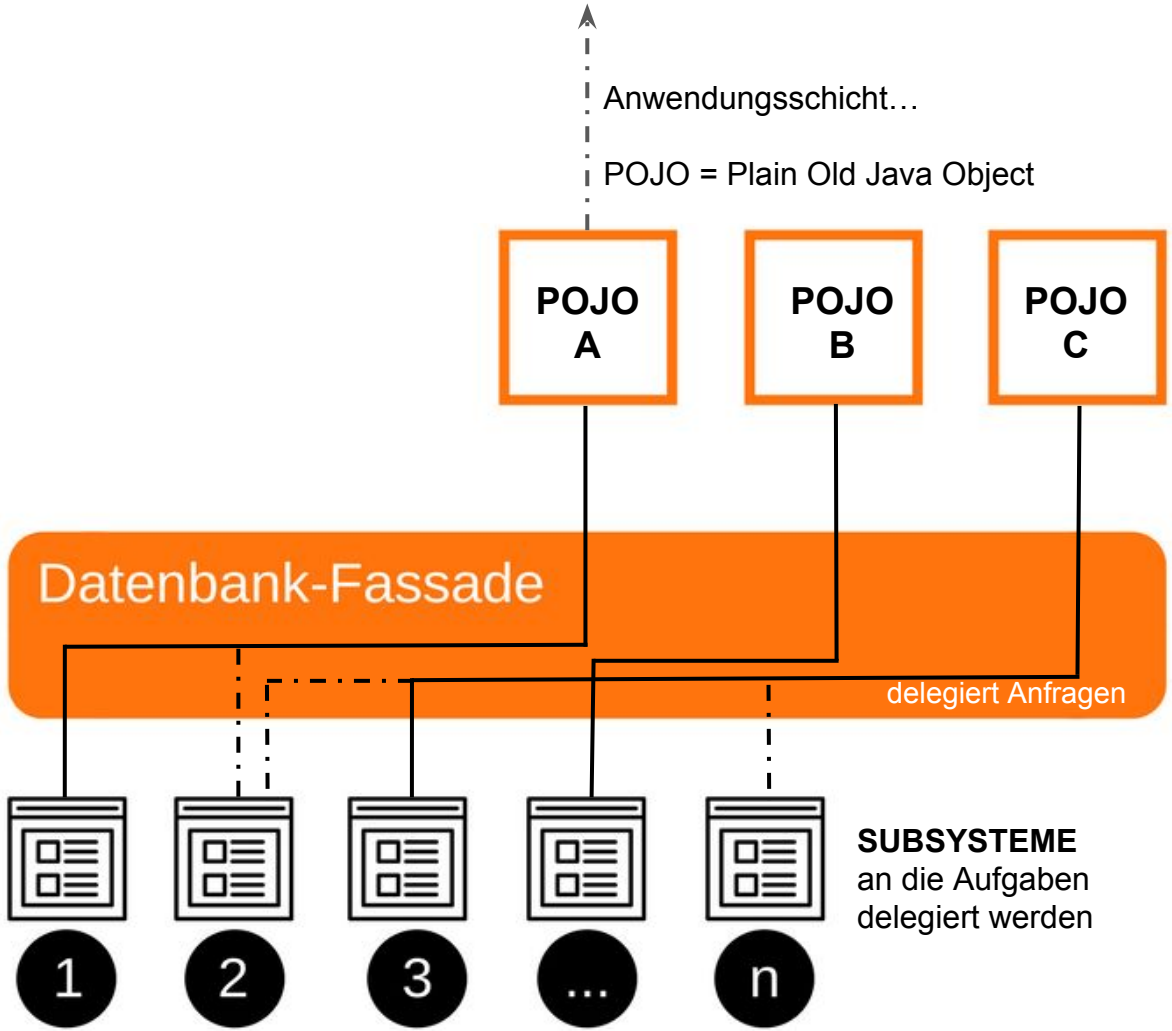
- Enthält Modell-Logik, wie Validierung, Verhalten, ...
- Wird niemals in der Persistenzschicht verwendet
- Wird ggf. durch POJO befüllt oder enthält POJO
- Kann optional sein

MOJOs sind die tatsächlichen Anwendungsobjekte, die das Verhalten und die Logik der Anwendung abbilden und die Daten aus der Persistenzschicht nutzen

MOJO
Model Java Object

Beispiel

```
public class Book {

    private String name;
    private String author;
    private String isbn;

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getAuthor() { return author; }
    public void setAuthor(String author) { this.author = author; }

    // ISBN Prüfziffernberechnung
    public int calculateISBNCheckDigit() { ... }

}
```


Anwendungsschicht…
POJO = Plain Old Java Object
POJO A
POJO B
POJO C
Datenbank-Fassade
delegiert Anfragen
1
2
3
...
n
SUBSYSTEME
an die Aufgaben
delegiert werden

Datenbankanwendung im Überblick