

SE Besprechung

Übung 4 – Architektur, Modulentwurf

Architekturstile & Entwurfsmuster

Architekturstil

definiert die strukturelle Organisation eines Systems
(globaler als Entwurfsmuster)

Framework / Bibliothek

übernimmt bestimmte Aufgaben (domänenspezifisch, z.B. ein Mathematik-Framework in C++, kann runtergeladen und verwendet werden)

Entwurfsmuster (Design Pattern)

adressiert ein bestimmtes Problem (domänenunabhängig), welches immer wieder auftauchen kann (muss man selber implementieren)

Aufgabe 2.1 – Architekturstile

Was sind die strukturellen Teile des Systems?

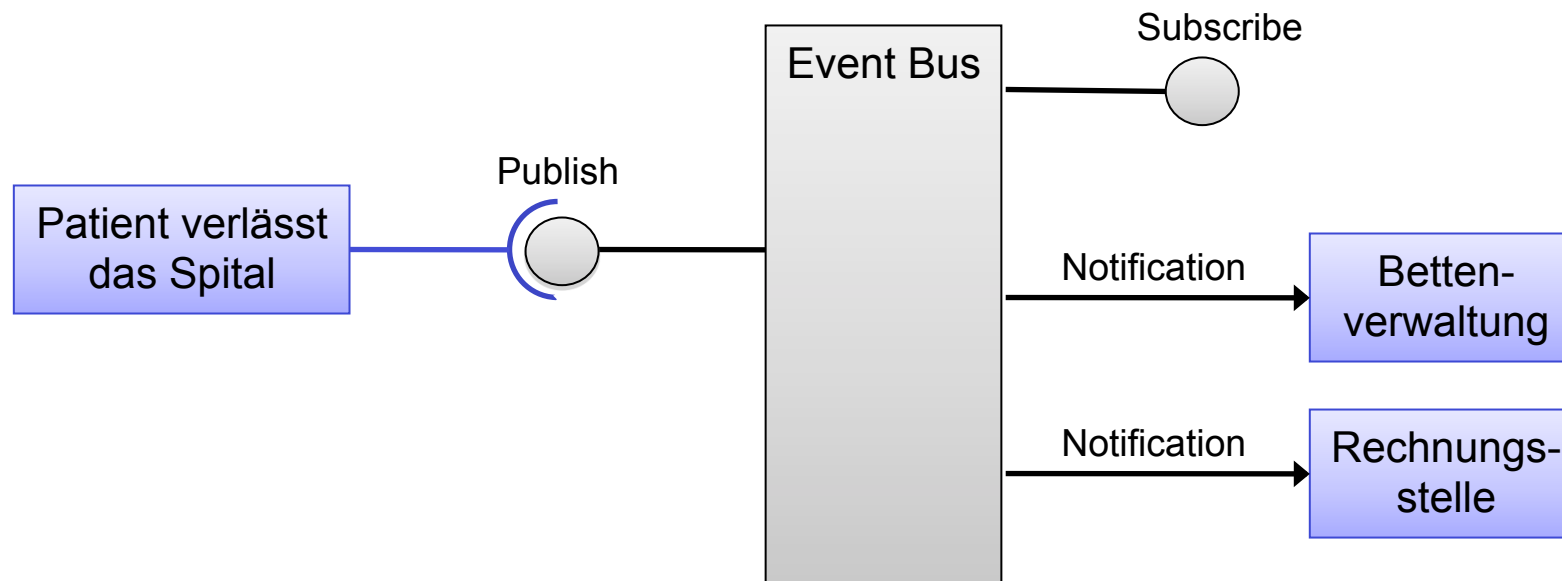
Wie kommunizieren diese Teile miteinander?

- Begründen der Auswahl → Brücke zwischen Text in der Aufgabe und dem gewählten Stil
- Komponenten, Konnektoren, **Einschränkungen**

Komponenten des Architekturstils $\neq$ System-Komponenten

Aufgabe 2.1.A – Verlassen des Spitals

Event-based System / Publish-Subscribe



Aufgabe 2.1.A – Verlassen des Spitals

Event-based System / Publish-Subscribe

Komponenten: Publishers / Subscribers

Konnektoren: Infrastruktur zur Übertragung von Events und zur Registrierung

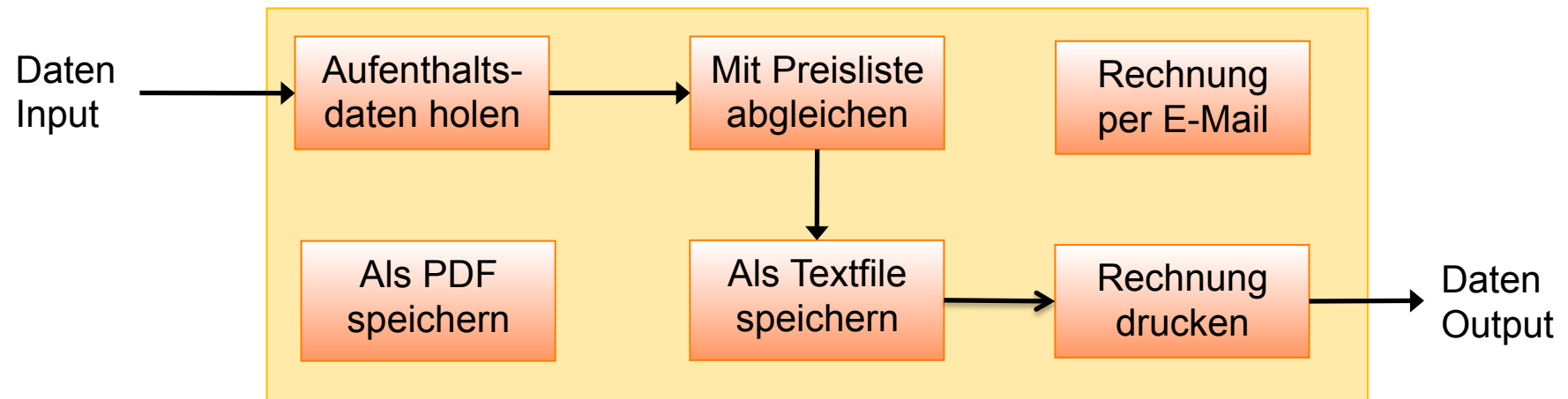
Einschränkungen: Kommunikation nur über den Event Bus

- + Komponenten sind lose gekoppelt
- Keine Garantie, dass Publisher eine Antwort bekommt
- Beliebige Reihenfolge der Antworten

...

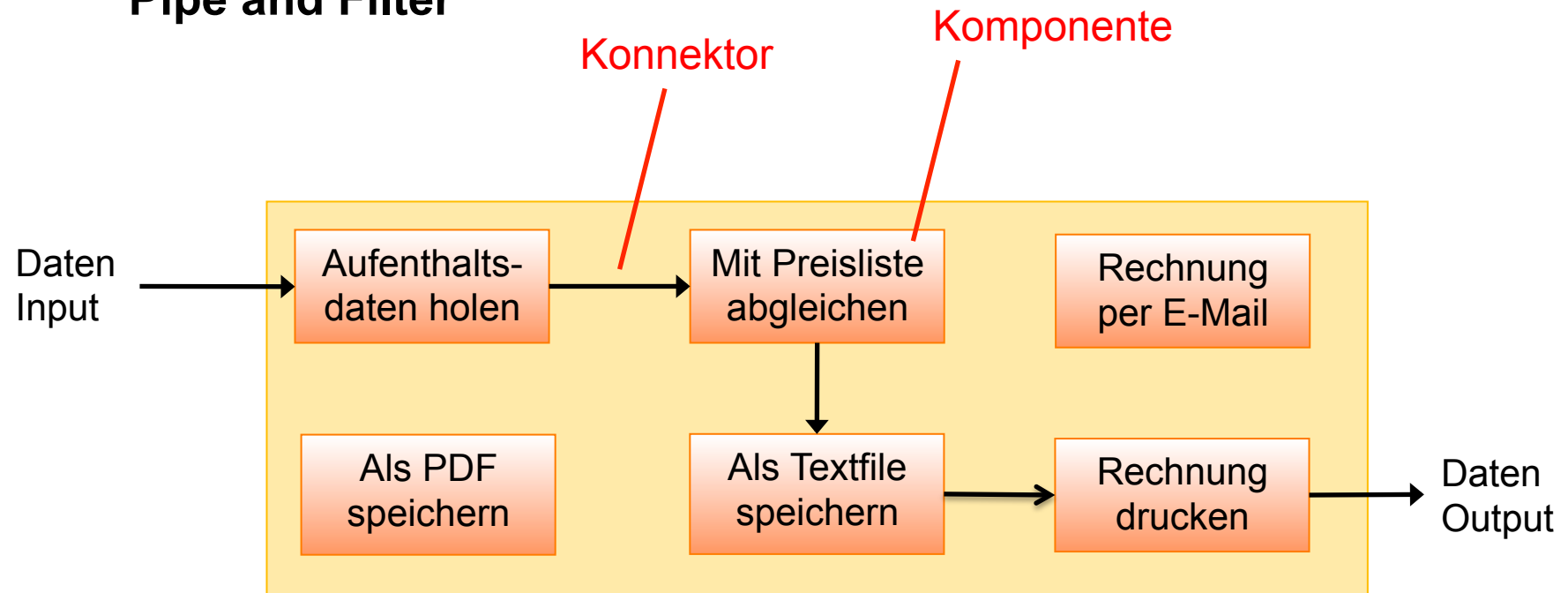
Aufgabe 2.1.B – Abrechnung

Pipe and Filter



Aufgabe 2.1.B – Abrechnung

Pipe and Filter



Aufgabe 2.1.B – Vorstellungsinfos

Pipe and Filter

Komponenten: Filter (Filter, Sortierer, Konverter)

Konnektoren: Pipes

Einschränkungen: Filter teilen keine Zustände, Filter kennen ihre Nachbarn nicht

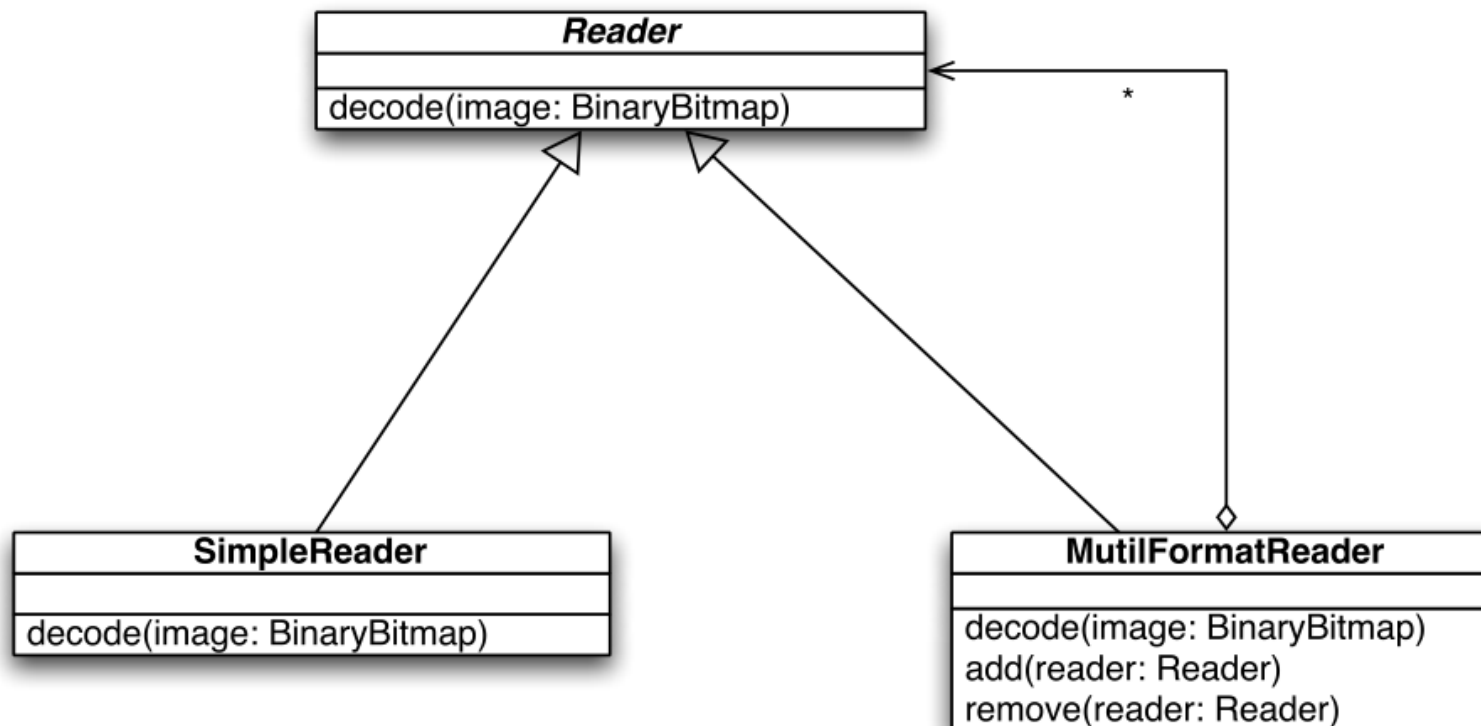
- + Wiederverwendbarkeit (hinzufügen, umsortieren, auswechseln von Filtern)
- Stapelverarbeitung (Batch Processing)
- Rechnungsaufwand (Overhead, z.B. muss jeder Filter die Datei erneut parsen)

Aufgabe 2.2 – Entwurfsmuster

- **Erzeugungsmuster (Creational Patterns)**
Verlagern der Objekt-Erzeugung
(z.B. *Singleton*, *Abstract Factory*)
- **Strukturmuster (Structural Patterns)**
Komposition von Klassen / Objekten
(z.B. *Adapter*, *Decorator*)
- **Verhaltensmuster (Behavioral Patterns)**
Interaktion von Klassen / Objekten
(z.B. *Strategy*, *Observer*)

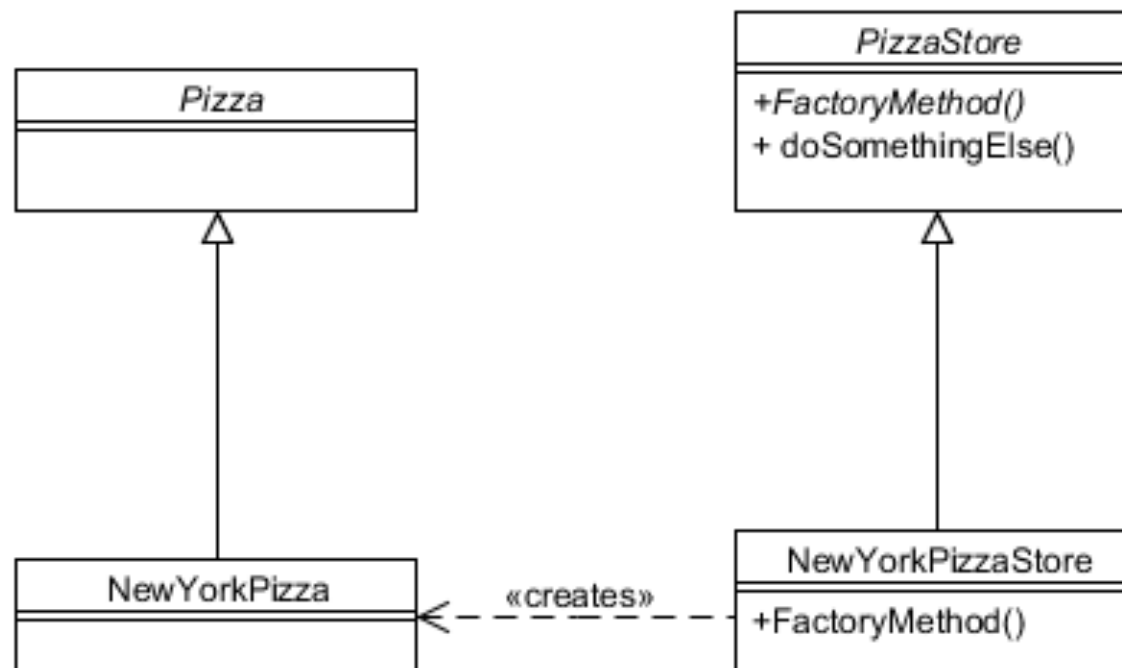
Aufgabe 2.2.A – Strukturelle Muster

- Composite Pattern



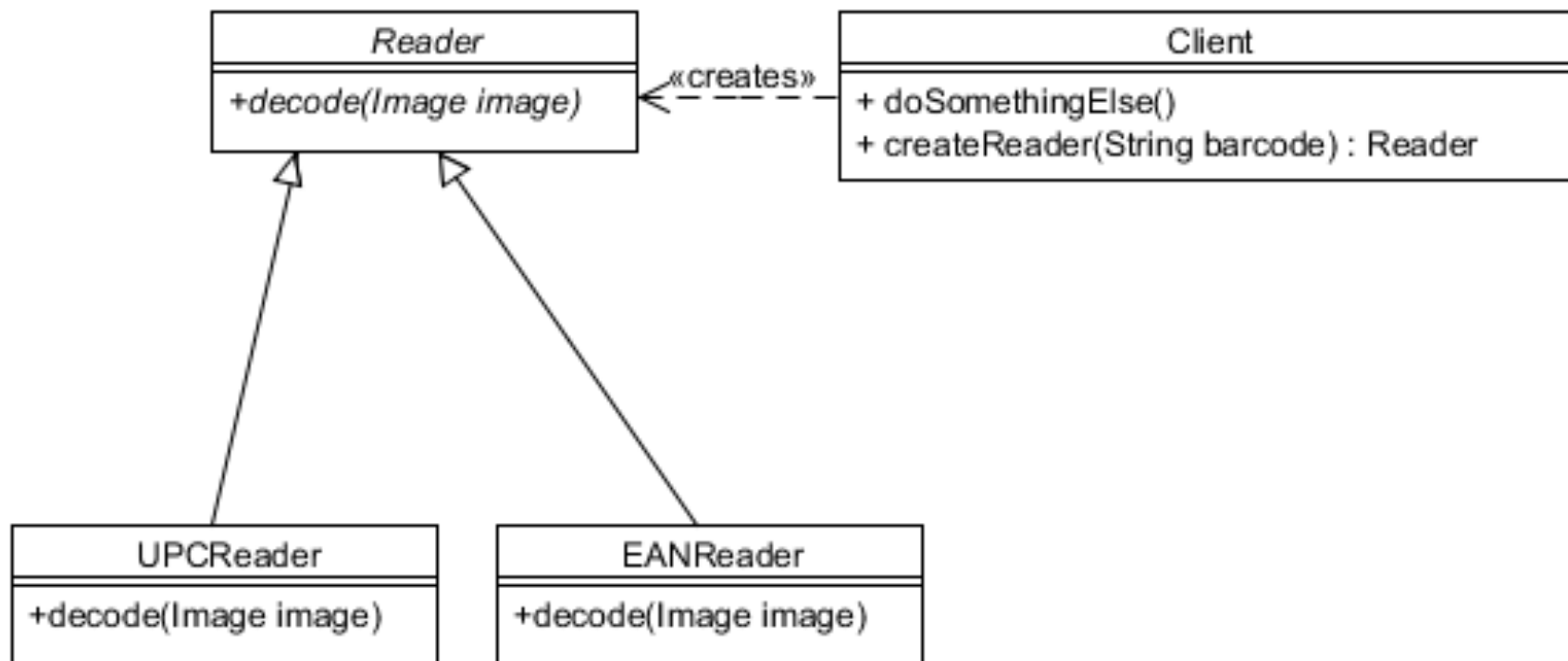
Aufgabe 2.2.B – Erzeugungsmuster

- Factory Method



Aufgabe 2.2.B – Erzeugungsmuster

- Parameterized Factory Method



Aufgabe 2.2.B – Erzeugungsmuster

- Parameterized Factory Method

```
class ReaderFactory {
    public static enum BarcodeType {
        QR_CODE, DATAMATRIX, UPC_E
        // many more
    };

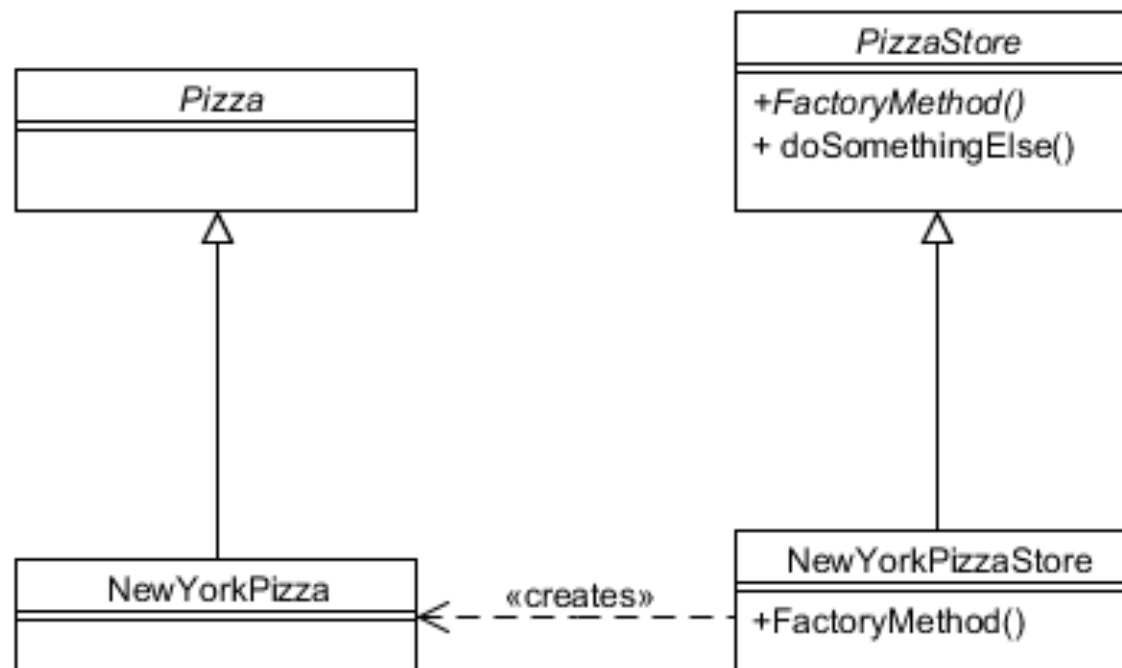
    public static Reader createReader(BarcodeType type) throws Exception {
        switch(type) {
            case QR_CODE:
                return new QRCodeReader();
            case DATAMATRIX:
                return new DataMatrixReader();
            case UPC_E:
                return new UPCEReader();
        }
        throw new Exception("Barcode_type_not_supported.");
    }
}
```

Aufgabe 2.2.B – Erzeugungsmuster

- **Zum Vergleich: Abstract Factory**
 - **-> deutlich mehr Klassen, aufwändiger**

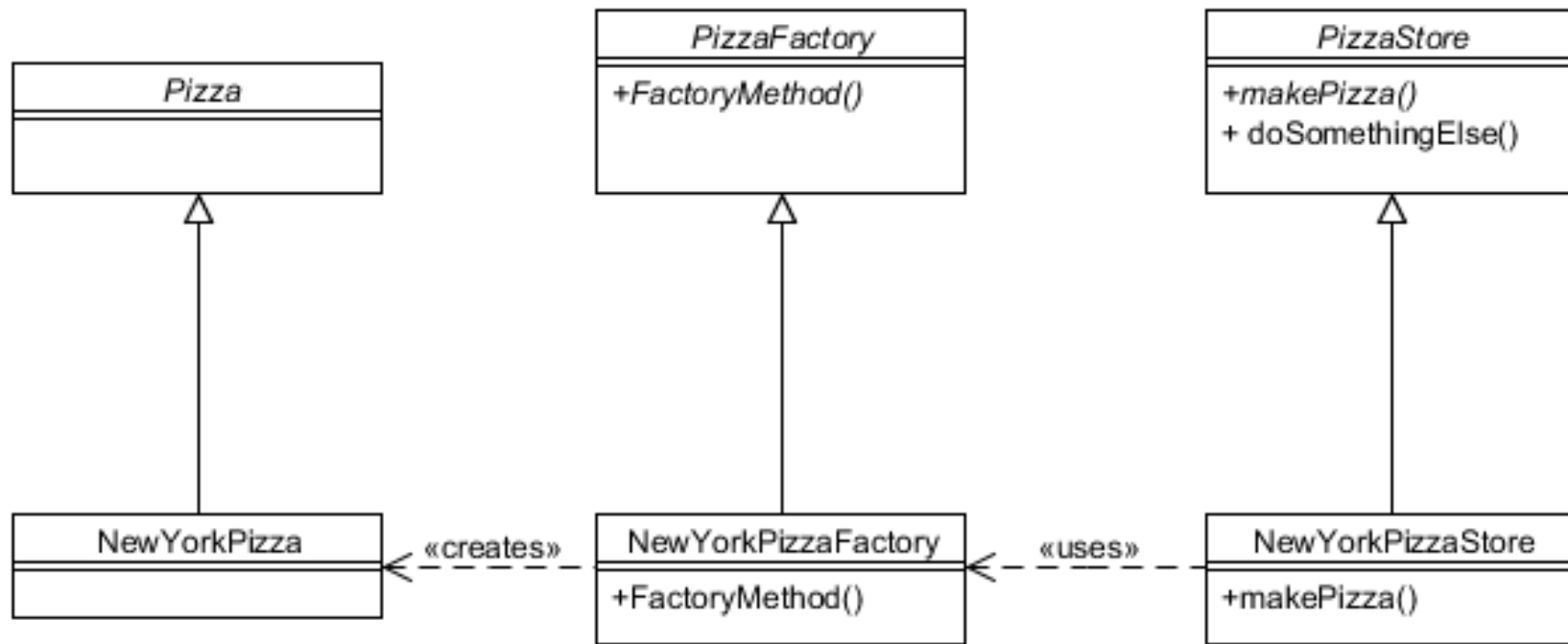
Aufgabe 2.2.B – Erzeugungsmuster

- Factory Method



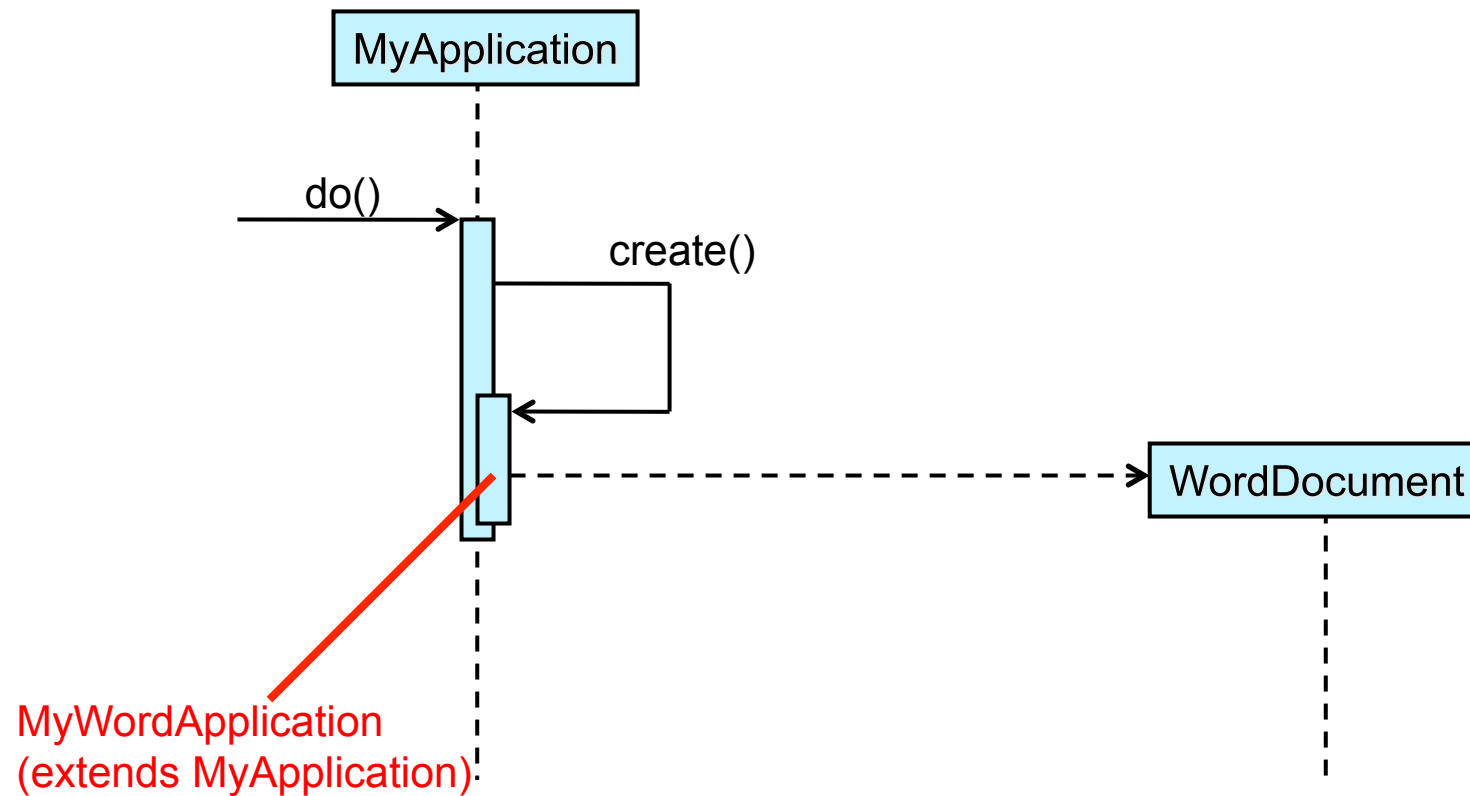
Aufgabe 2.2.B – Erzeugungsmuster

- Abstract Factory



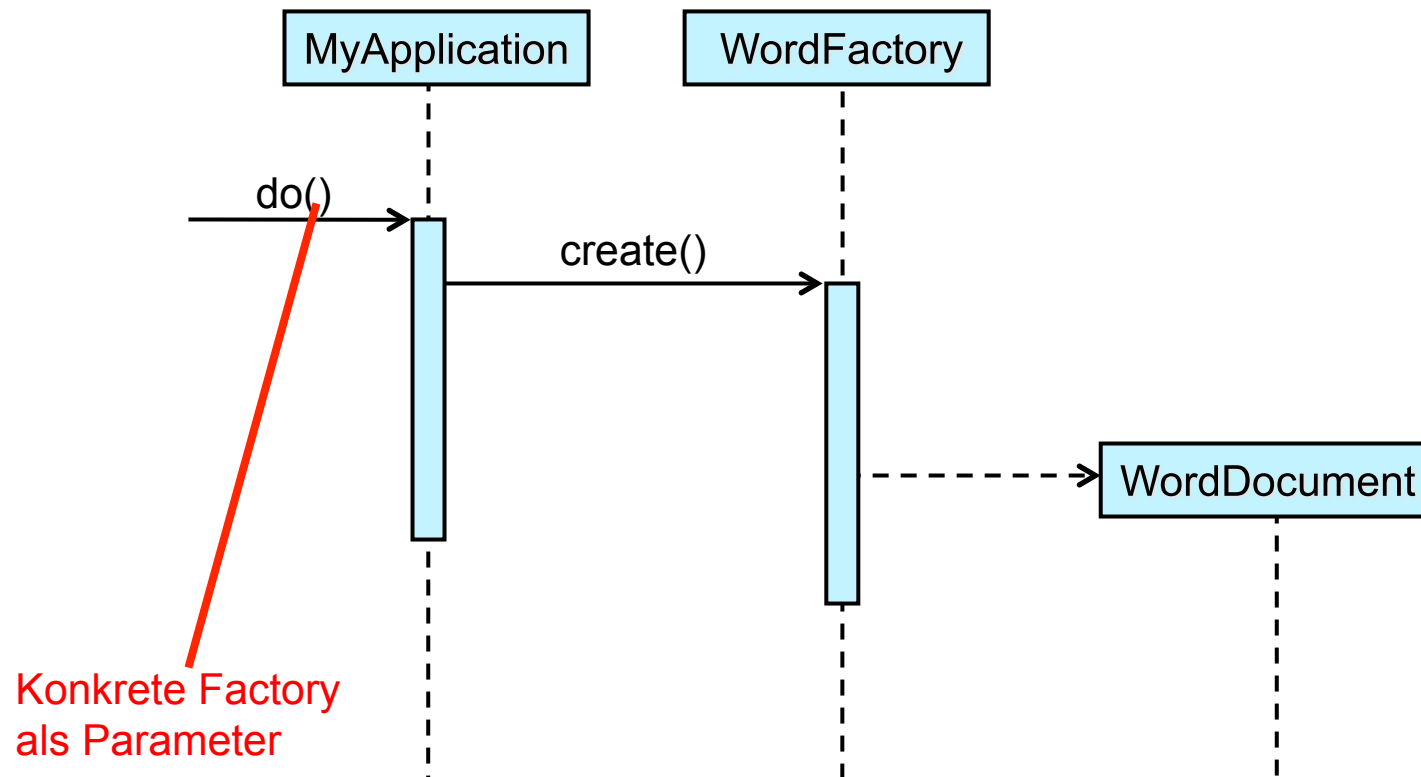
Abstract Factory vs. Factory Method

Factory Method



Abstract Factory vs. Factory Method

Abstract Factory



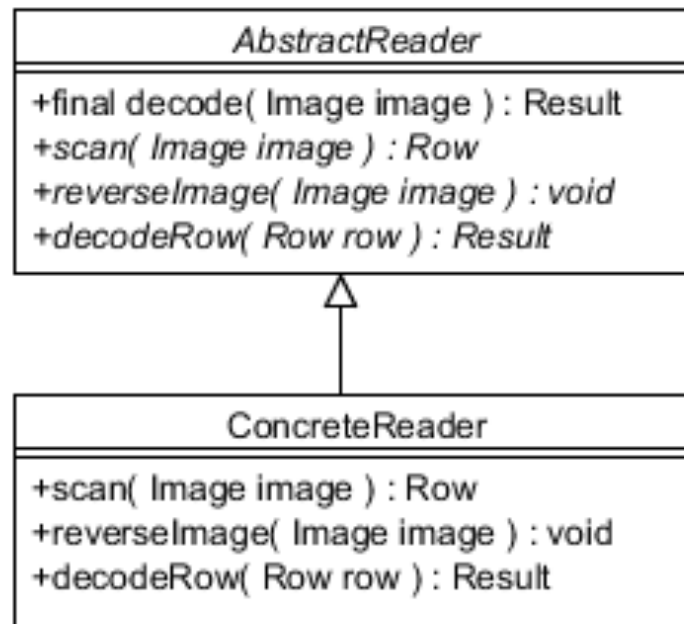
Aufgabe 2.2.C – Verhaltensmuster

Template Method

- Abstrakte Klasse legt Ablauf des Algorithmus vor („Template Methode“), eine Serie von Aufrufen abstrakter Methoden
 - Subklassen müssen diese abstrakten Methoden implementieren, je nach konkretem Reader
 - Template Methode muss final sein, damit sie nicht überschrieben werden kann
-
- Damit ist der Ablauf des Algorithmus bei allen gleich
 - Aber: Verantwortung liegt bei Subklassen

Aufgabe 2.2.C – Verhaltensmuster

Template Method



```
public abstract class AbstractReader {

    public final Result decode( Image image ){
        Row row = scan( image );
        if( !row.isSuccess() ){
            reverseImage( image );
            row = scan( image );
        }

        Result result = decodeRow( row );
        return result;
    }

    public abstract Row scan( Image image );
    public abstract void reverseImage( Image image );
    public abstract Result decodeRow( Row row );
}
```

Übung 4 Resultate

