# *SE Besprechung*

## Übung 4 – Architektur, Modulentwurf



Mengia Zollinger

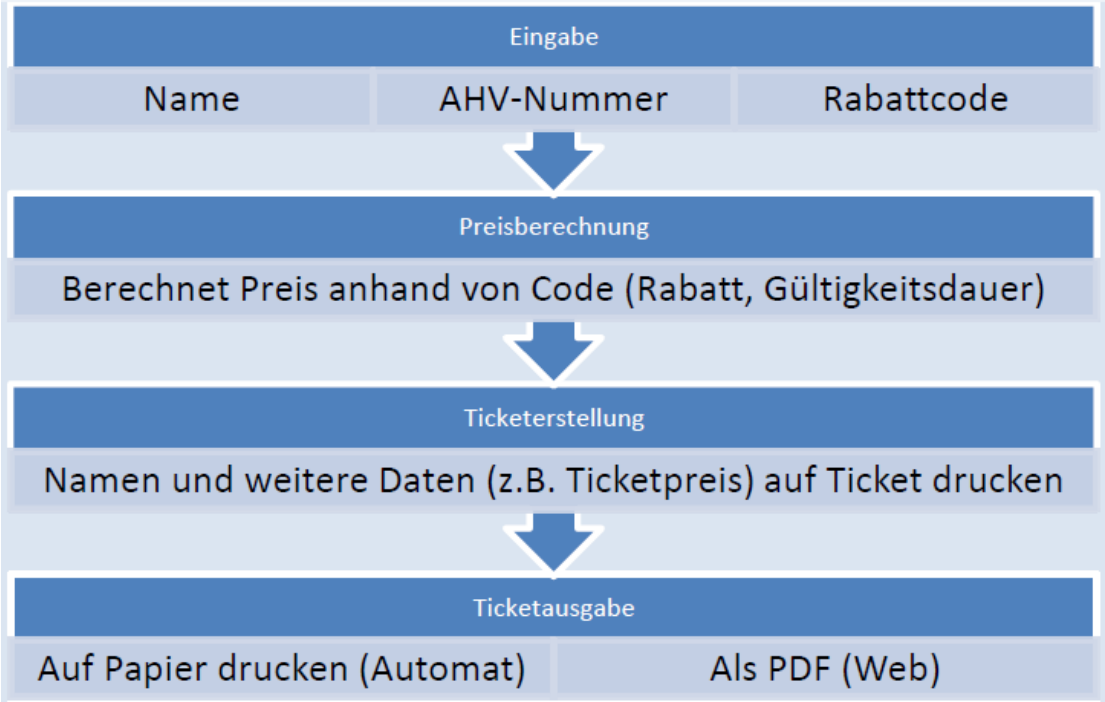
# 2.1 Architekturstile *(6 Punkte)*

## 2.1.A Ausgabe eines Monatsabos an Angestellte

### Lösung: Pipe-and-Filter

**Beispiel Lösung**

*[Benz, Zihler, Aggeler]*




**Wertung:**

- Richtige Architekturwahl: 2 Punkte
- Erklärung für die Wahl: 1 Punkt

# *2.1 Architekturstile (6 Punkte)*

## 2.1.A Ausgabe eines Monatsabos an Angestellte

**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)

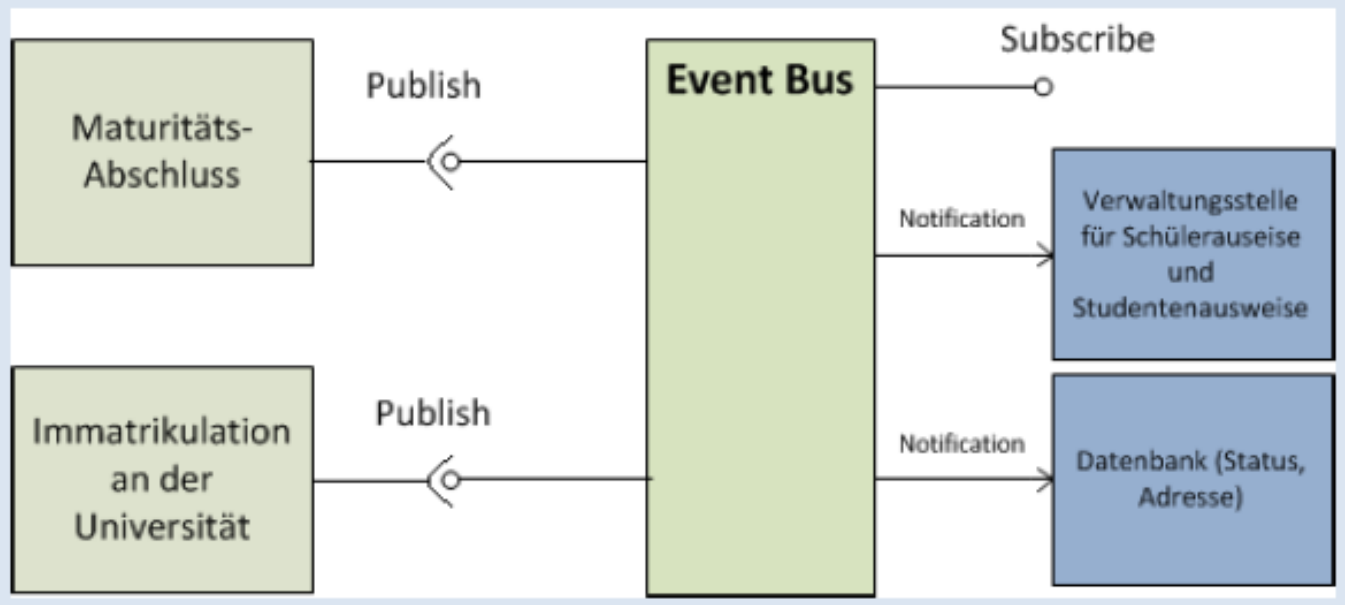
# 2.1 Architekturstile *(6 Punkte)*

## 2.1.B Abschliessen der Matura und Immatrikulation an einer Hochschule

### Lösung: Event-Based System (Publish-Subscribe)

**Beispiel Lösung**

*[Benz, Zihler, Aggeler]*




**Wertung:**

- Richtige Architekturwahl: 2 Punkte
- Erklärung für die Wahl: 1 Punkt

# *2.1 Architekturstile (6 Punkte)*

## 2.1.B Abschliessen der Matura und Immatrikulation an einer Hochschule

**Event-based System / Publish-Subscribe**

•Komponenten: Publishers / Subscribers

•Konnektoren: Infrastruktur zur Übertragung von Events und zur Registrierung

•Einschränkungen: Kommunikation nur über den Event Bus

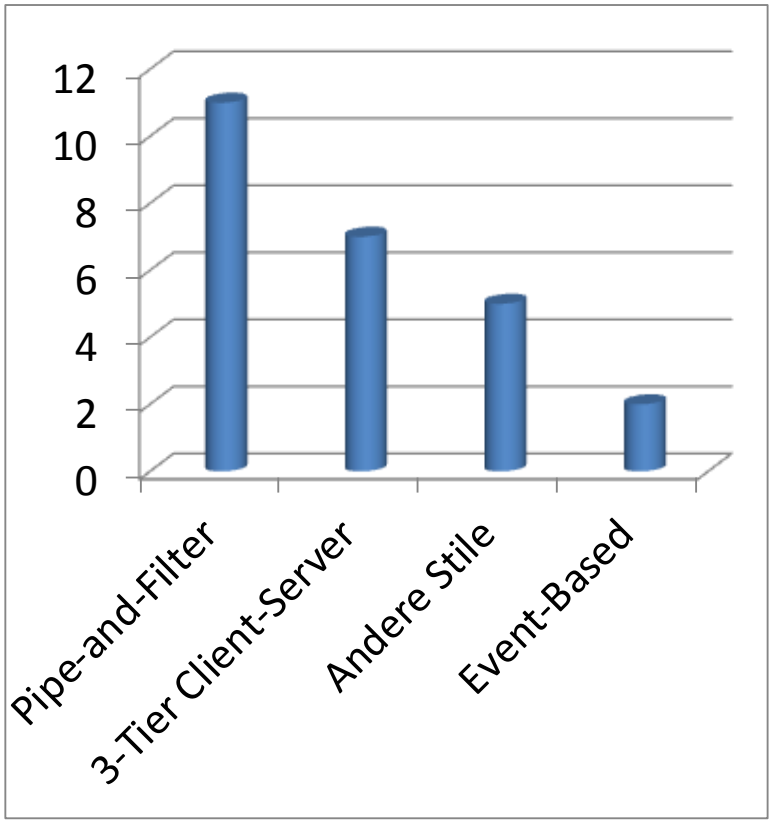
**(+)** Komponenten sind lose gekoppelt

**(−)** Keine Garantie, dass Publisher eine Antwort bekommt

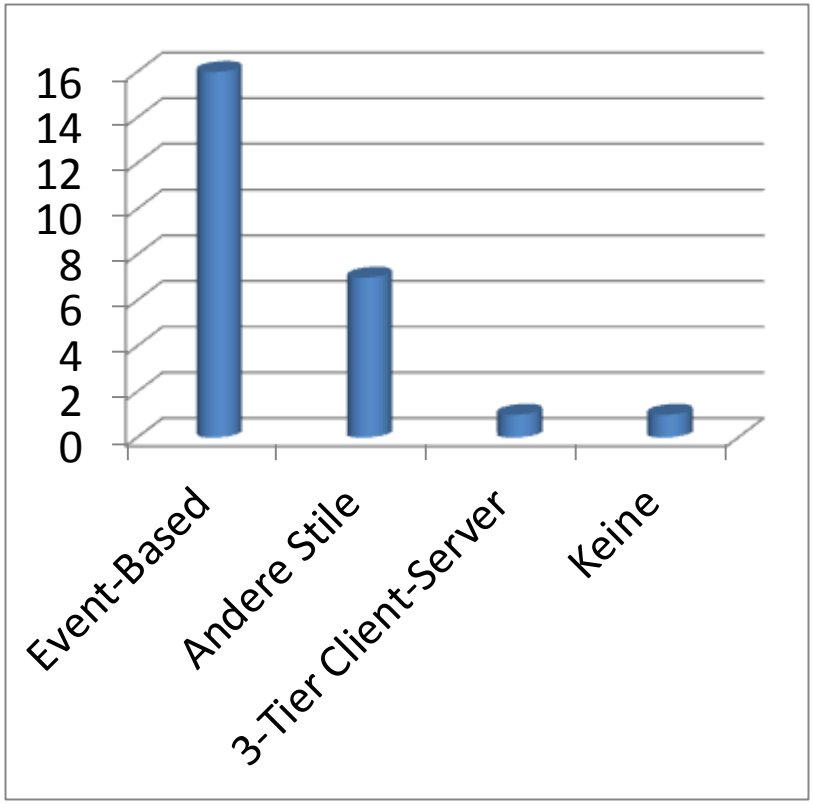
**(−)** Beliebige Reihenfolge der Antworten

# 2.1 Architekturstile *(6 Punkte)*

## 2.1.A Monatsabo




## 2.1.B Matura



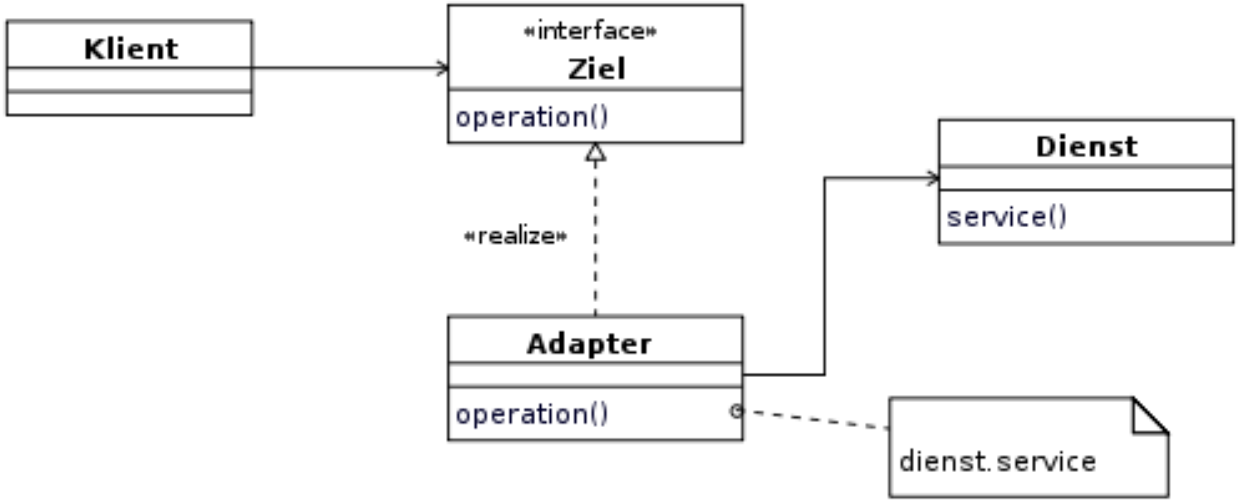
# 2.2.A Strukturelle Muster *(2 Punkte)*

System soll mit vielen Uni-DBs kommunizieren können

➢Welches Strukturpattern ist geeignet?

Strukturpattern: **Adapter**

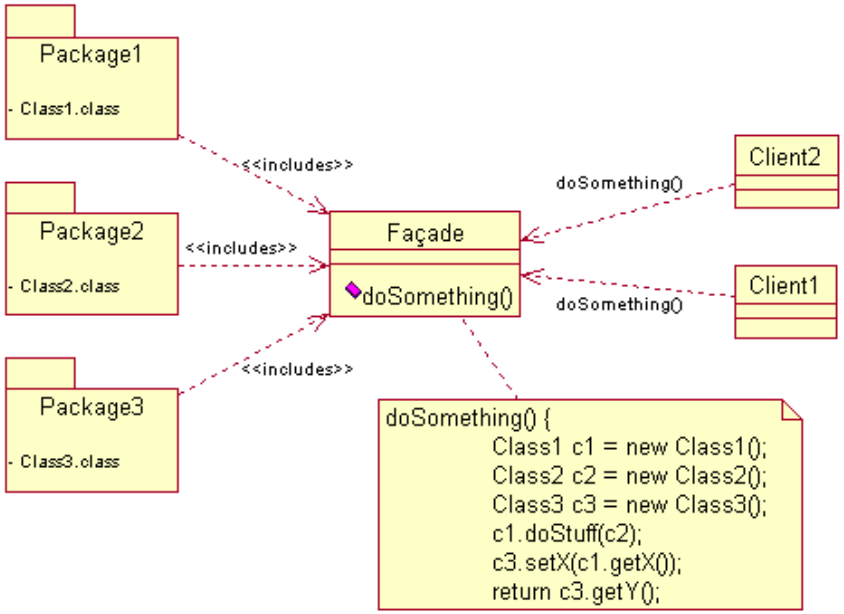
Beschreibung: Interface zwischen System und DBs als einheitliche Schnittstelle, Adapterklassen implem



# 2.2.A Strukturelle Muster *(2 Punkte)*

Strukturpattern: **Fassade**

Beschreibung: Fassade bietet einheitliche & vereinfachte Schnittstelle zu verschiedenen Schnittstellen eines Subsystems



# 2.2.B Erzeugungsmuster *(4 Punkte)*

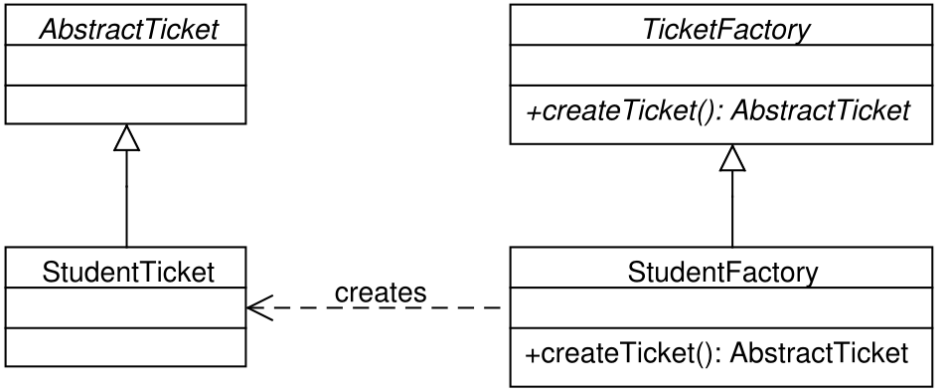
System: Unterscheidung verschiedener Abotypen

Strukturpattern: **Factory Method**

Beschreibung: Erstellung des Objekts wird delegiert. Anstelle von **`new`** wird eine Fabrikmethode aufgerufen.

Vorteil: Entkoppelt die Erzeugung & Verwendung

Verbirgt die Details der Erzeugung



# 2.2.B Erzeugungsmuster *(4 Punkte)*

Strukturpattern: **Factory Method**

Variante: Parametrisierte Fabrikmethode

Je nach Typ wird eine andere Fabrik erzeugt. Für den Benutzer geschieht dies völlig transparent!

```
public class TicketFactoryCreator {

    public static TicketFactory getTicketFactory(TicketType type){
        switch(type){
        case STUDENT:
            return new StudentFactory();
        case EMPLOYEE:
            return new EmployeeFactory();
        default:
            return new NormalFactory();
        }
    }
}
```

# *2.2.C Verhaltensmuster (4 Punkte)*

Erstellung eines Tickets hat fixen Ablauf:

- Ticketart abfragen
- Daten validieren
- Ticket drucken

- Welches Design Pattern ist geeignet?
- Begründung?
- Klassen (Prototyp) erstellen

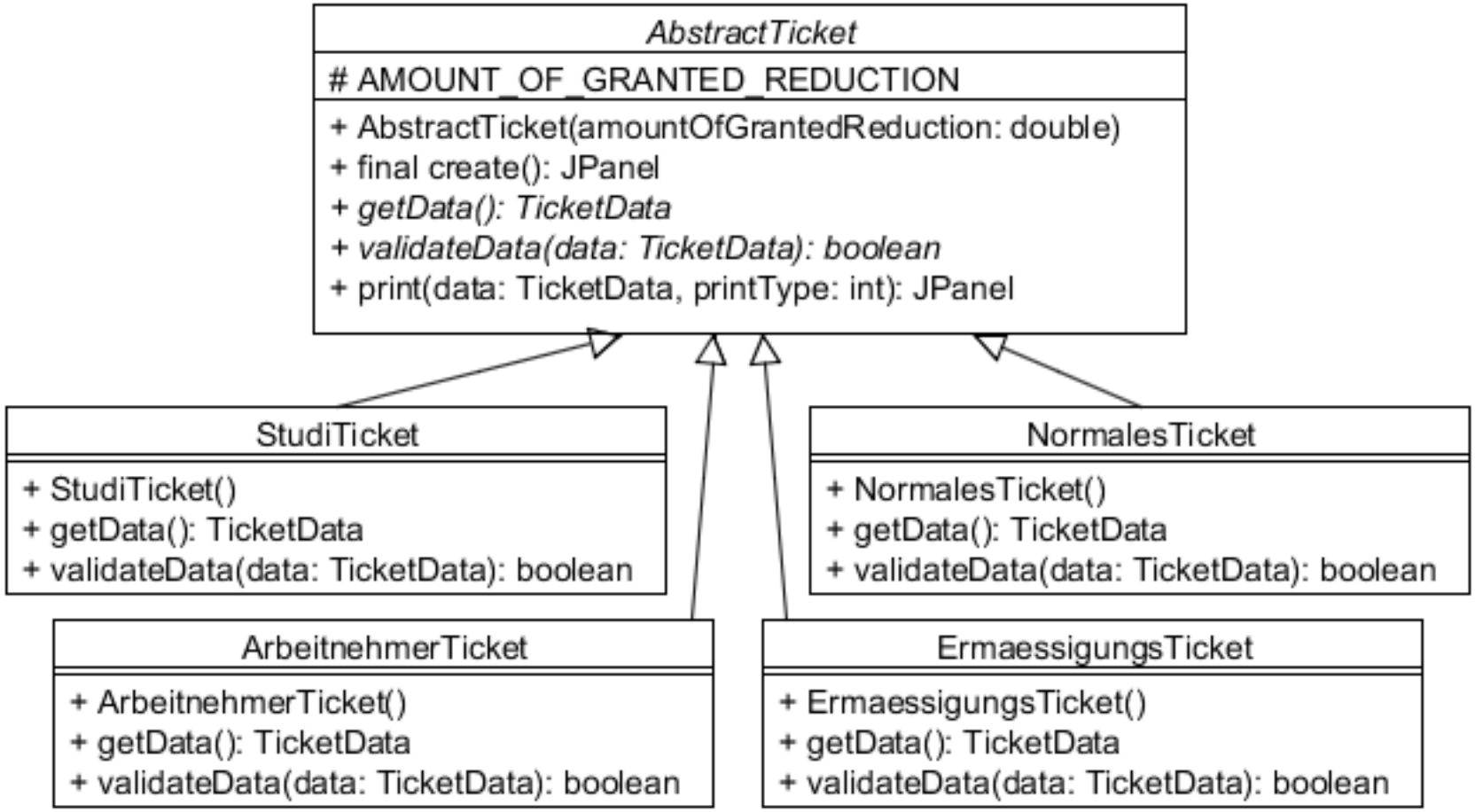
# *2.2.C Verhaltensmuster (4 Punkte)*

Design Pattern: **Template Method**

Begründung:

- final createTicket() stellt sicher, dass Ablauf eingehalten wird (nicht überschreibbar).
- Teilschritte des Algorithmus kann man in Subklassen auslagern

# 2.2.C Verhaltensmuster *(4 Punkte)*



# 2.2.C Verhaltensmuster *(4 Punkte)*

```java
public abstract class Ticket {

    public abstract String getData();
    public abstract boolean validateData();

    public final void createTicket() {
        getData();
        validateData();
        printData();
    }

    private void printData() {
        if (isPDF()) {
            // print PDF
        } else {
            // print paper
        }
    }

    private boolean isPDF() {
        return true;
    }

   public abstract static class StudentTicket extends Ticket {

        @Override
       public String getData() {
           return null;
       }

        @Override
       public boolean validateData() {
           return true;
       }
    }

}
```

# 2.3 Entwurfsmuster in Java API *(5 Punkte)*

**Kommentare:**

•Anforderungen: Pattern Name, Kategorie, Beteiligte, Java Elemente – z.B. in Tabellenform:

| Entwurfsmuster Name | Kategorie | Java Beispiele | Beteiligte | Java Elemente |
|---|---|---|---|---|
| | | | | |

•Häufige Fehler:

- Pattern Name vergessen
- Nicht klar, welches Java Beispiel gewählt wurde
- java.awt.Composite kein Beispiel für Composite Pattern
- Java.awt.event.MouseAdapter kein Beispiel für Adapter Pattern

# *2.3 Entwurfsmuster in Java API (5 Punkte)*

**Behavioral Patterns in Java – Beispiele**

- Iterator: java.util.Iterator (iteriert über Collection)
- Observer: java.util.Observer (beobachtet ein Observable)
- Strategy: java.awt.LayoutManager (z.B.: GridBagLayout, FlowLayout)
- Template Method: java.io.InputStream (read(byte[] b) braucht abstrakte Methode read())
- Visitor: javax.lang.model.element.ElementVisitor (visits a Java 6 AST, made of Element)
- Command: javax.activation.CommandObject
- Mediator: java.awt.KeyboardFocusManager (manages the “focus” among javax.swing.JComponent)

# *2.3 Entwurfsmuster in Java API (5 Punkte)*

**Structural Patterns in Java – Beispiele**

- Adapter: java.io.InputStreamReader (adapts an InputStream to a Reader)
- Decorator: java.io.FilteredInputStream (decorates an InputStream)
- Composite: java.awt.Component (Container, like Panel, contains Label, Button)
- Bridge: java.net.Socket (implemented by a SocketImpl)
- Facade: java.awt.Font
- Proxy: java.lang.reflect.Proxy

# *2.3 Entwurfsmuster in Java API (5 Punkte)*

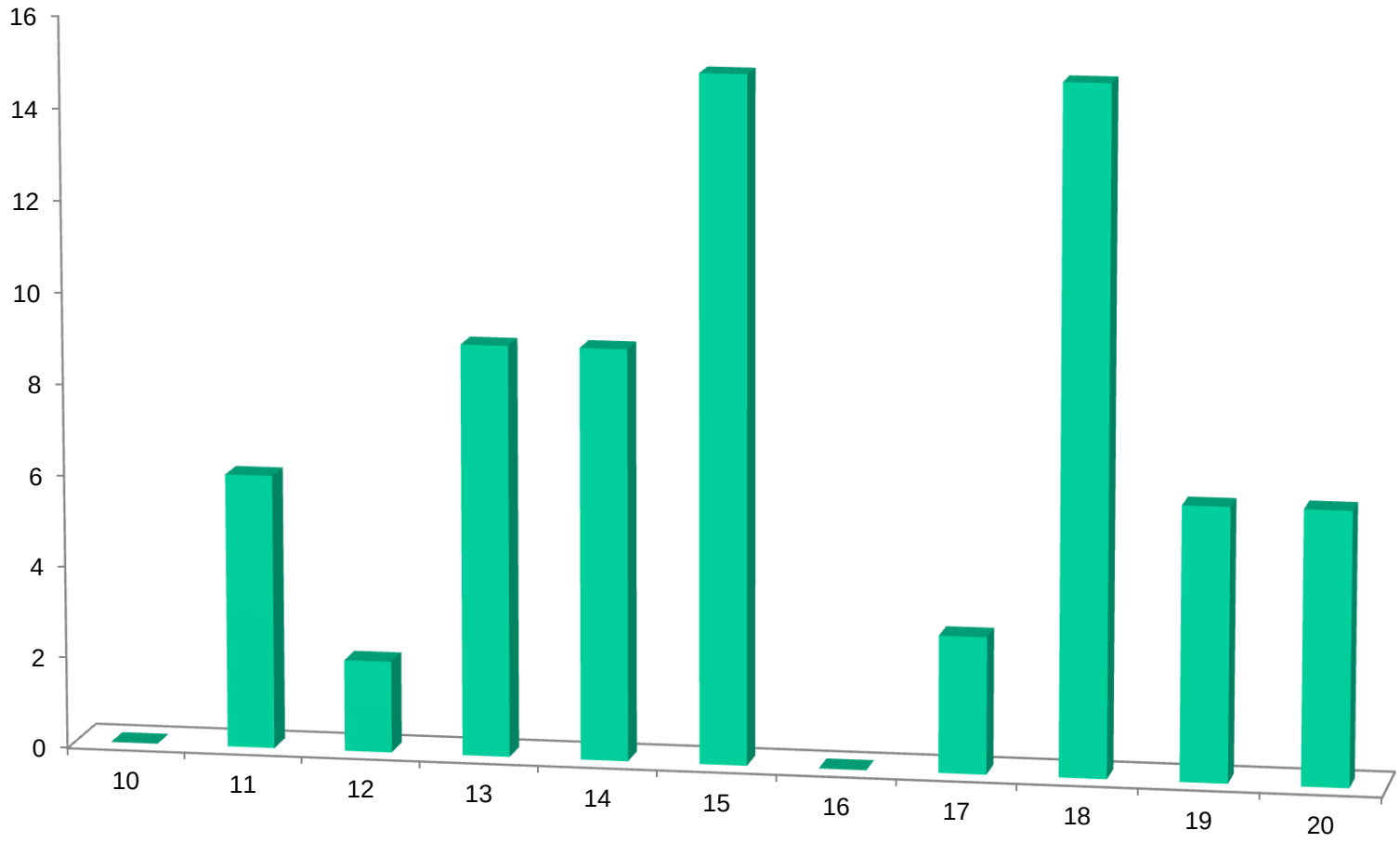
## Creational Patterns in Java – Beispiele

- Singleton: java.lang.Runtime
- Abstract Factory: javax.xml.validation.SchemaFactory (creates a Schema)
- Factory Method: java.net.SocketImpl (creates a java.io.InputStream)
- Prototype: java.lang.Cloneable
- Builder: java.lang.StringBuilder (builds String)

## Wertung:

- Beispiel für die relevanten Kategorien: 0.5 Punkte * 3
- Liste der Beteiligten: 0.5 Punkte * 3
- Java Elemente: 0.5 Punkte * 3
- Erklärung: 0.5 Punkte

# Übung 4 Resultate



# *Allgemeines*

- Fragen/Feedback zu Übung 4?