



# POSE

## Design-Patterns: Mediator

Simon Fischer

5AHIF



# POSE

## Mediator

Simon Fischer  
25.2.2023

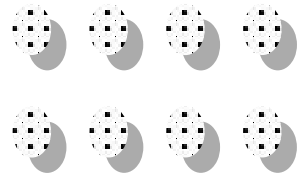
5AHIF

# Inhalt

- Motivation
- Einleitung
  - Zusätzliche Themen
  - Design Pattern
  - Praxisbeispiele: 1 konkret
- Vor / Nachteile
- Verwandte Patterns
- Beispiel



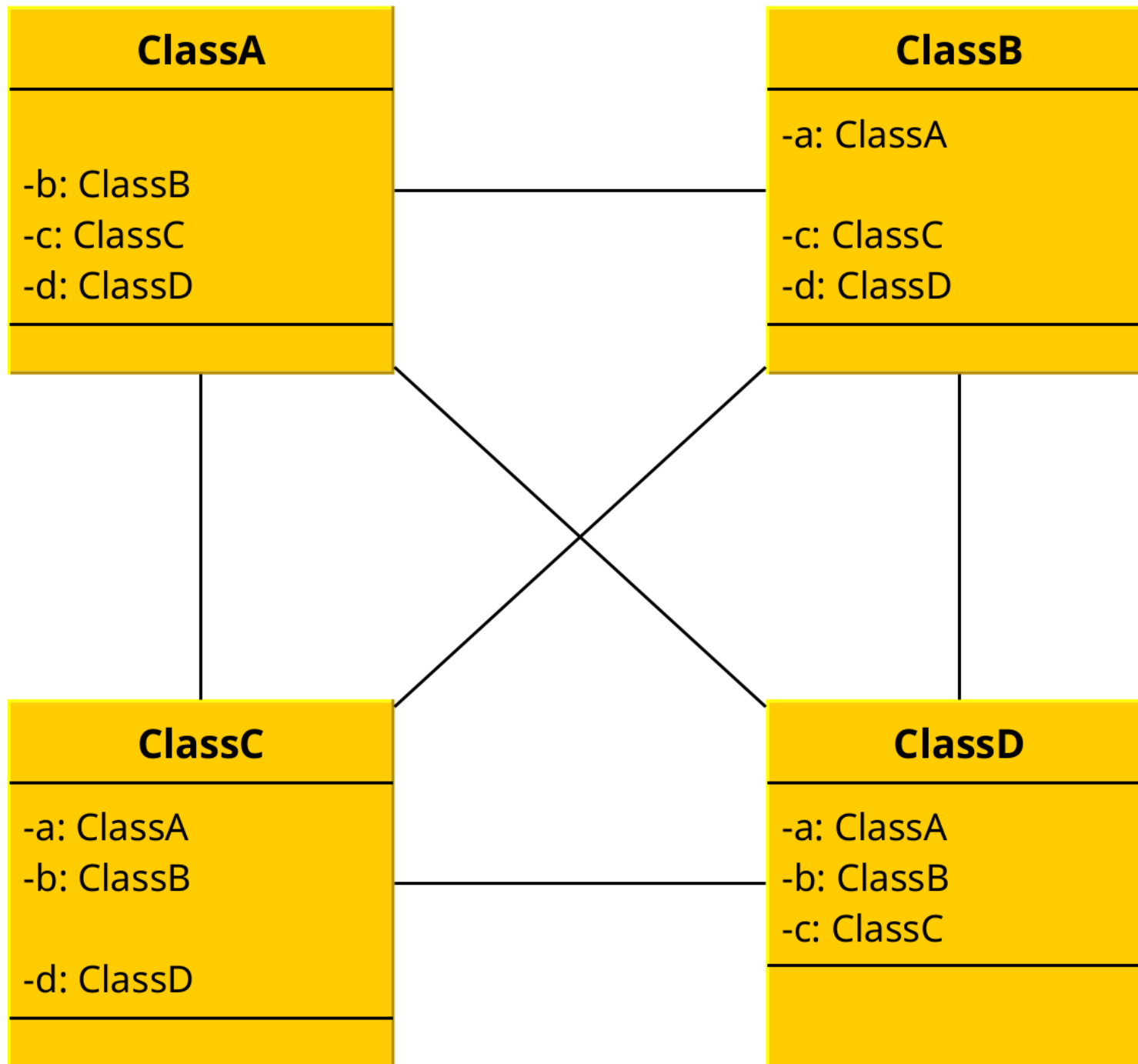
Illustrations by Pixeltrue on  
[icons8](#)





# Motivation

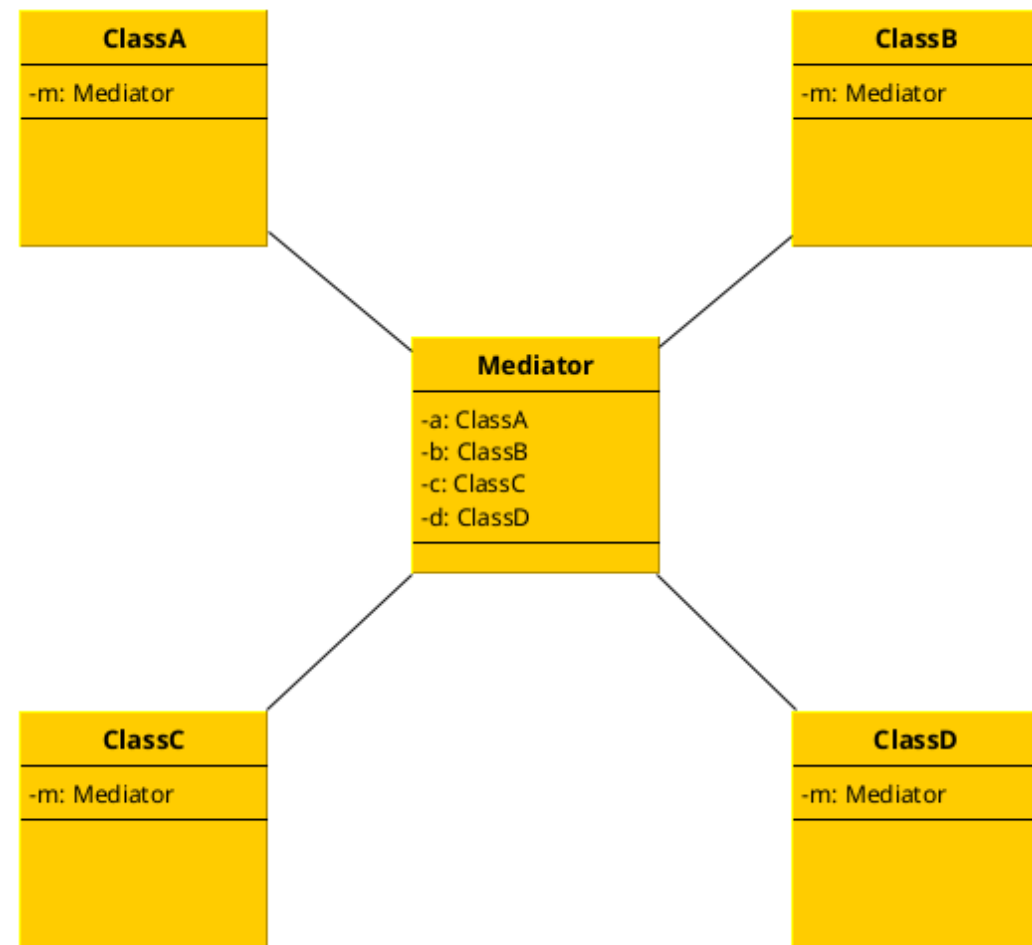
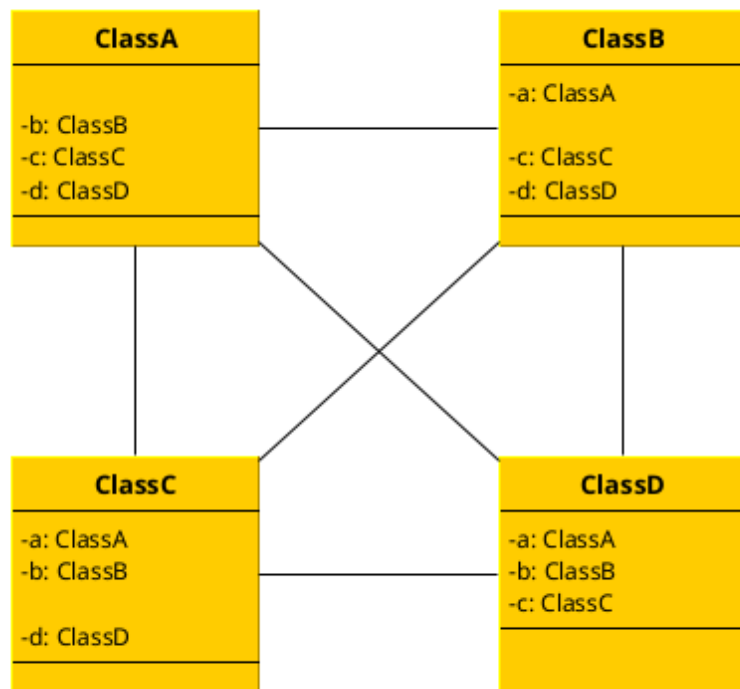
# Motivation





# Einleitung

# Besser





# Grundlagen

## Lose Kopplung



Was ist Lose Kopplung?





# (Lose) Kopplung

- Unabhängigkeitsgrad
- Lose / starke Kopplung
- Änderung → keine Änderung  
woanders

# Lose Kopplung - Vorteile



**Wartbarkeit**



**Wiederverwendbarkeit**



**Testbarkeit**

...

# BSP - starke Kopplung

```
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int multiply(int a, int b) {
        return a * b;
    }
}

@RequiredArgsConstructor
public class CalculartorApp {
    private final Calculator calculator;

    public int add(int a, int b) {
        return calculator.add(a, b);
    }

    public int multiply(int a, int b) {
        return calculator.multiply(a, b);
    }
}

public class Main {
    public static void main(String[] args) {
        var calcApp = new CalculartorApp();
        System.out.println(calcApp.add(3, 14));

        System.out.println(calcApp.multiply(4, 2));
    }
}

~
"tight.java" 32L, 647B
```

# BSP - lose Kopplung

```
public interface ICalculator {
    public int calculate(int a, int b);
}

public class Adder implements ICalculator {
    public int calculate(int a, int b) {
        return a + b;
    }
}

public class Multiplier implements ICalculator {
    public int calculate(int a, int b) {
        return a * b;
    }
}

@RequiredArgsConstructor
public class CalculatorApp {
    private final ICalculator calculator;

    public int calculate(int a, int b) {
        return calculator.calculate(a, b);
    }
}

public class Main {
    public static void main(String[] args) {
        var adder = new CalculatorClient(new Adder());
        System.out.println(adder.calculate(3, 14));


        var multiplier = new CalculatorClient(new Multiplier());
        System.out.println(multiplier.calculate(4, 2));
    }
}

~
"loose.java" 35L, 806B
```



# Design Pattern

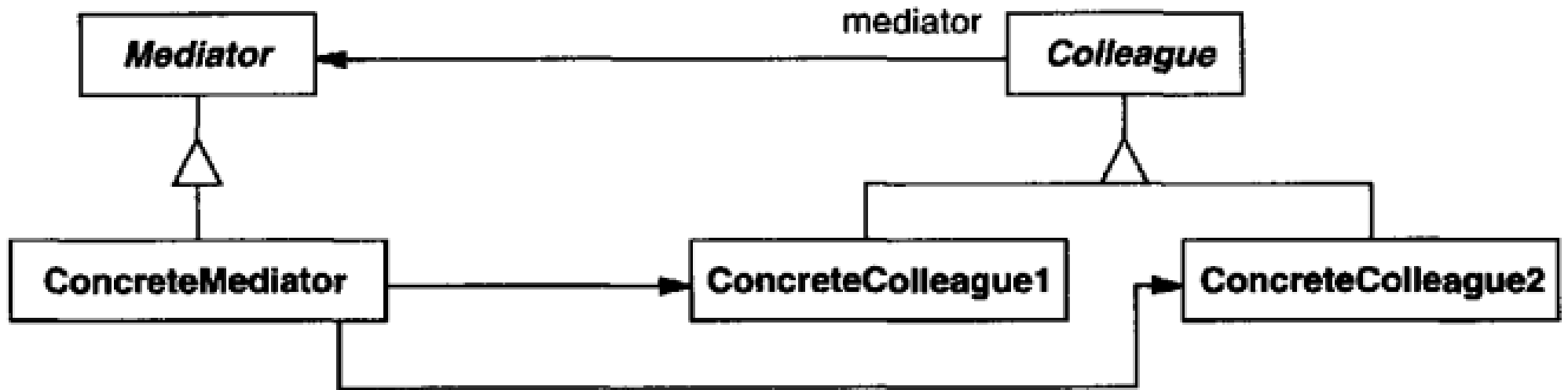
## Mediator



Wie funktioniert Mediator konkret? -  
Design Pattern des Mediators



# Mediator





# Praxisbeispiele

## Mediator



Verwendungsbeispiele des Mediators in der  
Praxis



## **MVC**

Controller

## **Swing**

Event Dispatch Thread

## **AWT**

ActionListener

## **Java.util.concurrent**

Koordination der Thread-Kommunikation

## **JavaScript**

Event-Loop

## **D-Bus**

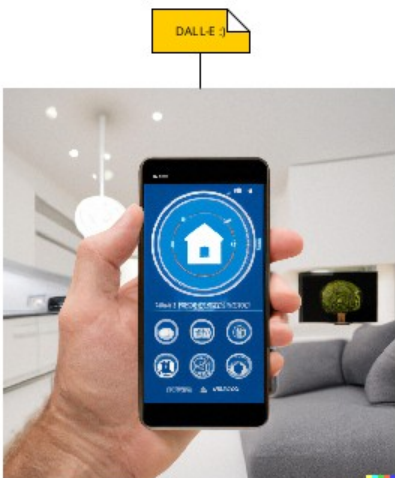
Linux: Inter-Process-Communication



# Konkrete Anwendung - grafisch



Smart-Home-Hub (Mediator)  
z.B. Home-Assistant, IoBroker, ...





# Vor- Nachteile

# Vorteile

- Lose Kopplung
  - Wartbarkeit
  - Wiederverwendbarkeit
  - Testbarkeit
- Änderbarkeit / Erweiterbarkeit
- Zentrale Interaktionslogik
  - Übersichtlichkeit
  - Wartbarkeit
  - Many → Many => One → Many
- ...

# Nachteile

- Verschiebung der Komplexität
- “god object”
- geringere Flexibilität
- Single-responsibility?
- Skalierbarkeit
- Overhead
- ...



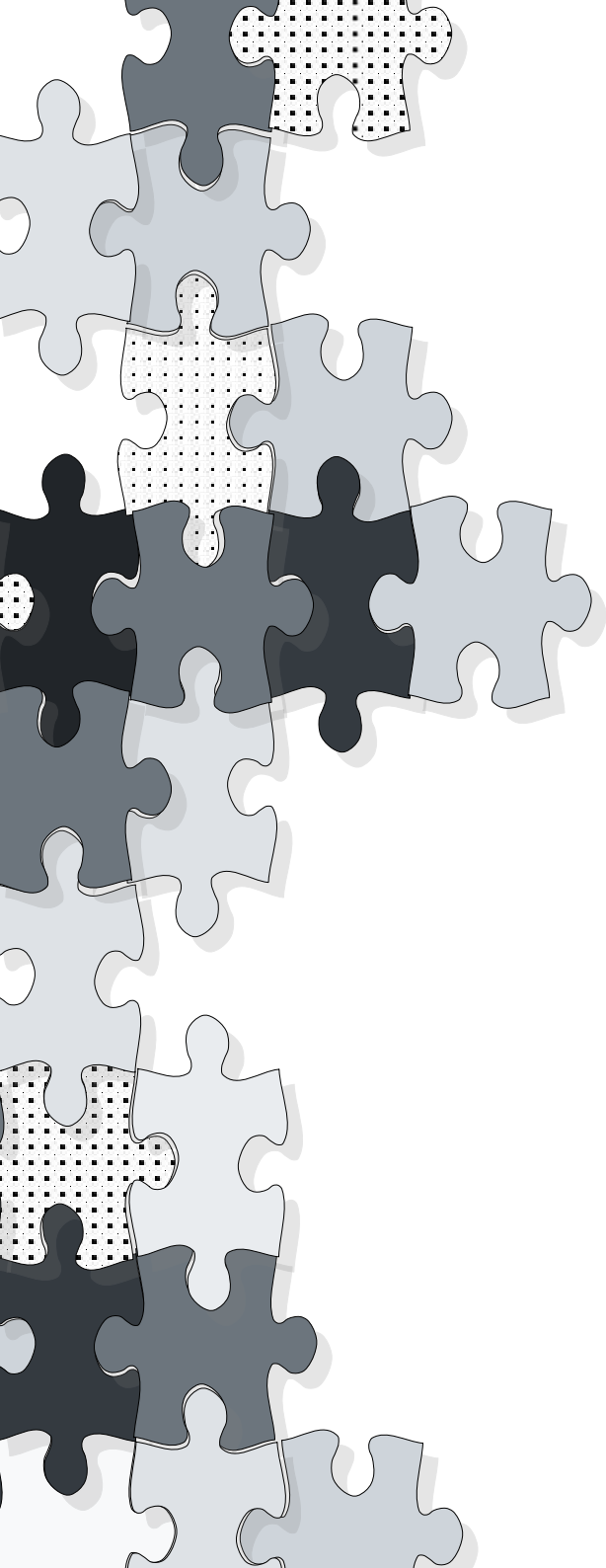
# Verwandte Patterns

# Mediator vs Facade

- „two way Facade“
- + extra Funktionalität



Beispiel



“

END

”

- This Presentation

