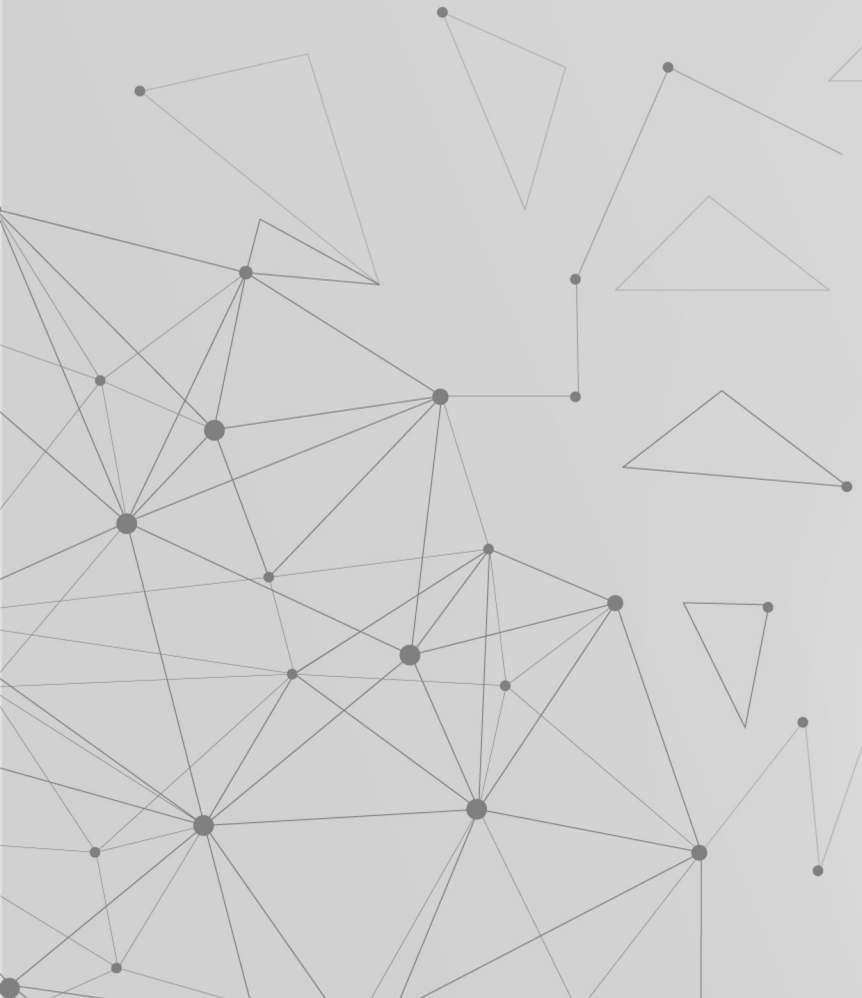


Pattern of the Week

Week 2: Interfaces, Composite Pattern, ZIIP,

TABLE OF CONTENTS

01

Interfaces

Er ist ein Mensch, er ist ein Tier, er ist …

Interfaces

- Klasse ohne Implementierung (“abstract class/method”)
- Definition der Funktionen → Signaturen
- Vertrag über die implementierte Funktionalität
- Unabhängig von der konkreten Implementierung
- Polymorphie (Vielgestaltigkeit) ohne “Diamond of Death”
    a. Eine Klasse, mehrere Interfaces
    b. Explizite Implementierung von Methoden
    c. Ein Interface, mehrere Implementierungen

Interface Segregation Principle (ISP)

- SOLID Prinzip
- Große Schnittstellen in mehrere kleine Schnittstellen aufteilen
- Trennung der Zuständigkeiten
- Zusammenhalten der Abhängigkeiten

Aufgabe

Person "interfacen"

Projekt: "Person"

Aufgabe: Sinnvolle *Interfaces* extrahieren

David Wheeler

- “All problems in computer science can be solved by another level of indirection.”
- Definieren von Abstraktionen für die Kommunikation

- Schnittstellen als gemeinsamer Vertrag
  - Interface
  - Typen
  - Methodensignaturen

➔ Alles was “public” ist, ist der Vertrag mit meiner API

Liskov Substitution Principle (LSP)

- Das LSP besagt, dass Subtypen sich so verhalten müssen wie ihr Basistyp

- Löst der Basistyp keine Exception aus, dürfen Subtypen auch keine Exception werfen
- Der Subtyp darf die Funktionalität eines Basistyps lediglich erweitern (z.B. Wertebereich)

→ Besser “Favour Composition over Inheritance”

02

Composite Pattern

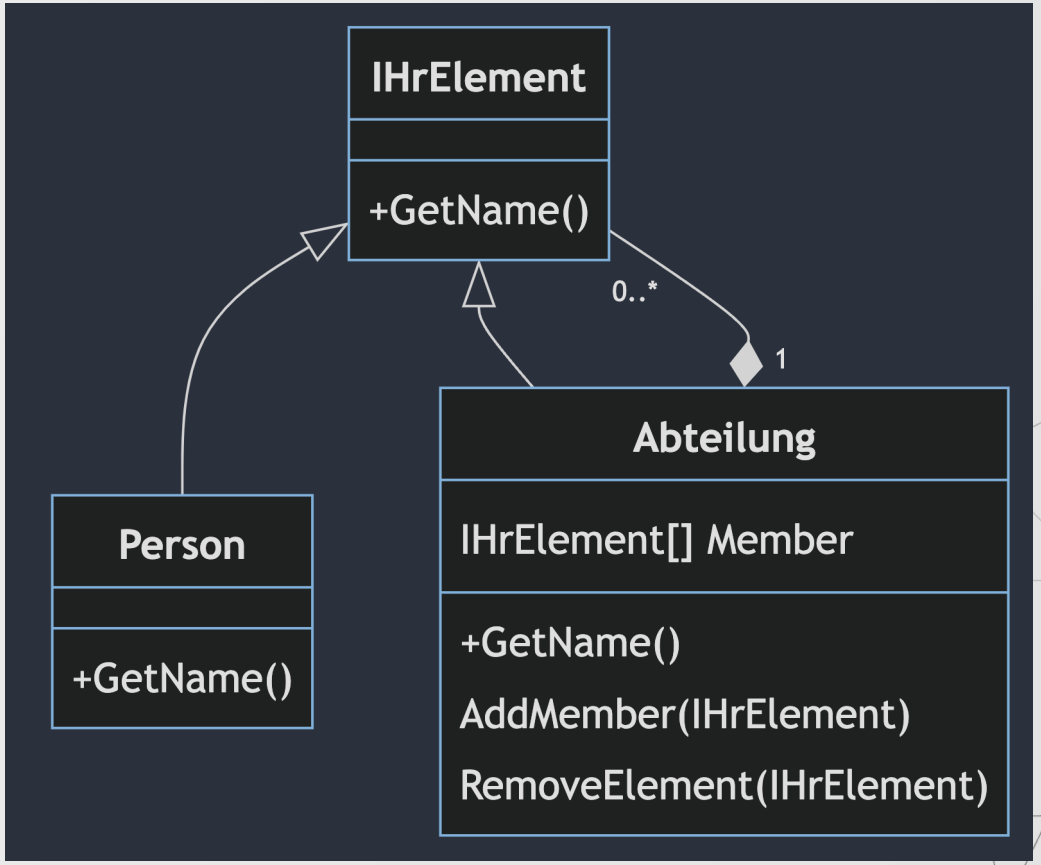
You can enter here the subtitle if you need it

Composite Pattern

- Iteration/Implementierung einer Baumstruktur
- Ein Knoten implementiert das gleiche Interface wie ein Blatt
- Der Klient arbeitet auf dem Interface

- *Idee: Eine Liste von Elementen ist auch ein Element, gekapselt in einer Klasse*
- *Falle: Das Interface beschreibt eine übergeordnete Funktionalität*

UML Composite Pattern



Aufgabe

Implementiere das Composite Pattern

Projekt “Person”

Interface “IHrElement”
Methode “GetName()”

Person implementiert das Interface
Abteilung = Member[]
Abteilung implementiert das Interface

03

Refactoring

Refactoring

- “Refactoring […] bezeichnet […] die manuelle oder automatisierte **Strukturverbesserung** […] unter **Beibehaltung** des *beobachtbaren* **Programmverhaltens**” –Wikipedia

- Keine Veränderung des äußeren Verhaltens
- Veränderung des inneren Verhaltens
  - Performance
  - Erweiterbarkeit, Austauschbarkeit
  - Testbarkeit

Complex Refactorings

- “Es ist nicht möglich, Code direkt in der ultimativen Form zu schreiben.”
- Irgendwann sind größere Eingriffe erforderlich
- Komplexe Refactorisierung benötigt Unit Tests

Zero-Impact-Injection Pattern (ZIIP)

- Klassen-internes Refactoring
  - Extract Class
  - Dependency Injection
  - Default-Parameter (null)
  - null-coalescing operator (??)
- Ergebnis: Code außerhalb der Klasse bleibt bestehen

Demo

Code Generator als ZIP

Projekt “PdfTools”

Code Generator injizieren

Vorteile

- Ausgelagter Code ist Testbar
    a. Neue Klasse sollte höchste Qualitätsansprüche haben (Testing, Dokumentation, …)
    b. Alte Klasse mit schlechter Qualität wird kleiner
    c. Tipp: Bibliothek extrahieren, wenn der Code „reif“ (Mature) ist
- Methodensignaturen ändern sich nicht
- Ausgangsklasse ist testbarer (nicht direkt in allen Bereichen)