Entwurfsmuster: Einführung, Composite Pattern

- Einige Konzepte der OOP sollen wiederholt und diskutiert werden
- Begriffe White Box Reuse und Black Box Reuse sollen verstanden werden
- Anhand von Beispielen soll die Notwendigkeit der Definition von Design Patterns geweckt werden
- Erkennen wie Design Patterns beschrieben werden können
- Kategorien und Arten von Design Patterns unterscheiden können
- Einige Design Patterns aufzählen können

Literaturhinweis

Entwurfsmuster von Kopf bis Fuß, Elisabeth und Eric Freeman, O'Reilly Verlag, ISBN 978-389721421

Folgende objektorientierte Konzepte sind bekannt

- Klassen und Interfaces
- Vererbung
- Überladen (Methode kann unterschiedliche Parameter haben)
- Überschreiben (Methode kann in abgeleiteter Klasse redefiniert werden)
- Polymorphie
- Dynamisches Binden

Frage: Wie verwendet man diese Konzepte bei konkretem Problem?

- Welche Klassen werden benötigt?
- Wie stehen Klassen untereinander in Beziehung (Vererbung, Komposition)
- In welcher Klasse werden welche Methoden definiert?

Vererbung

- Subklasse erbt Implementierung der Superklasse
- Teile der Superklasse sind in Subklasse sichtbar (White Box Reuse)

Vorteile	Nachteile
Einfache Methode zur Wiederverwendung	Statischer Mechanismus: Implementierung der Superklasse kann nicht dynamisch geändert werden Veränderungen an Superklasse können Subklassen beeinflussen Wiederverwendung der Subklasse wird eingeschränkt da abhängig von Superklasse

White Box Reuse

```
public class MyArrayList<T> extends ArrayList<T> {
  public boolean add(T o) {
    return super.add(o);
  }
}
```

Problem verschärft sich, wenn Basisklasse von außerhalb vorgegeben ist

Komposition

- Objekte benutzen andere Objekte
- Die Interna sind dem aufrufenden Objekt nicht bekannt
- Nur Schnittstelle ist bekannt (Black Box Reuse)

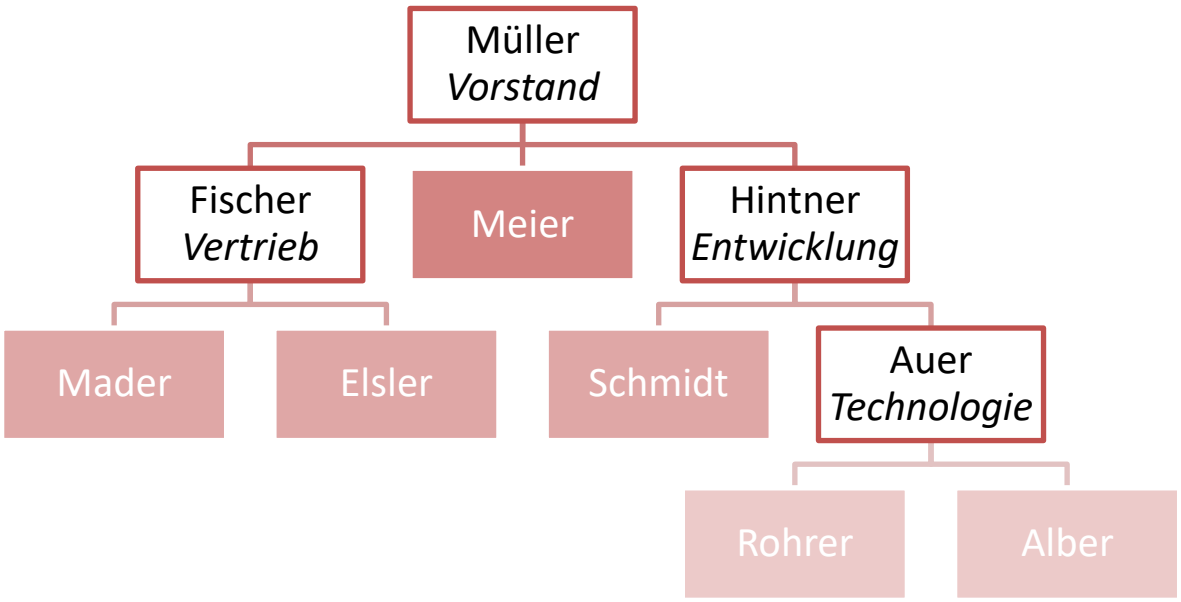
Vorteile	Nachteile
Dynamischer Mechanismus: Referenz auf ein anderes Objekt kann zur Laufzeit bestimmt werden Kapselung wird nicht verletzt	Code muss oft mehrmals implementiert werden Viele Klassen Systemverhalten ist nicht mehr in einer Klasse definiert (hängt von Klassenbeziehungen ab)

Black Box Reuse

Implementierung der Schnittstelle der Basisklasse und deren Integration als Membervariable die jederzeit geändert werden kann

```
public class MyArrayList<T> implements List<T> {
  private List<T> l = new ArrayList<T>();
  public boolean add(T o) {
    return l.add(o);
  }
}
```

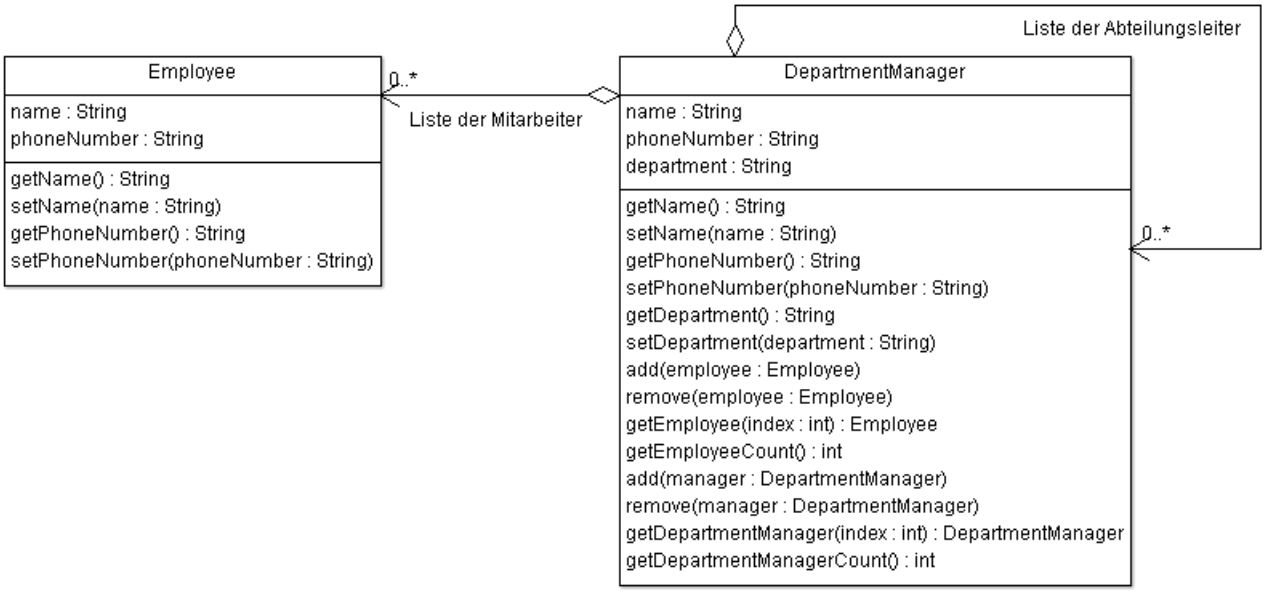
Einleitendes Beispiel: Hierarchiebaum einer Firma



Anforderungen

- Mitarbeiter/Abteilungsleiter besitzen Namen und Telefonnummer (`getName()`, `getPhoneNumber()`)
- Abteilungsleiter kennt Namen seiner Abteilung (`getDepartment()`)
- Anhand von Operationen sollen Mitarbeiter im Baum verwaltet werden können (`add()`, `remove()`, `getEmployee()`)
- Jeder Abteilungsleiter soll die Anzahl der Mitarbeiter/Abteilungsleiter in seiner Abteilung kennen (`getEmployeeCount()`)
- Die gesamte Hierarchie solle ausgedruckt werden können (`print()`)

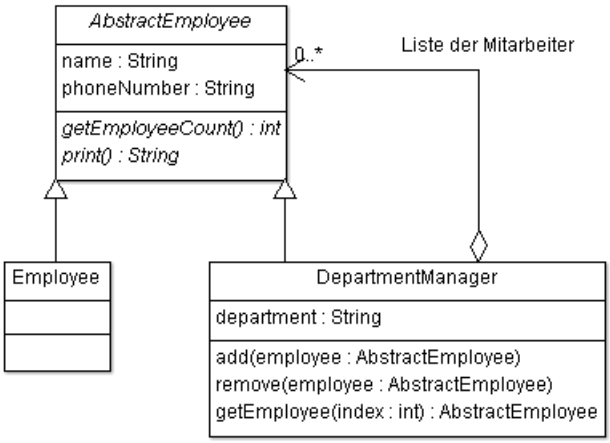
Schlechte Lösung



- Code doppelt vorhanden
- Der Verwender dieser Struktur und Abteilungsleiter müssen im Code ständig zwischen Mitarbeiter und Abteilungsleiter unterscheiden

Bessere Lösung

Einführung der abstrakten Klasse `AbstractEmployee`, von dieser wird normaler Mitarbeiter und Abteilungsleiter abgeleitet



- Getter- und Setter-Methoden für Name und Telefonnummer müssen nur einmal codiert werden
- Abteilungsleiter führt nur eine Liste mit Mitarbeitern und Abteilungsleitern. Mitarbeiter und Abteilungsleiter werden gleichbehandelt
- Abstrakte Methoden `getEmployeeCount()` und `print()` werden erst in Subklassen zwingend implementiert

Vorteile

Hierarchie kann sehr einfach verwendet werden (`print()`, `getEmployeeCount()`). Struktur kümmert sich rekursiv selbständig und die Abarbeitung der Operationen

Es muss nicht mehr zwischen Mitarbeiter und Abteilungsleiter unterschieden werden, weder innerhalb der Objekte noch außerhalb beim Verwender der Struktur

Bestehende Klassen können einfach um weitere Klassen erweitert werden (z.B. Fachgruppenleiter, Vertreter)

Gleiche Lösung kann auch in anderen Situationen verwendet werden: Dateisystem, GUI-Programmierung, Menüs, usw.

Entwurfsmuster (Design Pattern)

Konkretes Beispiel wird als Kompositum (Composite Pattern) bezeichnet

Entwurfsmuster

- Entwurfsmuster sind nicht neu (Architektur der Häuser)
- Entwurfsmuster beschreibt eine bewährte Schablone für ein Entwurfsproblem „battle-proven“
- „Das Rad nicht immer neu erfinden“
- Dient zur besseren Kommunikation zwischen Entwicklern
- Beschreibt ein Problem welches immer wieder auftritt
- Beschreibt den allgemeinen Kern der Lösung
- Bietet Anhaltspunkt (guideline) zum Gegensatz zu „anti-pattern“

Entwurfsmuster werden beschrieben durch

Name

Problem

- Beschreibt die Anwendung
- Es wird Problem und dessen Umfeld beschrieben

Composite

- *"Füge Objekte zu Baumstrukturen zusammen, um Teil-Ganzes-Hierarchien zur repräsentieren. Das Kompositionsmuster ermöglicht es Klienten, einzelne Objekte sowie Kompositionen von Objekten einheitlich zu behandeln."*

[1]

Lösung

- Beteiligte Klassen mit Verantwortungen
- Beziehungen
- Zusammenspiel der Klassen
- Kann als allgemeine Schablone (Template) aufgefasst werden

---

[1] Nach Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: "Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software". Addison-Wesley. 1. Auflage 1996 (***Gang of Four***)

Unsere Klassenbezeichnung	Allgemeine Klassenbezeichnung
`AbstractEmployee`	`Component`
`Employee`	`Leaf`
`DepartmentManager`	`Composite`

Auswirkungen

- Wie wirkt sich der Einsatz aus?
- Vor- bzw. Nachteile

Einteilung

Klassenmuster	Objektmuster
• Beschreiben Beziehungen zwischen Klassen • Benutzen meist Vererbungsstrukturen • Compile Time	• Beschreiben Beziehungen zwischen Objekten • Benutzen meist Assoziation und Aggregation/Kompo-sition • Runtime

Arten

Erzeugermuster – Creational Patterns

- Wie kann Erzeugung von Objekten unterstützt werden
- Entkoppeln die Konstruktion eines Objektes von seiner Repräsentation

Strukturmuster – Structural Patterns

- Wie können Klassen/Interfaces verbunden werden, um Objekte zu erzeugen, die eine neue Funktionalität realisieren

Verhaltensmuster – Behavioral Patterns

- Wie kann man Interaktion zwischen Objekten und komplexen Algorithmen/Kontrollflüssen beschreiben
- Wie gruppiert man Objekte um Aufgaben zu lösen, die nicht von einem Objekt alleine gelöst werden können

		Zweck		
		Erzeugermuster	**Strukturmuster**	**Verhaltensmuster**
Bereich	Klasse	Fabrikmethode *(Factory Method)*	Adapter (klassen-basiert) *(Adapter(class))*	Interpreter *(Interpreter)* Schablonenmethode *(Template Method)*
	Objekt	Abstrakte Fabrik *(Abstract Factory)* Erbauer *(Builder)* Prototyp *(Prototype)* Singleton *(Singleton)*	Adapter (objekt-basiert) *(Adapter(object)* Brücke *(Bridge)* Dekorierer *(Decorator)* Fassade *(Facade)* Fliegengewicht *(Flyweight)* Kompositum *(Composite)* Stellvertreter *(Proxy)*	Befehl *(Command)* Beobachter *(Observer)* Besucher *(Visitor)* Iterator *(Iterator)* Memento *(Memento)* Strategie *(Strategy)* Vermittler *(Mediator)* Zustand *(State)* Zuständigkeitskette *(Chain of Responsibility)*

und viele mehr…