Technische Hochschule Rosenheim
Technical University of Applied Sciences

Objektorientierte Programmierung
Kapitel 12 – Refactorings

Prof. Dr. Kai Höfig

Inhalt

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Wiederholung UML
- Refactorings
- Code Smells
- Design Patterns

Vererbung (Wiederholung)

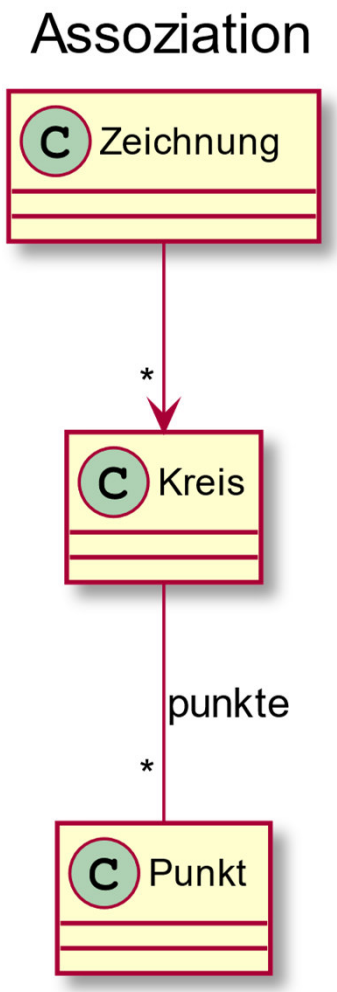
Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Vererbung bildet in der OOP eine *ist-ein*-Beziehung ab
  - Der Mensch ist ein Säugetier
- Systematische Spezialisierung von oben nach unten
- Die *Unterklasse* erbt damit alle Merkmale der *Oberklasse*
- Unterklassen erben alle zugreifbaren Member der Basisklasse:
  - Konstruktor und Destruktor
  - Attribute / Klassenvariablen
  - Methoden und Operatoren
- Was wird nicht vererbt?
  - Member die als *private* deklariert sind

Assoziation

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- lose Verbindung zwischen Objekten (Referenz)
- Objekte kennen sich (Richtung der Referenz)
- Kardinalitäten
- Name



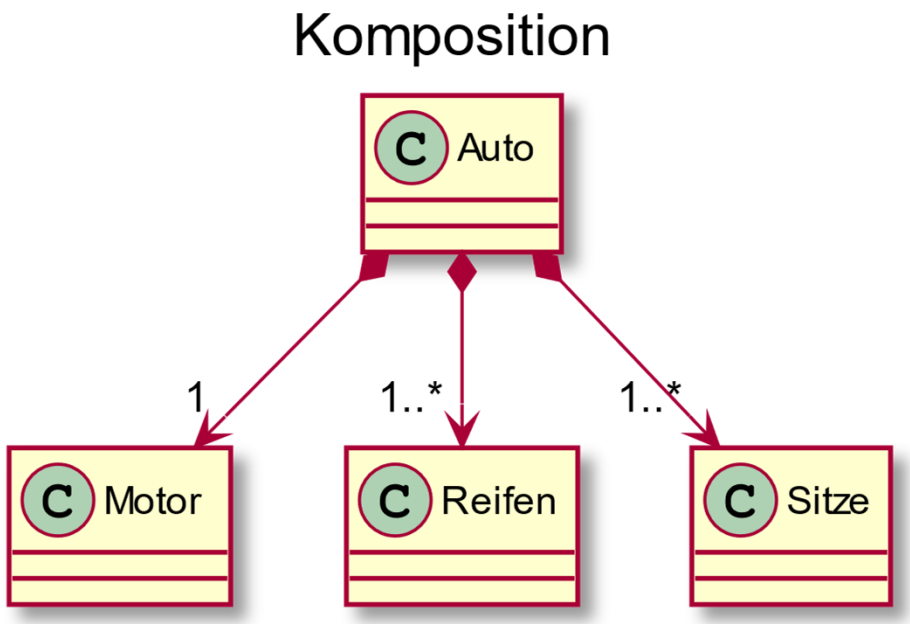
Komposition

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- reale und komplexe Objekte bestehen meist aus kleinen und einfachen Objekte
  - Auto besteht aus Reifen, Motor, Sitzen ...
  - PC besteht aus CPU, Motherboard, RAM, ...

- im objektorientierten Paradigma nennt man diese Beziehung: **Komposition**

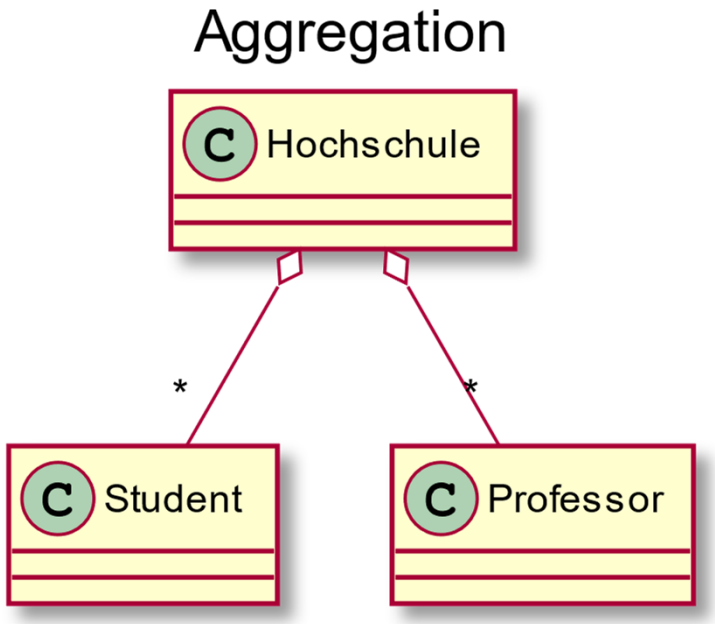
- bildet eine *besteht aus* oder *hat ein* Beziehung ab



Aggregation

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Aggregation ist eine spezielle Form der Komposition
- bildet auch eine *hat ein* Beziehung ab
- 'Besitz'-Klasse hat jedoch keine Besitzansprüche
  - Referenzierten Klassen leben weiter und werden nicht zerstört wenn die Klasse zerstört wird
  - Referenzierten Klassen werden auch nicht automatisch erstellt wenn die referenzierende Klasse erstellt wird



Refactoring

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- **Improving the Design of Existing Code**

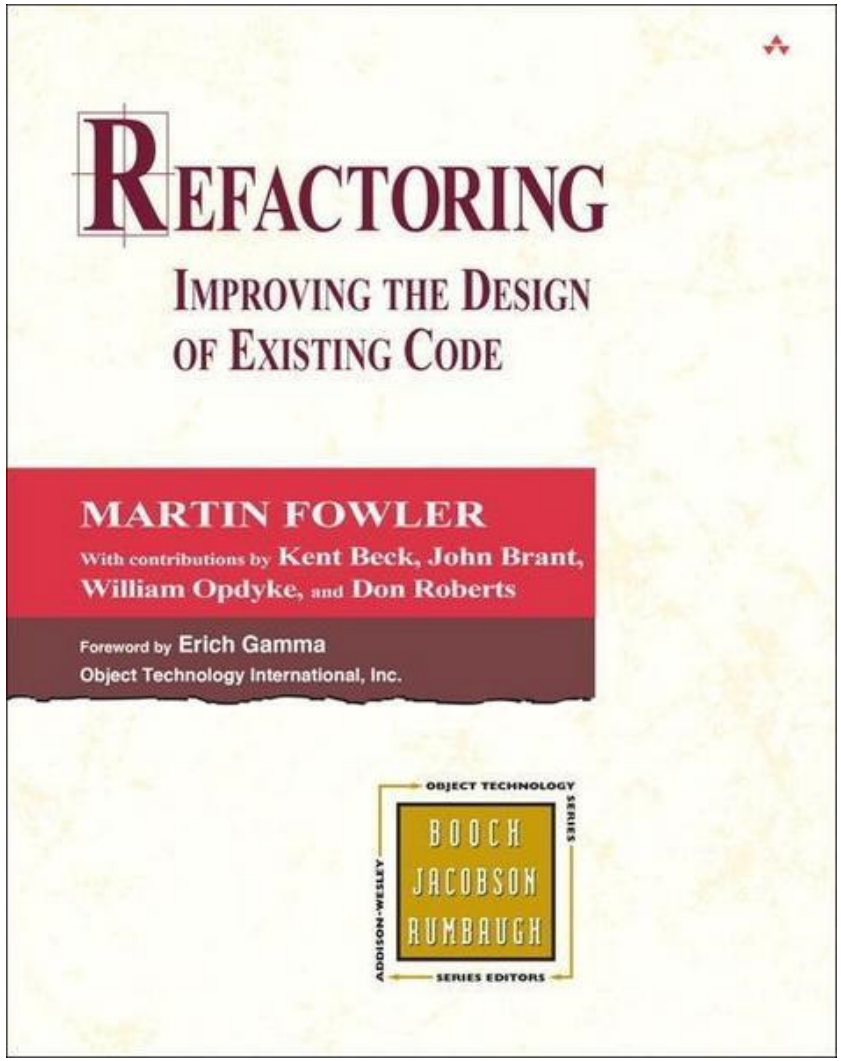
- „A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour.“ -- Martin Fowler,1999

- **Zur Geschichte**

- Ward Cunningham und Kent Beck beginnen bereits in den 1980er bei ihren Arbeiten mit Smalltalk explizit mit Refactoring. Sie erarbeiten eine Softwareprozess Xtreme Programming (XP), bei dem Refactoring integraler Bestandteil ist. [-1999]
- Ebenso arbeitet Ralph Johnson bei seinen Arbeiten zu Frameworks mit Refactoring [~1990]
- Die erste wissenschaftliche Arbeit schreibt William Opdyke zu diesem Thema „Erhalt von Semantic beim Einsatz von Refactoring“. Von ihm stammt auch die erste „Refactoring“ Liste [1992]
- Auf Basis dieser Ideen entwickeln John Brant, Don Roberts & Ralph Johnson den Refactoring Browser in Smalltalk [1996]

Buch

Technische Hochschule Rosenheim
Technical University of Applied Sciences



Warum Refactoring?

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Erhöht die Qualität des Designs
  “Building it this way is stupid, but we did it that way because it was faster”
  “Ugh, I’ll come back to this later and fix it up”
  “We should have done X, but ...”
- Verbessert die Verständlichkeit des Codes
- Hilft bei der Fehlersuche
- Verbessert die Wiederverwendbarkeit

Wann Refactoring?

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Copying and pasting is your enemy!
- Gerne kopiert man Fehler!
- Teilweise kopiert man Code, den man nicht versteht!

- also besser : „Refactor“
- bevor neue Funktionalität hinzukommt!
- um Fehler zu finden!
- beim Codereview!
- wenn „Code Smells“

Code Smells

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- A Code Smell is a hint that something has gone wrong somewhere in your code [. . .] Note that a CodeSmell is a hint that something might be wrong, not a certainty [. . .] Calling something a CodeSmell is not an attack; it’s simply a sign that a closer look is warranted.

- -- http://xp.c2.com/CodeSmells.html

Code Smell – „Duplizierter Code oder Bad Design“

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Vereinheitliche Code, wenn dieselbe Codestruktur an mehreren Stellen auftaucht: » Gleicher Ausdruck in Methoden derselben Klasse » Gleicher Ausdruck in Geschwisterunterklassen » Code ist ähnlich aber nicht gleich (gibt es hier Möglichkeiten?) » Methoden machen dasselbe mit einem anderen Algorithmus

Code Smell – “Gleicher Ausdruck in Unterklassen”

Technische Hochschule Rosenheim
Technical University of Applied Sciences

```
public class Employee {
}
```

```
public class Salesman {
    String getName(){
        return "";
    }
}
```

```
public class Engineer {
    String getName(){
        return "";
    }
}
```

- Wird zu…

```
public class Employee {
    String getName(){
        return "";
    }
}
```

```
public class Salesman {
}
```

```
public class Engineer {
}
```

Refactoring Technik – “Pull Up to Super”

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Wenn Code ist gemeinsame Code über alle Subklassen, gehört er in die Superklasse.

```
class Manager extends Employee {
    public Manager(String name, String id, int grade) {
        this.name = name;
        this.id = id;
        this.grade = grade;
    }
    // ...
}
```

- Wird zu...

```
class Manager extends Employee {
    int grade;
    public Manager(String name, String id, int grade) {
        super(name, id);
        this.grade = grade;
    }
    // ...
}
```

Code Smell – “Switch Statements”

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Switch Statements sind oft Hinweise auf schlechtes objektorientiertes Design. Aus der Klasse `Bird`, kann die Klasse `EuropeanBird` abgeleitet werden, und das Switch Statement verschwindet.

```
class Bird {

    public static final int EUROPEAN = 1;
    public static final int AMERICAN = 2;
    public static final int NORWEGIAN_BLUE = 3;

    int type;

    public double getSpeed() {
        switch (type) {
            case EUROPEAN:
                return getBaseSpeed();
            case AMERICAN:
                return getBaseSpeed() - numberOfLoad();
            case NORWEGIAN_BLUE:
                return getBaseSpeed()*2;
        }
        throw new RuntimeException("Unreachable");
    }
…
}
```

```
public class EuropeanBird extends Bird {
    public double getSpeed() {
        return getBaseSpeed();
    }
}
```

Code Smell – “Comments as Design”

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Kommentare sollten nicht dazu genutzt werden, den Gebrauch vorzuschreiben, sondern nur dokumentieren was die Methode macht.

```
/**
 * NUR FUER INTERNEN GEBRAUCH (Aufruf durch Controller)!
 *
 * Aktualisiert diese View anhand der uebergebenen ViewDef.
 *
 * @param pViewDef ViewDef, anhand diese View aktualisiert werden soll.
 * @return receiver modified
 */
public void updateWith( ViewDef pViewDef ) {
    if (pViewDef==null) return;
    setUpdatePending(false);
    View lView = replaceOrUpdateViewFor( pViewDef );
    lView.updatePresentationMode( pViewDef );
    if ( isReinitialize() )
        lView.toReinitialize();
}
```

Refactoring Technik – “Extract Method”

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Wenn Codefragmente mehrfach verwendet werden oder logisch zusammengehören, verschieben Sie diesen Code in eine separate neue Methode und ersetzen Sie den alten Code durch einen Aufruf der Methode.

```
void printOwing() {
    printBanner();
    // Print details.
    System.out.println("name: " + name);
    System.out.println("amount: " + getOutstanding());
}
```

- Wird zu..

```
void printOwing() {
    printBanner();
    printDetails(getOutstanding());
}

void printDetails(double outstanding) {
    System.out.println("name: " + name);
    System.out.println("amount: " + outstanding);
}
```

Refactoring Technik – “Extract Method”

Technische Hochschule Rosenheim
Technical University of Applied Sciences

```
public void actionPerformed(ActionEvent e) {
    ...
    // out
    for (int i = 0; i < klasses.size(); i++) {
        System.out.println(klasses.get(i));
    }
    // compile
    for (int i = 0; i < klasses.size(); i++) {
        Compiler.compileClass((Class) klasses.get(i));
    }
    // create database
    DatabaseBuilder lDatabase = new DatabaseBuilder(
            "mysql", "jdbc.mysql.MySqlDriver");
    try {
        lDatabase.executeSrcipt(script, ";");
    } catch (Exception ex) {
        ex.printStackTrace();
    }
    // load classes
    Method lMethod = lClass.getDeclaredMethod(
            "newInstance", new Class[]{Properties.class});
        ...
}
```

Refactoring Technik – “Remove Dead Code”

Technische Hochschule Rosenheim
Technical University of Applied Sciences

```
if(false) {
    doSomethingThatUsedToMatter();
}
```

- Wird zu..

```
//nichts
```

Refactoring Technik – “Replace Constructor with Factory Function”

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Wenn in den Konstruktoren mehr gemacht wird, als die Parameter eines Objekts zu initialisieren, lohnt es sich eine Fabrikmethode zu benutzen. Sonst kann es zu kaskadierenden Konstruktoraufrufen kommen, die schwer zu koordinieren sind.

```
public class Employee {
    String name;

    Employee(String name) {
        this.name = name;
    }
..
}
```

Wird zu

```
public class Employee {
    String name;

    private Employee(String name) {
        this.name = name;
    }

    static Employee create(String name) {
        Employee e = new Employee(name);
        // do some heavy lifting.
        return e;
    }

}
```

Design Pattern

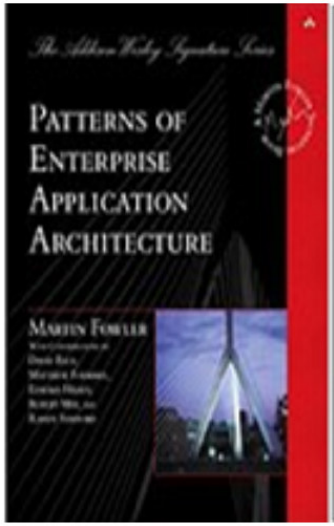
Technische Hochschule Rosenheim
Technical University of Applied Sciences

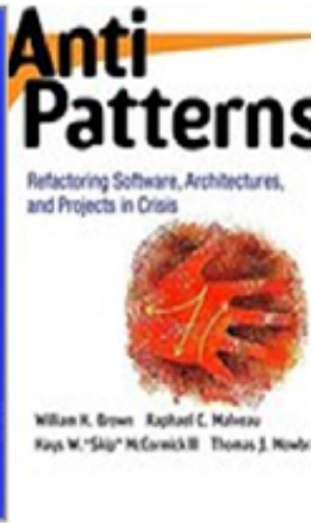
- Design Pattern (Entwurfsmuster) sind bewährte Lösungswege für wiederkehrende Designprobleme in der Softwareentwicklung
- Sie beschreiben die essenziellen Entwurfsentscheidungen (Klassen- und Objektarrangements)
- Durch den Einsatz von Design Pattern wird ein Entwurf flexibel, wiederverwendbar, erweiterbar, einfacher zu verwenden und änderungsstabil
- In den Design Patterns manifestiert sich die jahrelange Berufserfahrung vieler Softewareentwicklern.
- Zeitgleich schulen Design Pattern die Fähigkeit zur effektiven objektorientierten Modellierung

Design Pattern

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Do not reinvent the wheel!
- Use Patterns and learn from others!





The “gang of four” (GoF): Design Pattern

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- 23 verschiedene Design Patterns (Java, C++ & Smalltalk)
- 3 Kategorien (creational, structural, behavioral)

- Definierte Beschreibungsstruktur
  - Name (einschließlich Synonyme)
  - Aufgabe und Kontext
  - Beschreibung der Lösung
  - Struktur (Komponenten, Beziehungen)
  - Interaktionen und Konsequenzen
  - Implementierung
  - Beispielcode

Design Pattern in Kategorien – Creational Pattern

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Abstraktion
  - Macht ein System unabhängig davon, wie sein Objekte instanziiert, zusammengesetzt und repräsentiert werden

- Factory
- Abstract Factory
- Builder
- Prototype
- Singleton

Design Pattern in Kategorien – Structural Pattern

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Zusammensetzung von Klassen/Objekten zu größeren Strukturen, um neue Funktionalität zu realisieren

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

Design Pattern in Kategorien – Behavioral Pattern

Technische Hochschule Rosenheim
Technical University of Applied Sciences

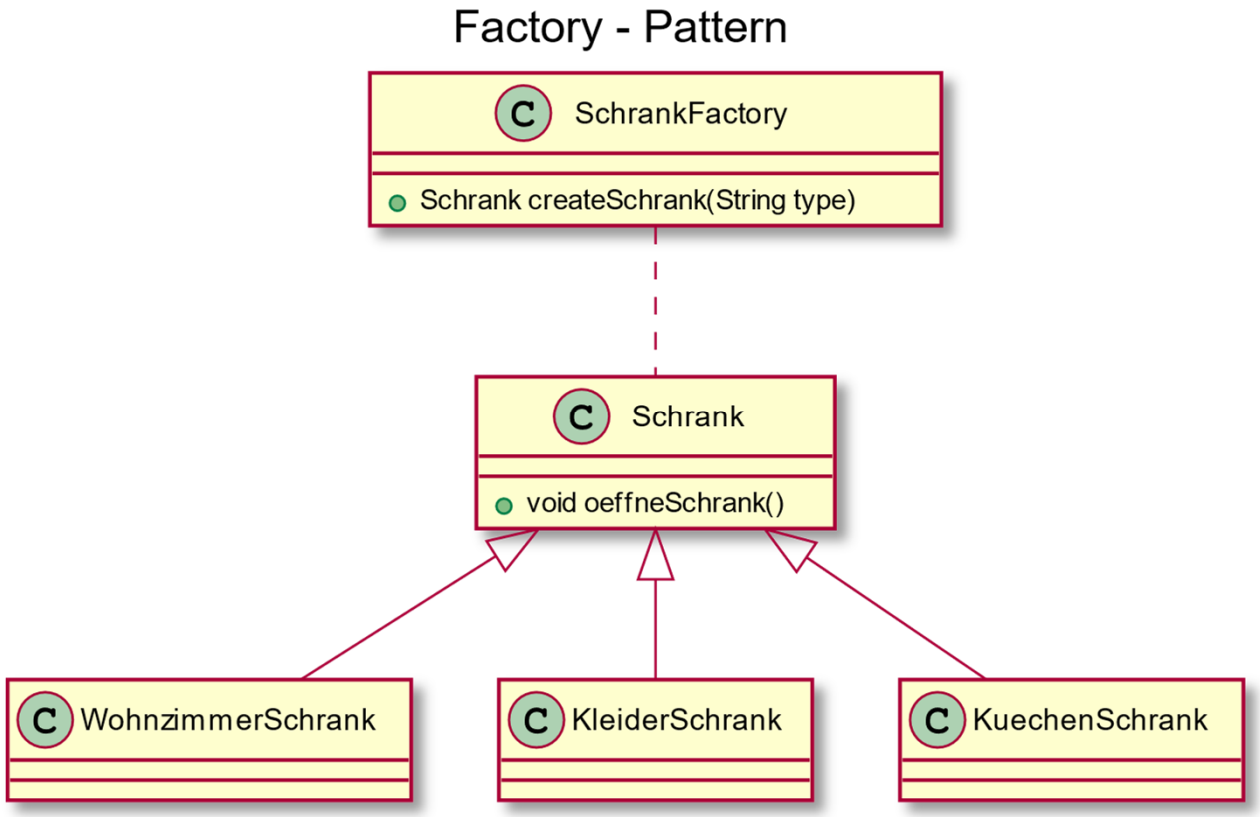
Zusammensetzung, Arbeitsteilung, Verantwortlichkeiten zwischen Klassen oder Objekten

- Interpreter
- Template Method
- Chain of Responsibility
- Command
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Visitor

Design Pattern – *Factory*

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Das Factory Method Entwurfsmuster dient der Entkopplung des Clients von der konkreten Instanziierung einer Klasse. Das erstellte Objekt kann elegant ausgetauscht werden. Oft wird es zur **Trennung von (zentraler) Objektverarbeitung und (individueller) Objektherstellung** verwendet.



Design Pattern – *Factory*

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Die (mitunter) komplexe Erstellung von Objekten, die ja auch wiederum aus anderen Objekten bestehen können, wird hierbei nicht mehr von dem Aufrufer oder der Verwendung eines Schranks aus bewerkstelligt, sondern, ähnlich wie bei der Fabrikmethode, in eine Methode ausgelagert. Bei diesem Muster wird allerdings nicht nur ein Typ erstellt, sondern mehrere Sub-Typen.

Aus..

```
public static void main(String[] args) {
    Schrank schrank = SchrankFactory.
            createSchrank("Kleiderschrank");
    schrank.oeffneSchrank();
}
```

..wird

```
public static void main(String[] args) {
    Schrank schrank = new Kleiderschrank();
    schrank.oeffneSchrank();
}
```

Design Pattern – *Factory*

Technische Hochschule Rosenheim
Technical University of Applied Sciences

```
public class SchrankFactory {
    public static Schrank createSchrank(String schrankArt) {
        if (schrankArt.equals("Kleiderschrank")){
            return new Kleiderschrank();
        }
        if (schrankArt.equals("Kuechenschrank")){
            return new Kuechenschrank();
        }
        if (schrankArt.equals("Wohnzimmerschrank")){
            return new Wohnzimmerschrank();
        }
        return null;
    }
}
```

Design Pattern: Observer-Pattern

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Das Observer-Pattern ist eines der am meisten genutzten und bekanntesten Patterns
- In diesem Muster gibt es zwei Akteure: Ein Subjekt, welches beobachtet wird und ein oder mehrere Beobachter, die über Änderungen des Subjektes informiert werden wollen
- Die Idee des Observer-Patterns ist es, dem zu beobachtenden Subjekt die Aufgabe aufzutragen, die Beobachter bei einer Änderung über die Änderung zu informieren
- Die Beobachter müssen nicht mehr in regelmäßigen Abständen beim Subjekt anfragen, sondern können sich darauf verlassen, dass sie eine Nachricht über eine Änderung erhalten

Observable and Observer

Technische Hochschule Rosenheim
Technical University of Applied Sciences

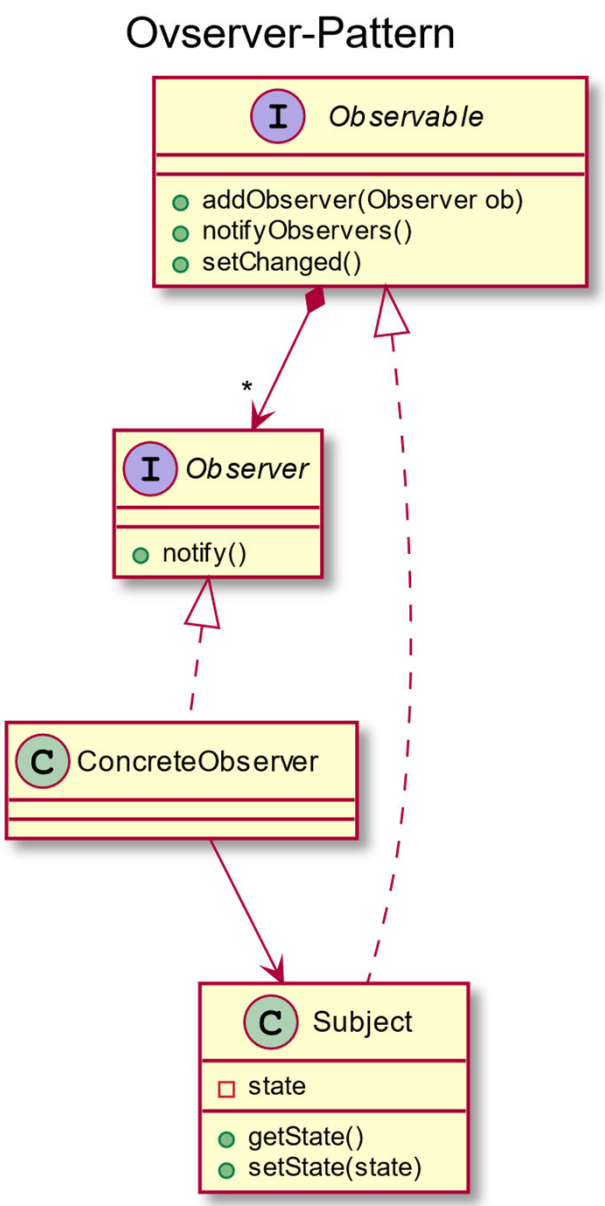
```
class Erzaehler extends Observable {

    public Erzaehler(){
        this.addObserver(new Zuhoerer_1());
        this.addObserver(new Zuhoerer_2());
        tell("laber, laber...");
    }


    public void tell(String info){
        if(countObservers()>0){
            setChanged();
            notifyObservers(info);
        }
    }
}
```

Design Pattern

- **Observer-Pattern (II)**



Design Pattern – *Singleton*

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Das Singleton-Entwurfsmuster gilt im Bezug auf die Komplexität als das einfachste Entwurfsmuster, da es nur aus einer einzigen Klasse besteht.

- Eine Klasse wird immer dann als Singleton implementiert, wenn es nur eine einzige Instanz von ihr geben darf.

- Singleton garantiert, dass es von einer Klasse höchstens eine Instanz geben kann und bietet dazu einen globalen Zugriffspunkt auf diese Instanz.

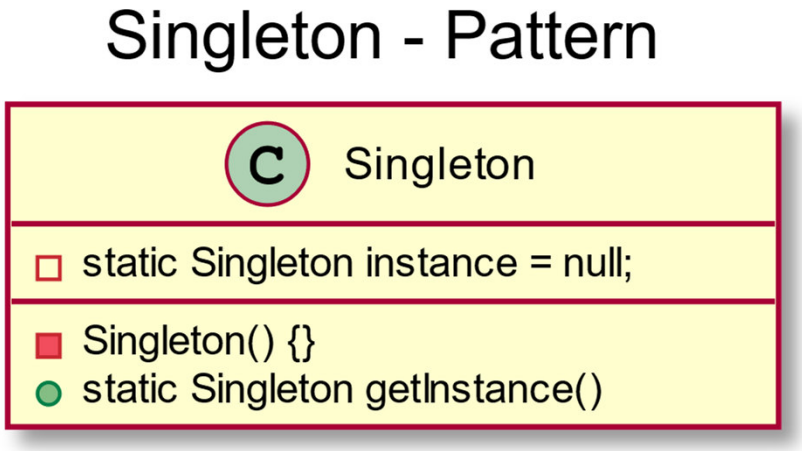
- Beispiele:
  - Datenbankverbindungen
  - Dialoge
  - Logging-Objekte
  - Objekte, die globale Daten und Einstellungen verwalten

Design Pattern – *Singleton*

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Wie bereits erwähnt, besteht das Singleton-Entwurfsmuster nur aus einer einzigen Klasse. Diese verfügt im Wesentlichen über die folgenden Merkmale:

- Eine statische Variable des gleichen Typs
- Einen privaten Konstruktor
- Einer statischen Methode, welche die Instanz zurück gibt



Design Pattern – *Singleton*

Technische Hochschule Rosenheim
Technical University of Applied Sciences

```
class Singleton {
    private static Singleton instance = null;

    private Singleton() {
        /* Initialisation goes here! */
    }

    public static Singleton getInstance() {
        if (Singleton.instance == null) {
            Singleton.instance = new Singleton();
        }
        return Singleton.instance;
    }

}
```

Design Pattern – *Composite*

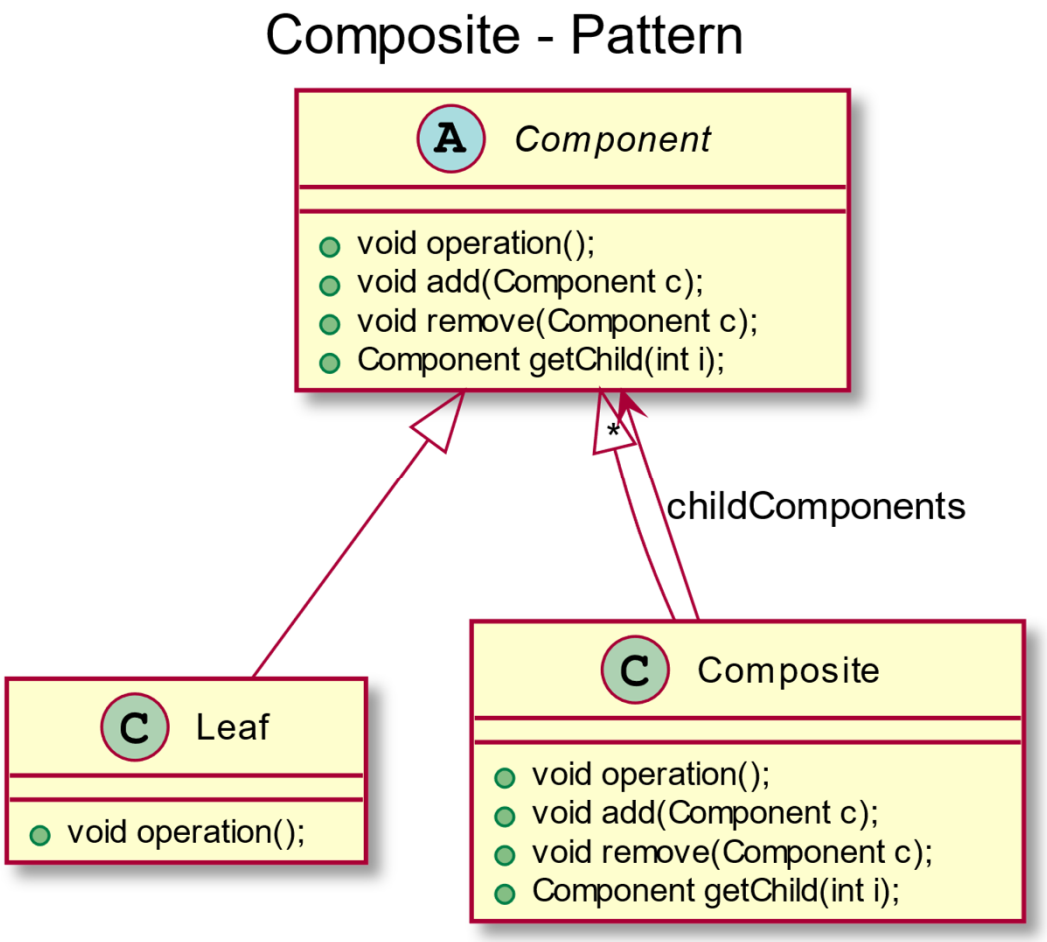
Technische Hochschule Rosenheim
Technical University of Applied Sciences

Das Composite Entwurfsmuster ermöglicht es, eine verschachtelte Struktur einheitlich zu behandeln, unabhängig davon, ob es sich um ein atomares Element oder um ein Behälter für weitere Elemente handelt

- Es wird eine gemeinsame Schnittstelle für die Elementbehälter (Composite, Kompositum; Aggregat, Knoten) und für die atomaren Elemente (Leaf, Blatt) definiert: Component.
- Diese Schnittstelle Component definiert die Methoden, die gleichermaßen auf Composites und auf Leafs angewandt werden sollen. Composites delegieren oft Aufrufe (operate()) an ihre Components, die atomare Leafs oder wiederrum zusammengesetzte Composites seien können.
- Ein Client muss nicht mehr zwischen Composite und Leaf unterscheiden!

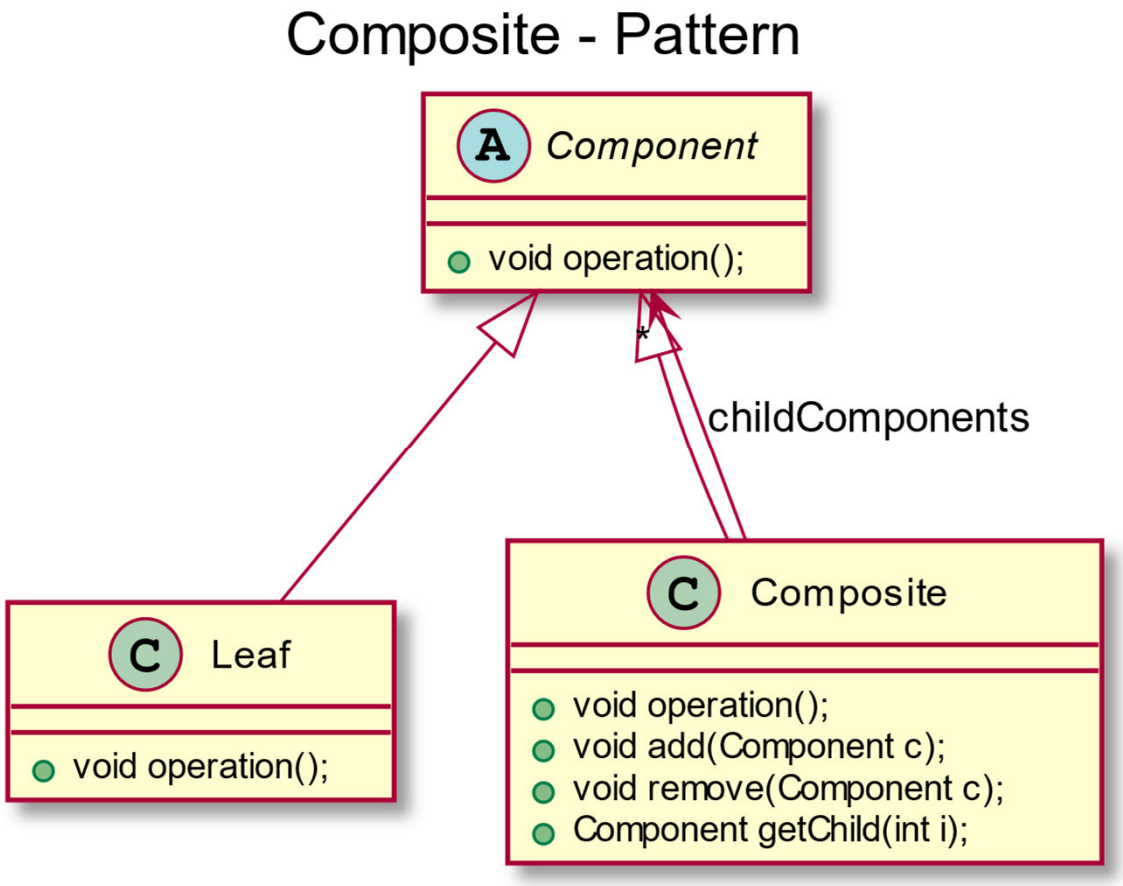
Design Pattern – *Composite*

Technische Hochschule Rosenheim
Technical University of Applied Sciences



Design Pattern – *Composite* (Alternative)

Technische Hochschule Rosenheim
Technical University of Applied Sciences



Design Pattern – *Composite*

Technische Hochschule Rosenheim
Technical University of Applied Sciences

```
class Composite extends Component{

    //hier: Components als Liste vorgehalten
    private List<Component> childComponents = new ArrayList<Component>();

    //rekursiver Aufruf auf kindComponents
    public void operation() {
        System.out.println("Ich bin ein Composite. Meine Kinder sind:");
        for (Component childComps : childComponents) {
            childComps.operation();
        }
    }

    //Überschreiben der Defaulimplementierung
    public void add(Component comp) {
        childComponents.add(comp);
    }
    public void remove(Component comp) {
        childComponents.remove(comp);
    }
    public Component getChild(int index) {
        return childComponents.get(index);
    }
}
```

Referenzen

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Design Patterns @ Refactoring Guru
- TutorialPoint - Design Pattern
- [GeeksForGeeks - Design Pattern] (https://www.geeksforgeeks.org/design-patterns-set-1-introduction/)
- OODesign - GoF Pattern