Klausur

- Termin: Mittwoch, der **20.09.23**, **14 – 16** Uhr
- Dauer: **120 Minuten**
- Ort: **HZO [10, 20, 30, 40]**
  - Hörsaalaufteilung wird zwei Wochen vorher über Moodle angekündigt
- Erlaubte Hilfsmittel: Ein beidseitig handbeschriebenes DIN A4 Blatt

Evaluierung der Lehrveranstaltung

https://tinyurl.com/23xygmq6

Ankündigung

- Die Vorlesung am 26.06. entfällt.
- Die Übung am 27.06. findet statt und kann als Wiederholungsstunde genutzt werden.
- Die Onlineübung am 30.06. entfällt.

RUHR-UNIVERSITÄT BOCHUM
Lehrstuhl für Informatik im Bauwesen

RUB

Programmierung und Programmiersprachen

Sommersemester 2023

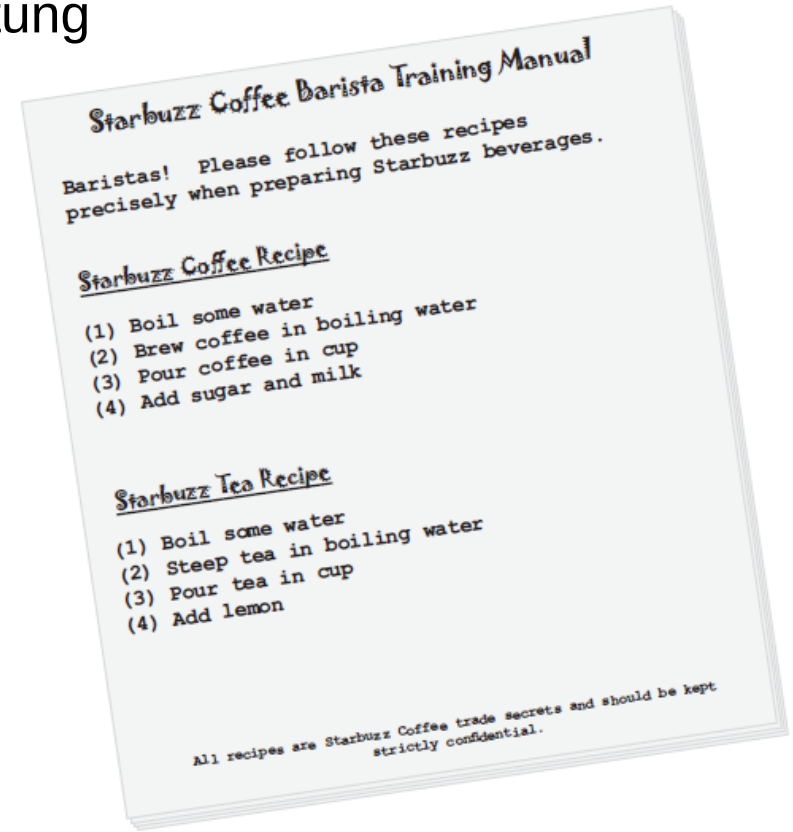
Template-Method-Muster

Template-Method-Muster

Ähnliche Methoden

In einem Kaffeeladen soll demnächst auch Tee verkauft werden.
Hierzu wurde für die Mitarbeiter eine Anleitung
für die Zubereitung von Tee ergänzt.

Was fällt dabei auf?



Starbuzz Coffee Barista Training Manual

Baristas! Please follow these recipes precisely when preparing Starbuzz beverages.

Starbuzz Coffee Recipe

(1) Boil some water
(2) Brew coffee in boiling water
(3) Pour coffee in cup
(4) Add sugar and milk

Starbuzz Tea Recipe

(1) Boil some water
(2) Steep tea in boiling water
(3) Pour tea in cup
(4) Add lemon

All recipes are Starbuzz Coffee trade secrets and should be kept strictly confidential.

Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Template-Method-Muster

Beispiel – Zubereitung von Kaffee und Tee

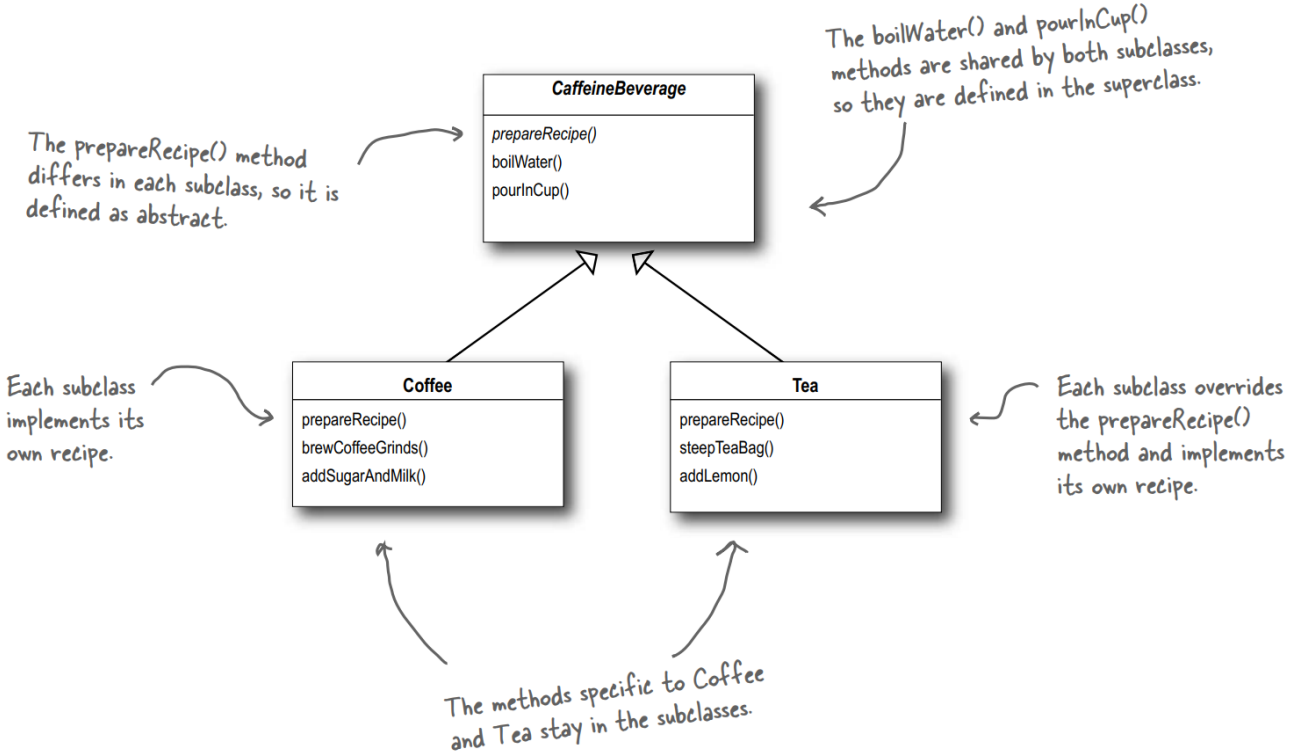
```
public class Coffee {

    public void prepareRecipe() {
        boilWater();
        brewCoffeeGrinds();
        pourInCup();
        addSugarAndMilk();
    }

    public void boilWater() {
        System.out.println("Boiling water");
    }

    public void brewCoffeeGrinds() {
        System.out.println("Dripping Coffee through filter");
    }

    public void pourInCup() {
        System.out.println("Pouring into cup");
    }

    public void addSugarAndMilk() {
        System.out.println("Adding Sugar and Milk");
    }
}
```

```
public class Tea {

    public void prepareRecipe() {
        boilWater();
        steepTeaBag();
        pourInCup();
        addLemon();
    }

    public void boilWater() {
        System.out.println("Boiling water");
    }

    public void steepTeaBag() {
        System.out.println("Steeping the tea");
    }

    public void addLemon() {
        System.out.println("Adding Lemon");
    }

    public void pourInCup() {
        System.out.println("Pouring into cup");
    }
}
```

Template-Method-Muster

Re-Design – Abstrakte Klasse

Gleiche Methoden und Implementierungen werden in einer abstrakten Klasse zusammengefasst.

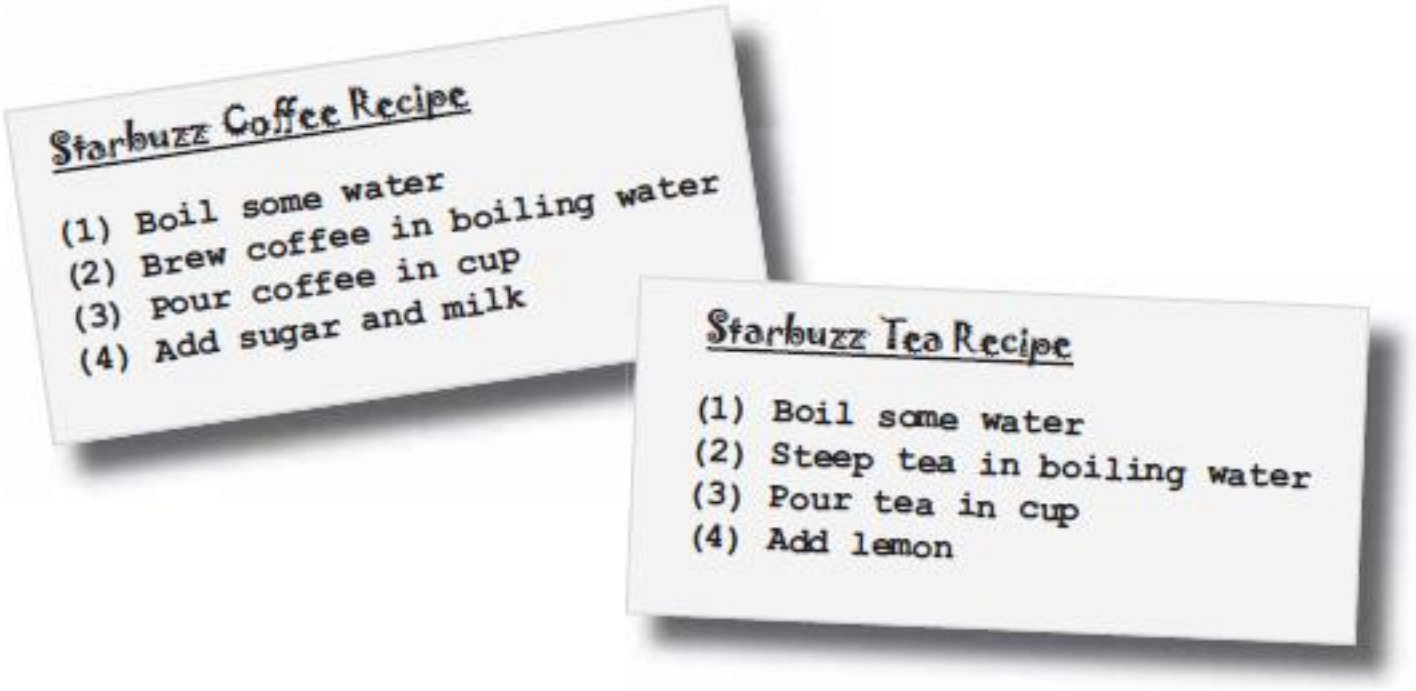


Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Template-Method-Muster

Re-Design – Abstrakte Klasse

Welche weiteren Methoden sind ähnlich und können eventuell vereinheitlicht werden?



Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Template-Method-Muster

Re-Design – Abstrakte Klasse

Die Methoden zum Aufbrühen von Kaffee und Tee sind sehr ähnlich. Des Weiteren werden verschiedene Aromate hinzugefügt

Coffee

```
void prepareRecipe() {
    boilWater();
    brewCoffeeGrinds();
    pourInCup();
    addSugarAndMilk();
}
```

Tea

```
void prepareRecipe() {
    boilWater();
    steepTeaBag();
    pourInCup();
    addLemon();
}
```

Können diese Methoden etwas allgemeiner definiert werden?

Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Template-Method-Muster

Re-Design – Abstrakte Klasse

In der abstrakten Klasse werden zwei neue abstrakte Methoden zum Aufbrühen und Hinzufügen von Aromaten definiert.

```
void prepareRecipe() {
    boilWater();
    brew();
    pourInCup();
    addCondiments();
}
```

Die neuen abstrakten Methoden müssen in den erweiterten Klassen implementiert werden.

Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Template-Method-Muster

Beispiel – Zubereitung von Kaffee und Tee

```
public abstract class CaffeineBeverage {

    final void prepareRecipe() {
        boilWater();
        brew();
        pourInCup();
        addCondiments();
    }

    abstract void brew();

    abstract void addCondiments();

    void boilWater() {
        System.out.println("Boiling water");
    }

    void pourInCup() {
        System.out.println("Pouring into cup");
    }
}
```

Die Methode **`prepareRecipe()`** wird als final deklariert. Dadurch wird verhindert, dass diese durch erweiterte Klassen überschrieben wird.

Template-Method-Muster

Beispiel – Zubereitung von Kaffee und Tee

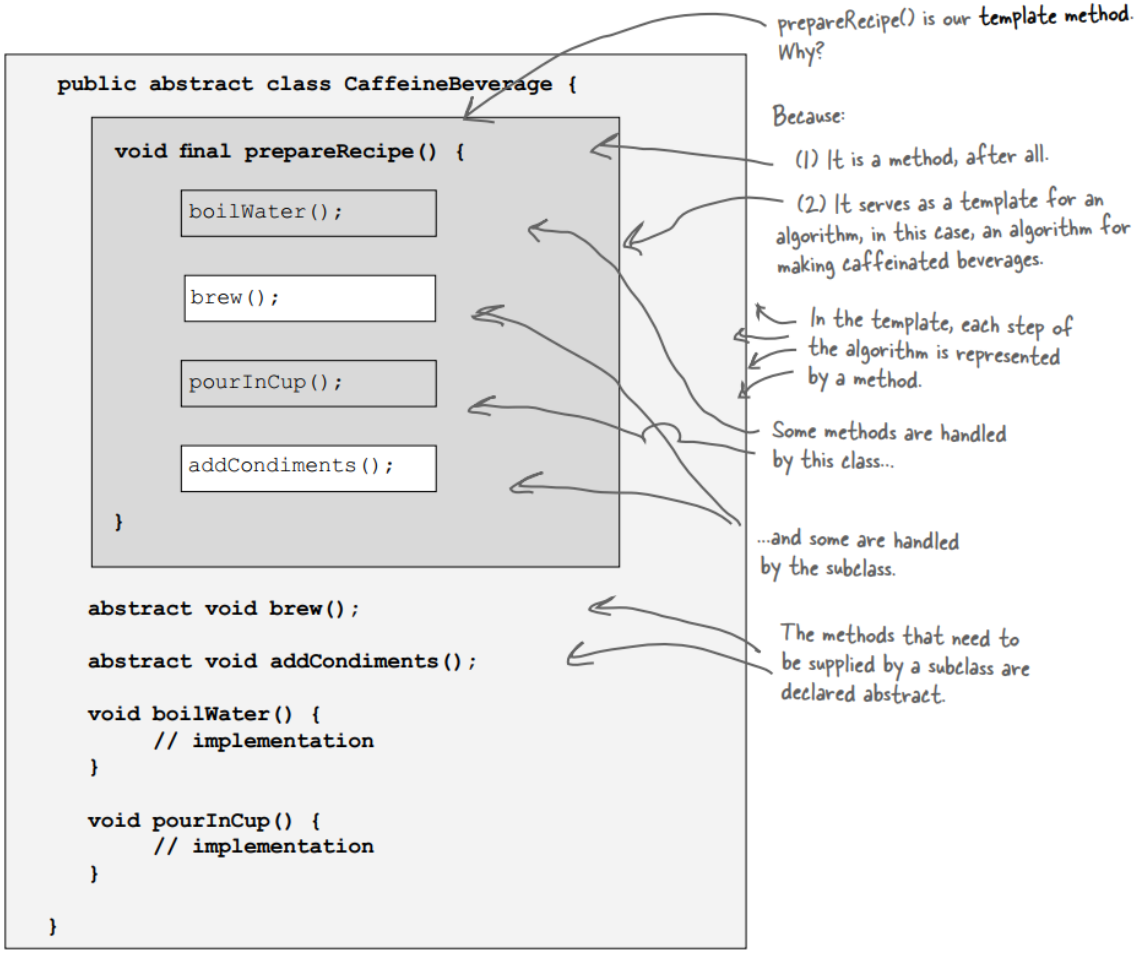
```
public class Coffee extends CaffeineBeverage {

    public void brew() {
        System.out.println("Dripping Coffee through filter");
    }

    public void addCondiments() {
        System.out.println("Adding Sugar and Milk");
    }
}



public class Tea extends CaffeineBeverage {

    public void brew() {
        System.out.println("Steeping the tea");
    }

    public void addCondiments() {
        System.out.println("Adding Lemon");
    }
}
```

Template-Method-Muster

Template Method Muster (Schablonenmuster)

Durch das hier eingesetzte Template Method Muster kann die Struktur eines Algorithmus vorgegeben werden

Die Implementierung der einzelnen Schritte kann an die Unterklassen delegiert werden



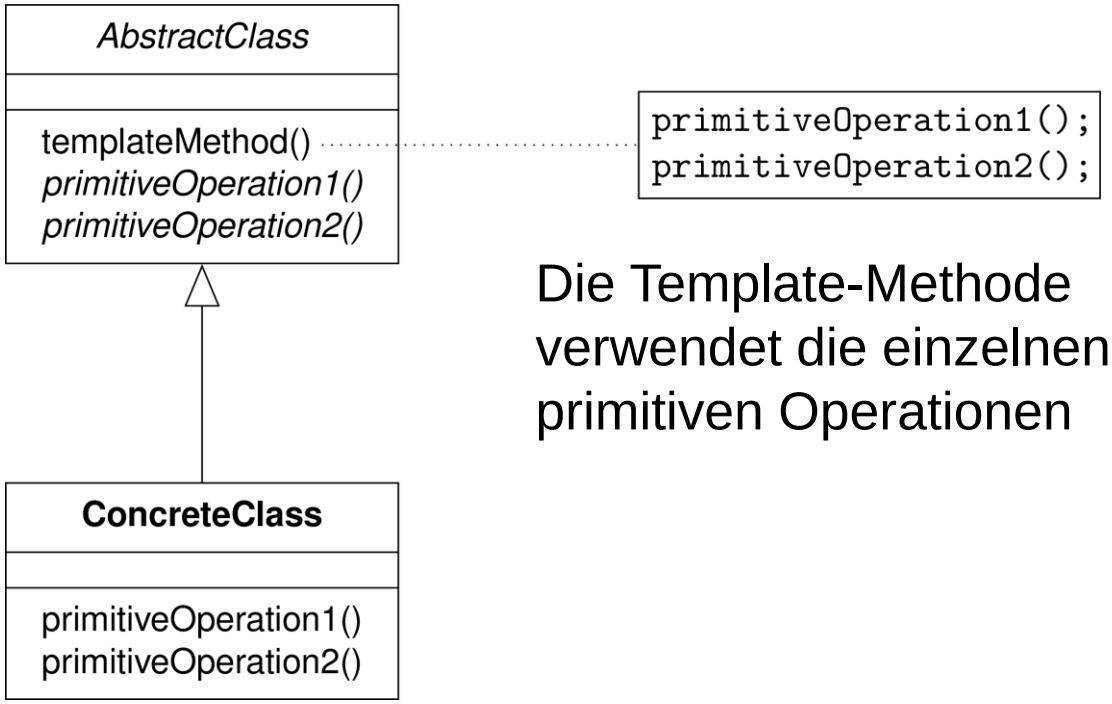
Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Template-Method-Muster

Template Method Muster (Schablonenmuster)

Durch das hier eingesetzte Template Method Muster kann die Struktur eines Algorithmus vorgegeben werden.

Die Implementierung der einzelnen Schritte bzw. Operationen kann an die Unterklassen delegiert werden.



Die Template-Methode verwendet die einzelnen primitiven Operationen

Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Template-Method-Muster

Optionale Schritte

Bei machen Algorithmen sind einzelne Operationen oder Schritte nur optional, d.h. diese werden nicht bei jeder Implementierung benötigt. Zur Vermeidung von Abfragen, können optionale Schritte „leer“ implementiert werden

```
public abstract class CaffeineBeverage {

    final void prepareRecipe() {
        boilWater();
        brew();
        pourInCup();
        addCondiments();
    }

    abstract void brew();

    void addCondiments() {}

    ...
}
```

Der Schritt `addCondiments()` wird „leer„ implementiert und muss somit nicht immer umgesetzt werden

Solche Methoden werden auch als „Hooks“ bezeichnet (Anker für die Implementierung von optionalen Operationen)

Template-Method-Muster

Beispiel – Zubereitung von Kaffee und Tee

```
public class Coffee extends CaffeineBeverage {

    public void brew() {
        System.out.println("Dripping Coffee through filter");
    }
}



public class Tea extends CaffeineBeverage {

    public void brew() {
        System.out.println("Steeping the tea");
    }
}
```

Template-Method-Muster

Beispiel – Zubereitung von Kaffee und Tee

```
public class CoffeeWithSugarAndMilk extends CaffeineBeverage {

    public void brew() {
        System.out.println("Dripping Coffee through filter");
    }

    public void addCondiments() {
        System.out.println("Adding Sugar and Milk");
    }
}
```

```
public class TeaWithLemon extends CaffeineBeverage {

    public void brew() {
        System.out.println("Steeping the tea");
    }

    public void addCondiments() {
        System.out.println("Adding Lemon");
    }
}
```

Template-Method-Muster

Beispiel – Zubereitung von Kaffee und Tee

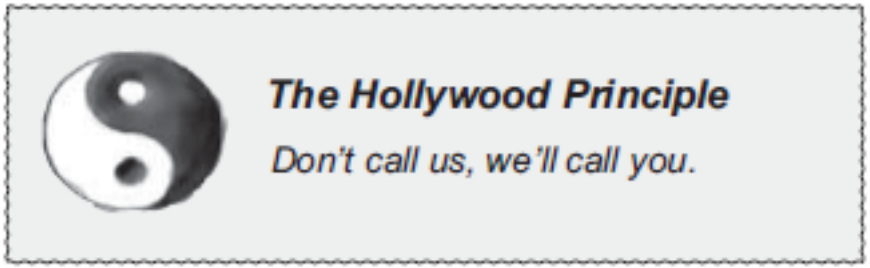
```
public class BeverageTestDrive {

    public static void main(String[] args) {

        Tea tea = new Tea();
        Coffee coffee = new Coffee();

        System.out.println("\nMaking tea...");
        tea.prepareRecipe();

        System.out.println("\nMaking coffee...");
        coffee.prepareRecipe();


        TeaWithLemon teaLemon = new TeaWithLemon();
        CoffeeWithSugarAndMilk coffeeSugarAndMilk = new CoffeeWithSugarAndMilk();

        System.out.println("\nMaking tea...");
        teaLemon.prepareRecipe();

        System.out.println("\nMaking coffee...");
        coffeeSugarAndMilk.prepareRecipe();
    }
}
```

Template-Method-Muster

Hollywood Prinzip

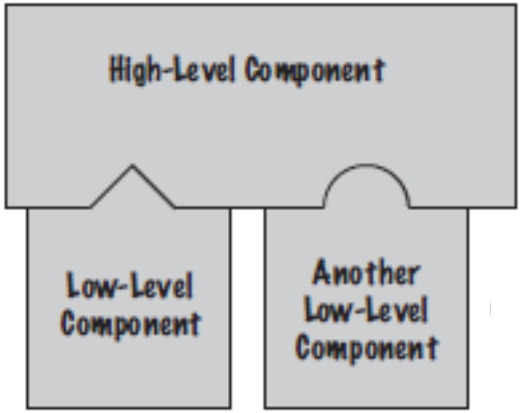
Zur Vermeidung von komplexen Abhängigkeiten sollte das sogenannte Hollywood Prinzip beachtet werden.

Übergeordnete bzw. zusammen-gesetzte Komponenten kennen und verwenden untergeordnete Komponenten.



Die untergeordneten Komponenten kennen die übergeordneten sowie gleichrangigen Komponenten nicht.

Komponenten werden somit nur verwendet, wenn sie wirklich gebraucht werden.



Template-Method-Muster

Template vs. Strategy

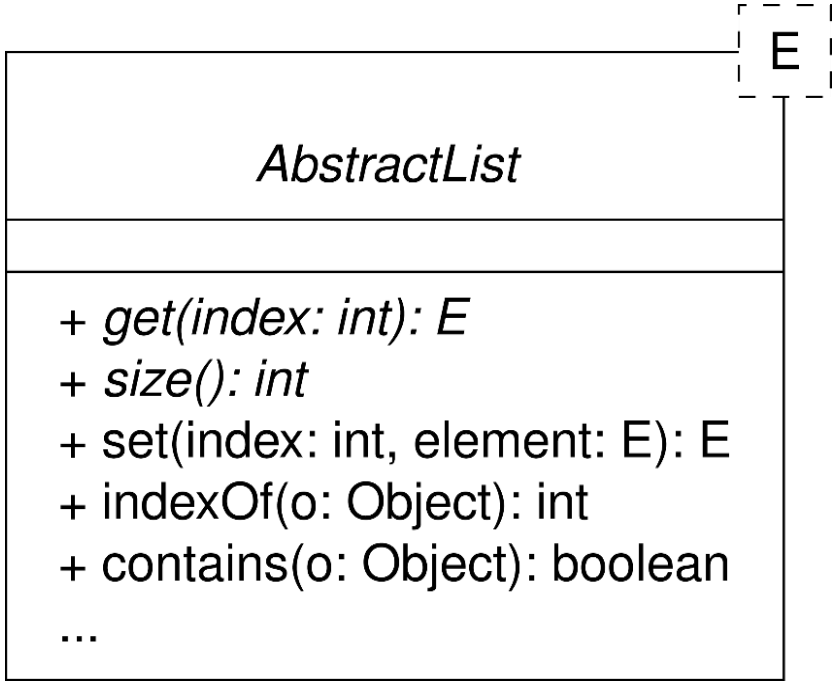
Alle drei Muster kapseln gewisse Funktionen zur Ausführung von bestimmten Operationen

Strategy	*Kapselt austauschbare Operationen eines Algorithmus und delegiert die Entscheidung, wie eine Operation implementiert wird*
Template	*Unterklassen entscheiden wie einzelne Schritte eines Algorithmus implementiert werden*

Template-Method-Muster

Templates Benutzen

Java bietet in der Standardbibliothek bereits mehrere Templates an welche implementiert werden können, z.B. *AbstractList*.

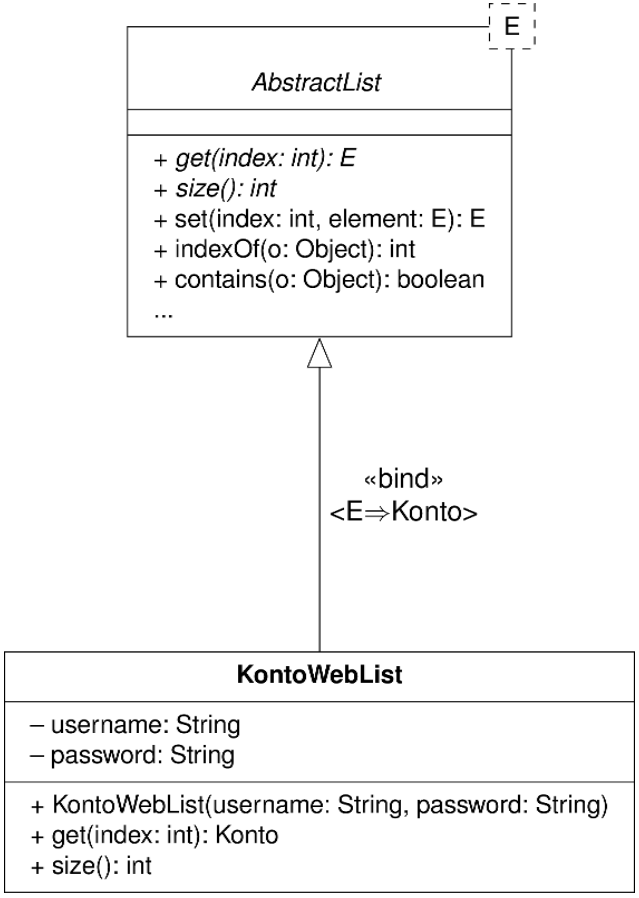


Template-Method-Muster

Templates Benutzen

Beispiel: Wir wollen den Zugriff auf eine Liste von Konten über eine WebAPI zulassen, jedoch sollen die Konten erst abgerufen werden wenn sie benötigt werden.

Wir benutzen das Listen-Template in welchem wir die Funktionen *get()* und *size()* überschreiben.



Template-Method-Muster

Templates Benutzen

```
public class KontoWebList extends AbstractList<Konto> {
    String username;
    String password;

    @Override
    public Konto get(int index) {
        return new Konto(restRequest("https://server.test/konten/id="+index,
                    username, password));
    }

    @Override
    public int size() {
        return Integer.parseInt(restRequest("https://server.test/konten/numIDs",
                    username, password));
    }
}
```

Jeder Aufruf von *get()* und *size()* ruft die aktuellen Daten ab, und nur relevante Konten werden abgerufen.

Template-Method-Muster

Templates Benutzen

KontoWebList kann jetzt wie eine schreibgeschützte Liste verwendet werden:

```
KontoWebList kontoweblist = new KontoWebList(user, pw);
Konto k1 = kontoweblist.get(1);
```

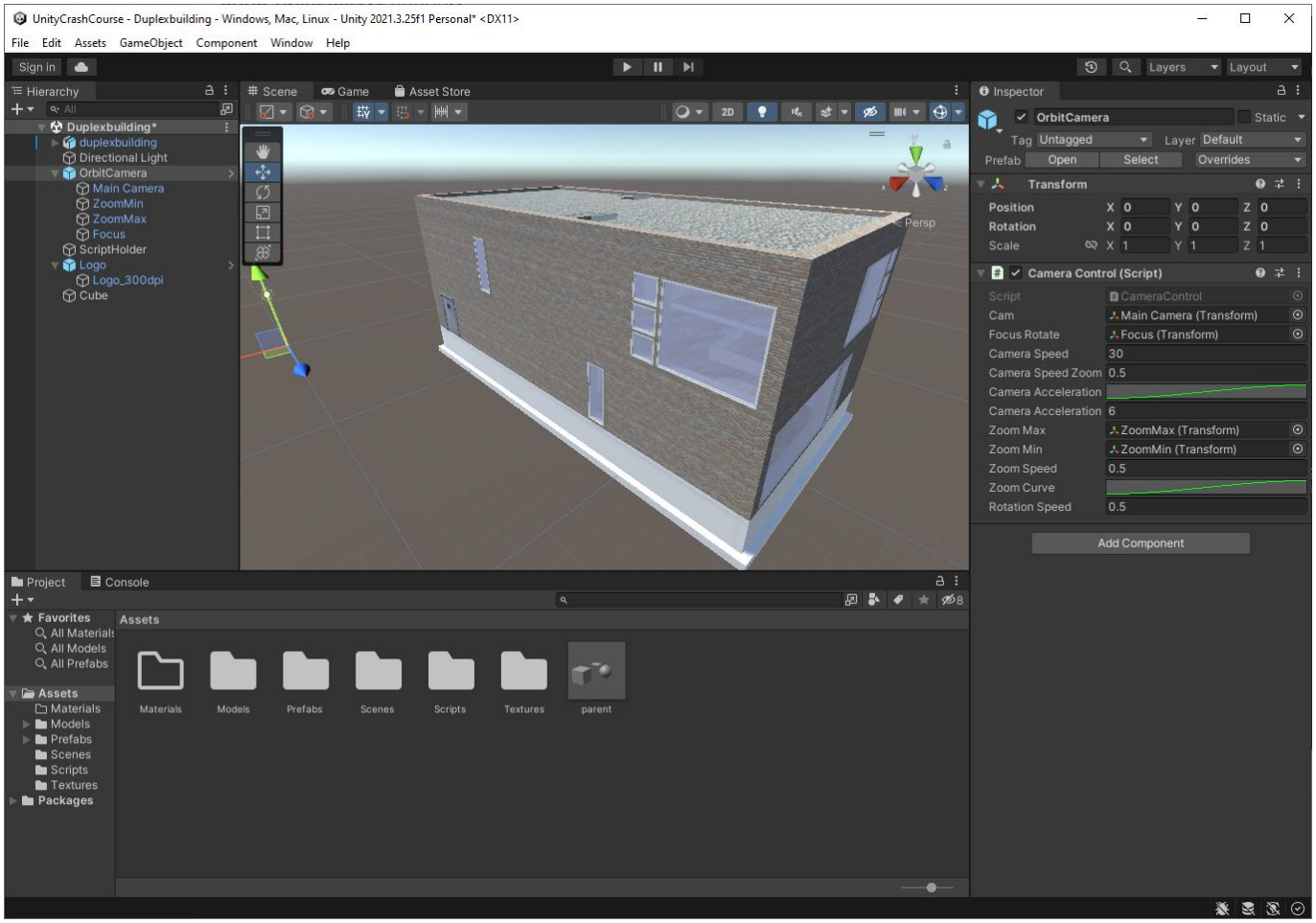
Funktionen wie Suchen oder Kopieren sind bereits von *AbstractList* implementiert:

```
boolean kontoExists = kontoweblist.contains(kontoGesucht);
int kontoIndex = kontoweblist.indexOf(kontoGesucht);
ArrayList<Konto> cache = new ArrayList<Konto>(kontoweblist);
```

Wird die *set()*-Funktion überschrieben kann man auch Daten wieder zurückschreiben. Durch Überschreiben der *add()* und *remove()*-Funktionen kann die Liste eine veränderbare Größe haben.

Unity 3D

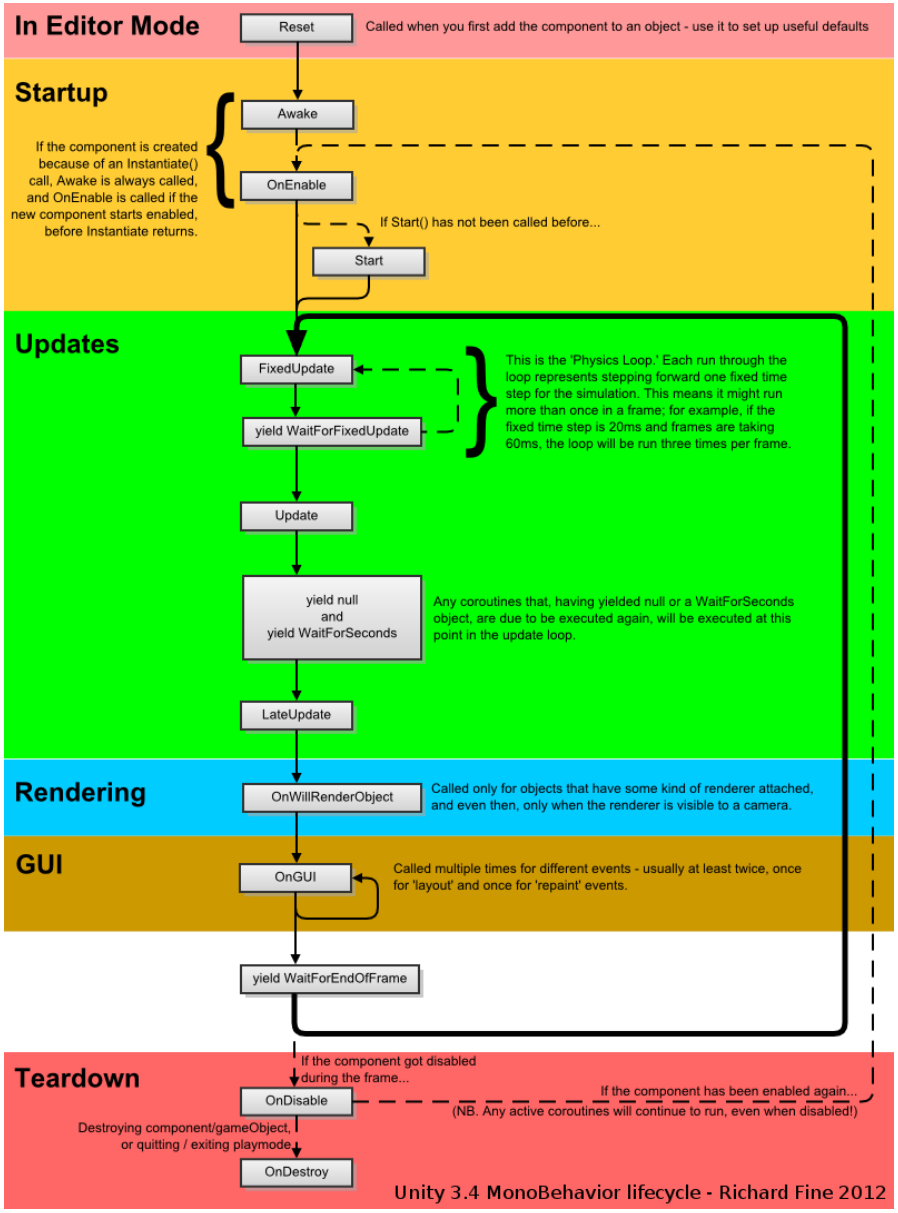
Beispiel Template-Muster: Unity3D



Unity 3D

Script-Template

```
public class MyBehaviour : MonoBehaviour
{
    // Start is called before the first
    // frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```



RUHR-UNIVERSITÄT BOCHUM
Lehrstuhl für Informatik im Bauwesen

RUB

Programmierung und Programmiersprachen

Sommersemester 2023

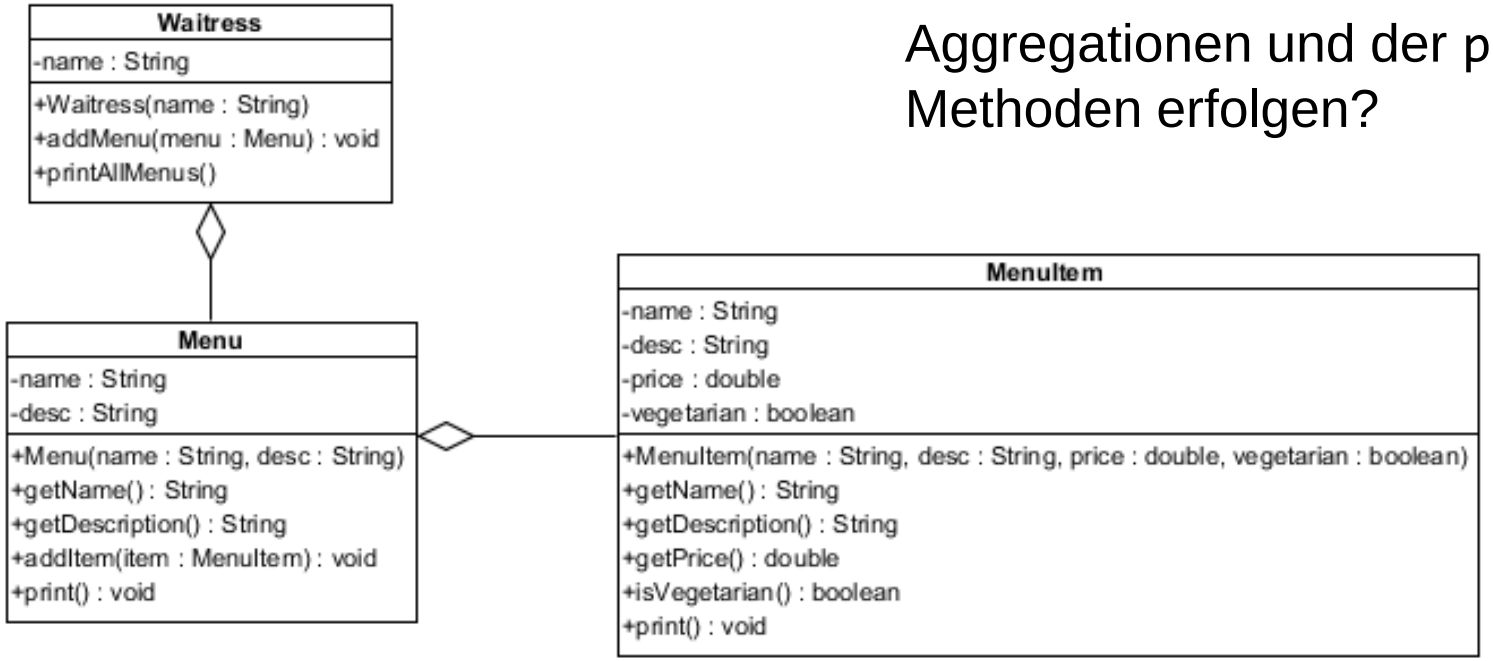
Composite Muster

Composite-Muster

Erweiterbarkeit durch dynamische Strukturen

Es soll eine Zusammenstellung von Menüs eines Restaurants zur Unterstützung der Servicekräfte entwickelt werden. Gegeben sind die folgenden Klassen:

Wie könnte die Umsetzung der Aggregationen und der `print` Methoden erfolgen?



Composite-Muster

Beispiel – Menüverwaltung

```
public class MenuItem {

    private String name;
    private String description;
    private double price;
    private boolean vegetarian;

    public MenuItem(String name, String description, boolean vegetarian, double price) {
        this.name = name;
        this.description = description;
        this.vegetarian = vegetarian;
        this.price = price;
    }
    ...

    public void print() {
        System.out.print("  " + getName());
        if (isVegetarian()) {
            System.out.print("(v)");
        }
        System.out.println(", " + getPrice());
        System.out.println("     -- " + getDescription());
    }
}
```

Composite-Muster

Beispiel – Menüverwaltung

```
public class Menu {

    private ArrayList<MenuItem> items = new ArrayList<MenuItem>();
    private String name;
    private String description;

    public Menu(String name, String description) {
        this.name = name;
        this.description = description;
    }

    public void add(MenuItem item) {
        items.add(item);
    }
    ...

    public void print() {
        System.out.print("\n" + getName());
        System.out.println(", " + getDescription());
        System.out.println("---------------------");
        for (MenuItem item : items) {
            item.print();
        }
    }
}
```

Composite-Muster

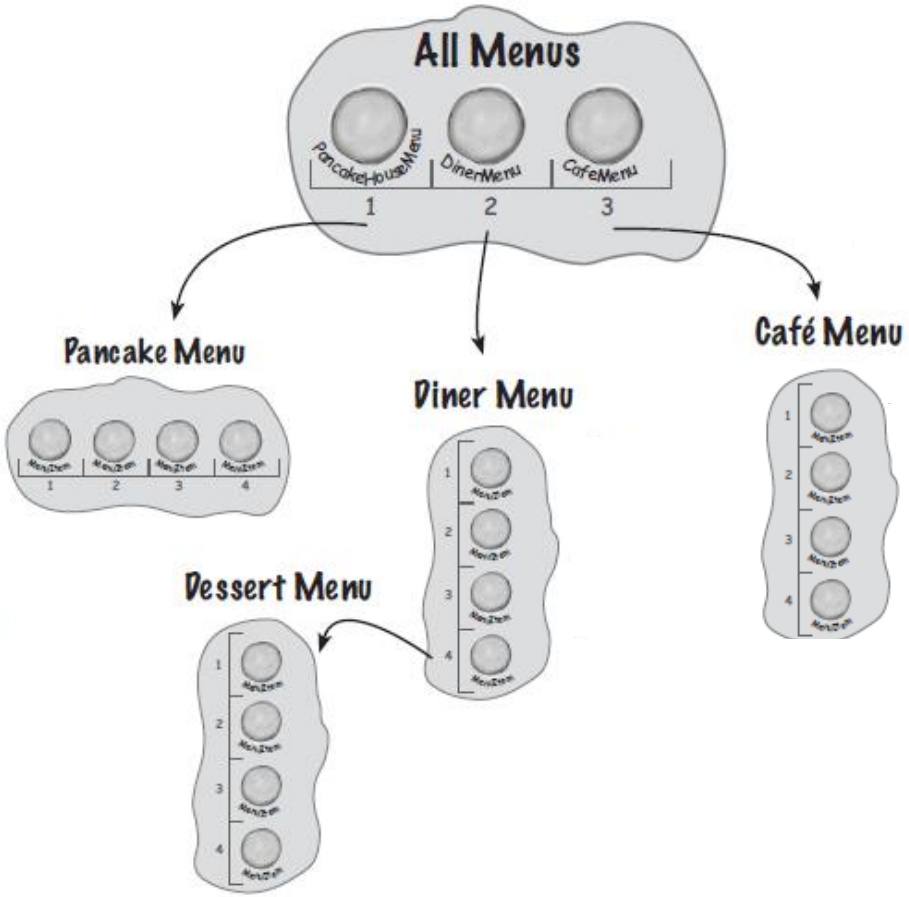
Beispiel – Menüverwaltung

```
public class Waitress {

    private ArrayList<Menu> menus = new ArrayList<Menu>();
    private String name;

    public Waitress(String name) {
        this.name = name;
    }

    public void addMenu(Menu menu) {
        this.menus.add(menu);
    }

...

    public void printAllMenus() {
        System.out.print("\n" + getName());
        System.out.println("---------------------");
        for (Menu menu : menus) {
            menu.print();
        }
    }
}
```

Composite-Muster

Hierarchische Organisation

In einem Restaurant sollen die verschiedenen Menüs, wie folgt, hierarchisch für die Servicekräfte organisiert werden.

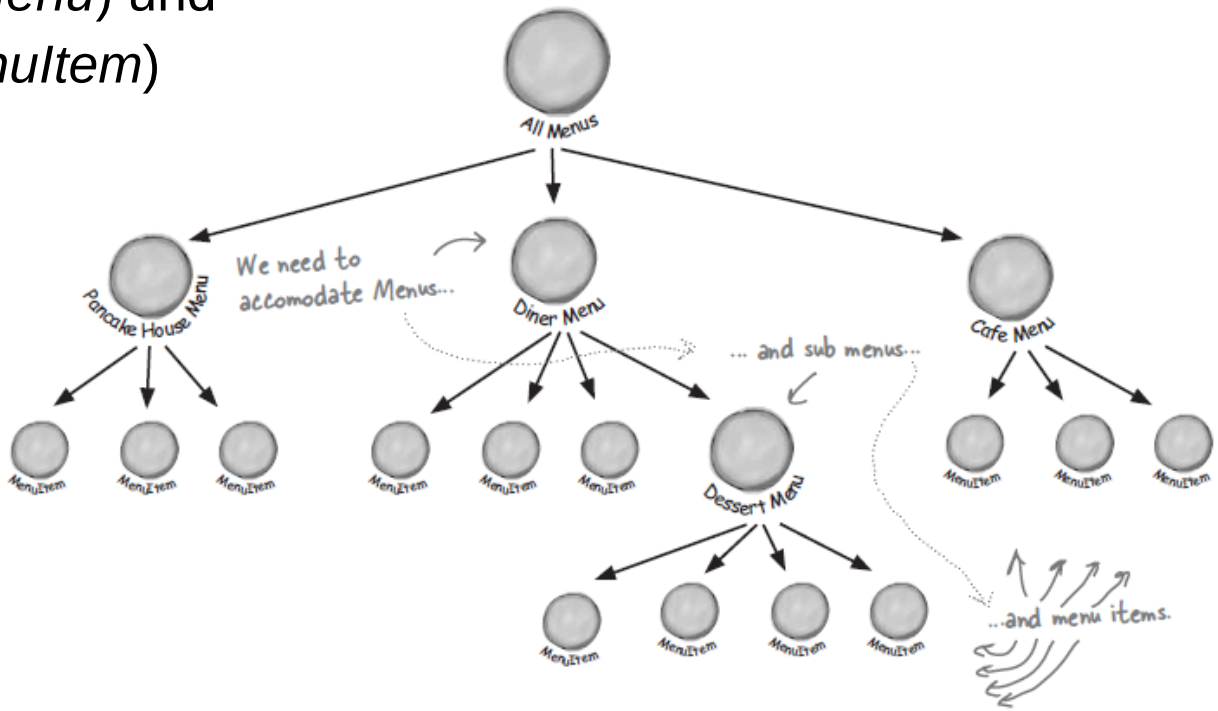


Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Composite-Muster

Re-Design – Baumstruktur

Hierarchische Strukturen werden häufig als Baumstrukturen abgebildet. Ein Baum besitzt eine Wurzel (hier *All Menus*), verschiedene Knoten (bzw. Teilbäume – hier z.B. *Cafe Menu*) und sogenannte Blätter (hier *MenuItem*)

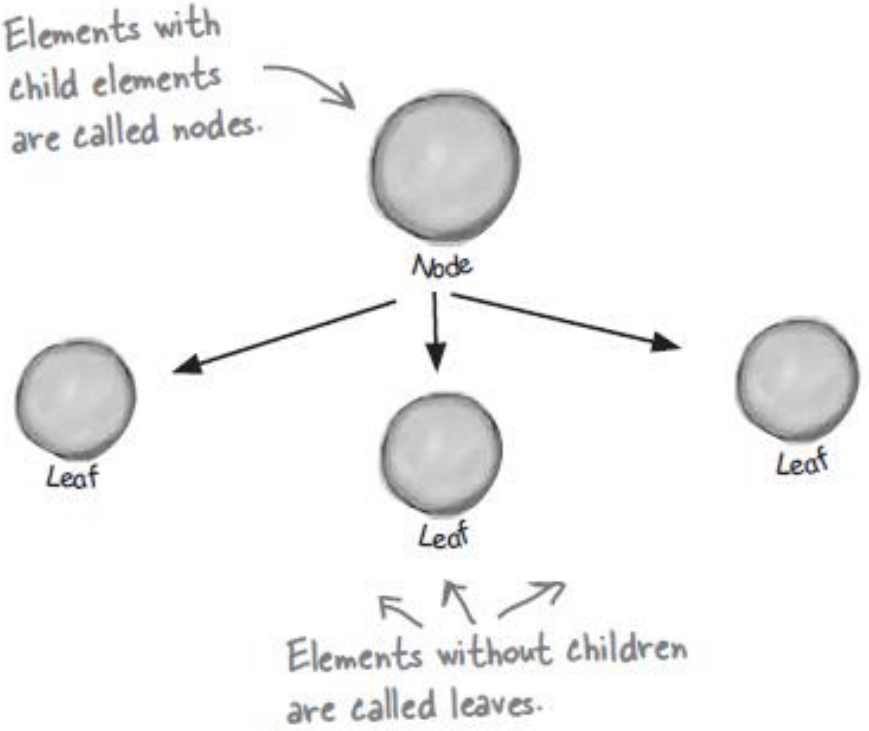


Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Composite-Muster

Composite Muster

Das sogenannte Composite Muster ermöglicht die hierarchische Organisation von Objekten auf Basis einer Baumstruktur. Dadurch können einzelne Objekte und Gruppen von Objekten einheitlich behandelt werden.



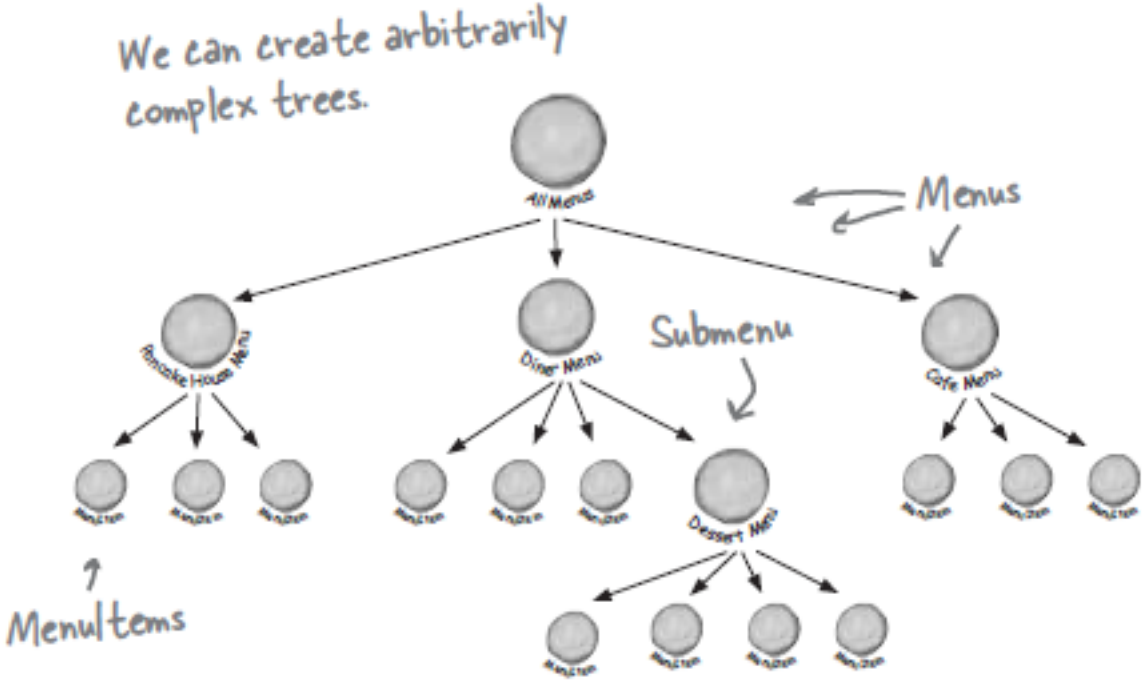
Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Composite-Muster

Composite Muster

Der hierarchische Aufbau kann dynamisch verändert werden.

Operationen können auf den gesamten Baum, Teilbäume oder einzelne Blätter angewendet werden.



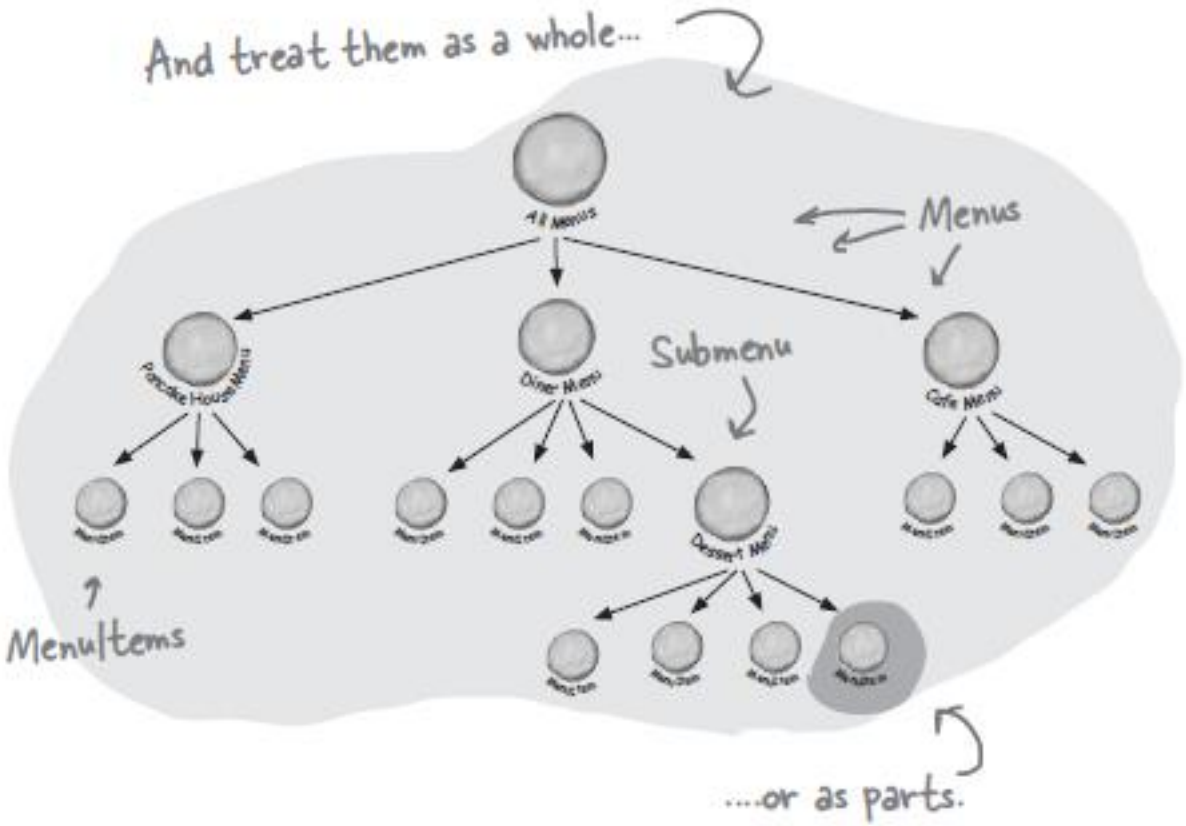
Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Composite-Muster

Composite Muster

Der hierarchische Aufbau kann dynamisch verändert werden.

Operationen können auf den gesamten Baum, Teilbäume oder einzelne Blätter angewendet werden.



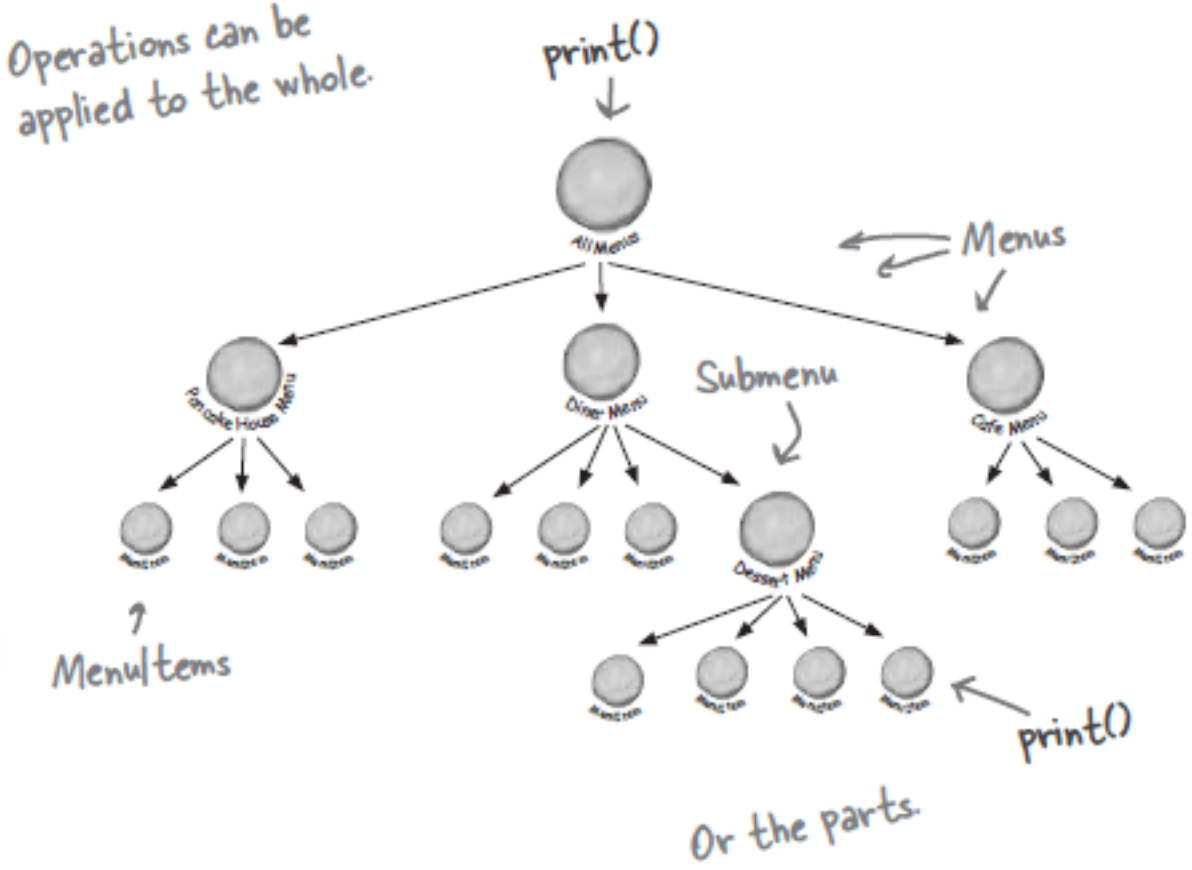
Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Composite-Muster

Composite Muster

Der hierarchische Aufbau kann dynamisch verändert werden

Operationen können auf den gesamten Baum, Teilbäume oder einzelne Blätter angewendet werden

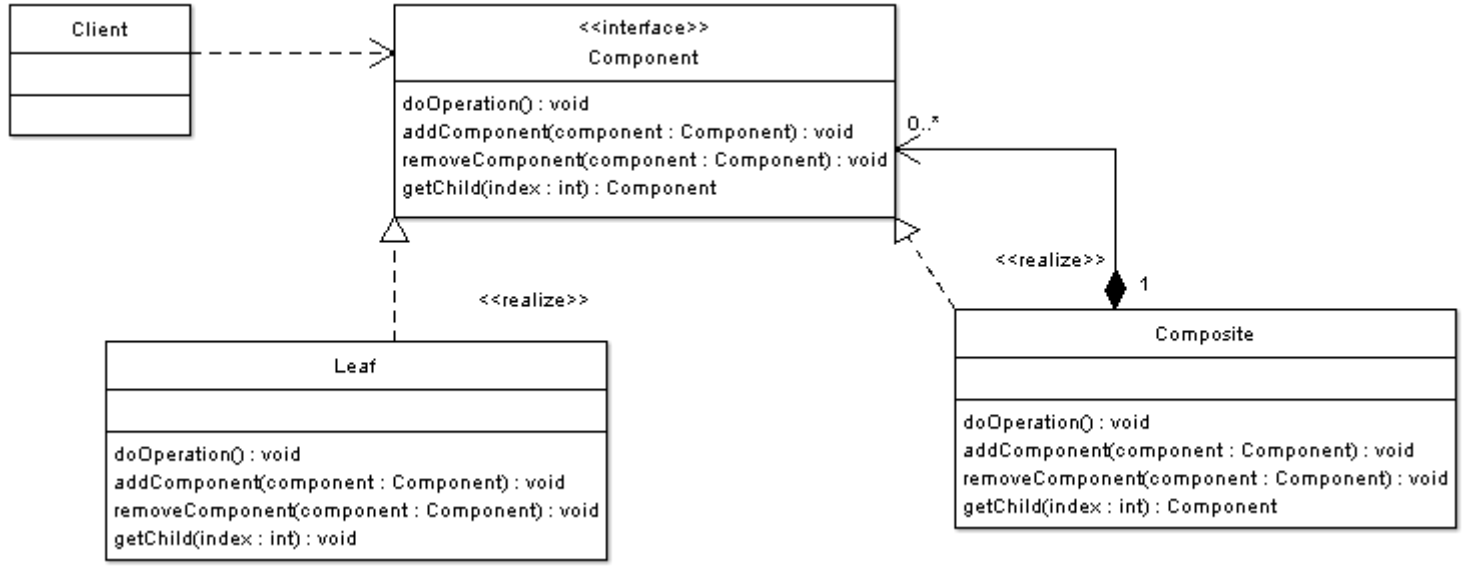


Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Composite-Muster

Composite Muster

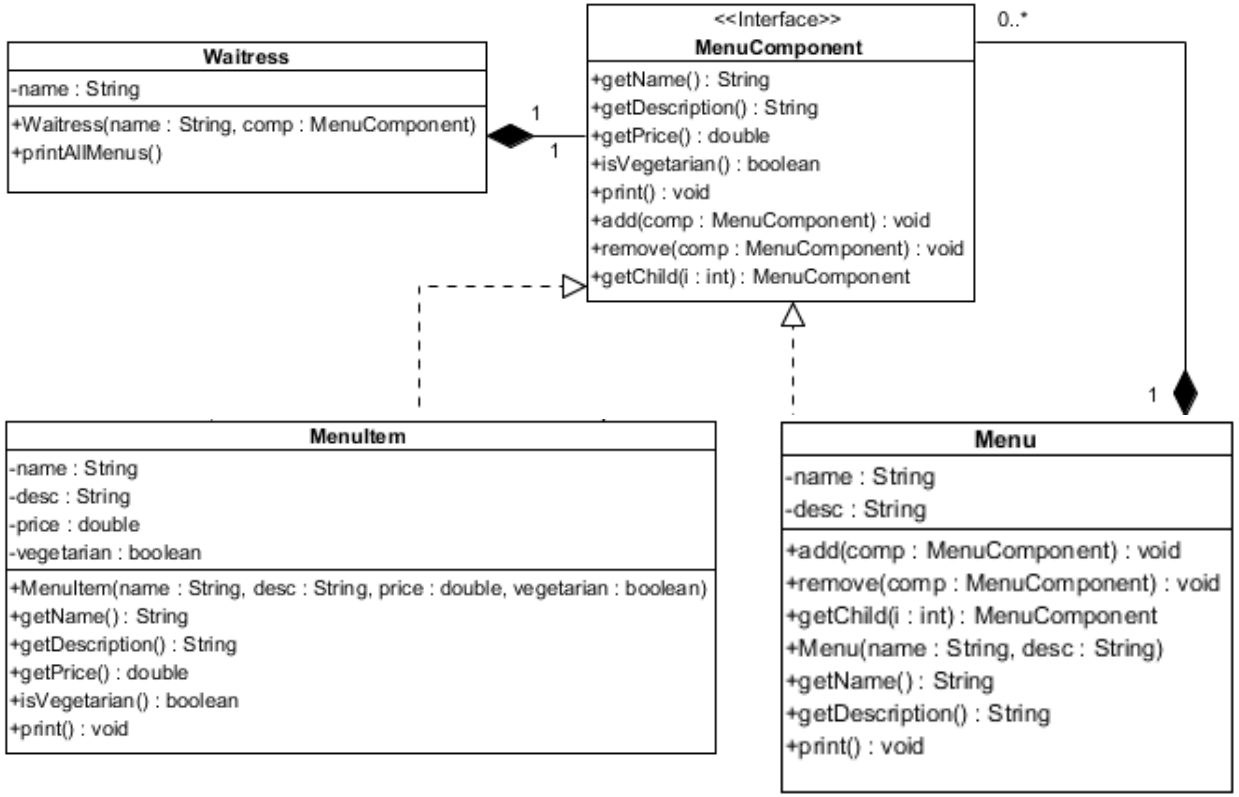
Für Knoten (Teilbäume) und Blätter wird eine gemeinsame Schnittstelle erstellt. Knoten und Blätter implementieren die verschiedenen Methoden nicht vollständig bzw. nur „unsupported“.



Composite-Muster

Re-Design – Menüverwaltung

Die Klassen `Menu` und `MenuItem` implementieren nun eine gemeinsame Schnittstelle `MenuComponent`



Warum wurden diese Beziehungstypen und Multiplizitäten gewählt?

Composite-Muster

Beispiel – Menüverwaltung

```
public interface MenuComponent {

    public void add(MenuComponent menuComponent);
    public void remove(MenuComponent menuComponent);
    public MenuComponent getChild(int i);

    public String getName();
    public String getDescription();
    public double getPrice();
    public boolean isVegetarian();
    public void print();
}
```

Composite-Muster

Beispiel – Menüverwaltung

```
public class MenuItem implements MenuComponent {

    ...

    @Override
    public void add(MenuComponent menuComponent) {
        throw new UnsupportedOperationException();
    }

    @Override
    public void remove(MenuComponent menuComponent) {
        throw new UnsupportedOperationException();
    }

    @Override
    public MenuComponent getChild(int i) {
        throw new UnsupportedOperationException();
    }

    @Override
    public String getName() {
        return name;
    }
...
}
```

Composite-Muster

Beispiel – Menüverwaltung

```
public class Menu implements MenuComponent {

    private ArrayList<MenuComponent> menuComponents = new ArrayList<MenuComponent>();
    ...
    @Override
    public void add(MenuComponent menuComponent) {
        menuComponents.add(menuComponent);
    }
    @Override
    public MenuComponent getChild(int i) {
        return menuComponents.get(i);
    }
    @Override
    public double getPrice() {
        throw new UnsupportedOperationException();
    }
    @Override
    public void print() {
        System.out.print("\n" + getName());
        System.out.println(", " + getDescription());
        System.out.println("---------------------");
        for (MenuComponent menuComponent : menuComponents) {
            menuComponent.print();
        }
    }
}
```

Composite-Muster

Beispiel – Menüverwaltung

```
public class Waitress {

    private MenuComponent allMenus;
    private String name;

    public Waitress(String name, MenuComponent allMenus) {
        this.name = name;
        this.allMenus = allMenus;
    }

    public String getName() {
        return name;
    }

    public void printAllMenus() {
        System.out.print("\n" + getName());
        System.out.println("---------------------");
        allMenus.print();
    }
}
```

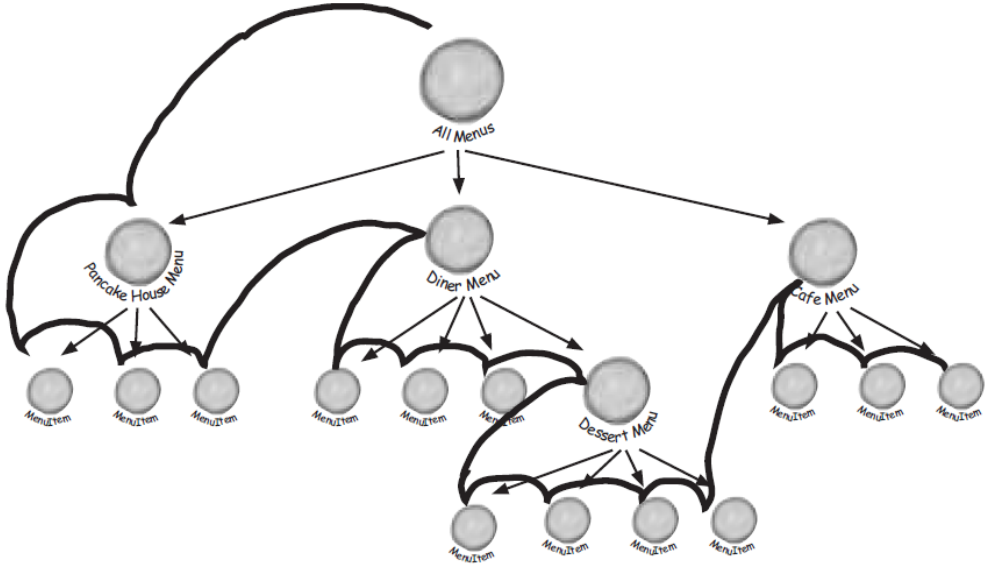
Eigene Iteratoren

Häufig sollen hierarchische Strukturen in unterschiedlicher Art und Weise durchlaufen werden. Beispielsweise sollen alle vegetarischen Gerichte ausgegeben werden. Damit eine solche Filterung vorgenommen werden kann, müssen alle Einträge durchlaufen werden. Hierzu werden in der Regel Iteratoren verwendet.

Composite-Muster

Composite-Iteratoren

Bei den sogenannten Composite-Iteratoren werden alle untergeordneten Knoten und Blätter durchlaufen. Der Durchlauf wird bei der Wurzel gestartet. Es werden alle untergeordneten Elemente von links nach rechts durchlaufen. Falls ein Element ein Knoten (Teilbaum) ist, wird in die nächste Ebene gewechselt.

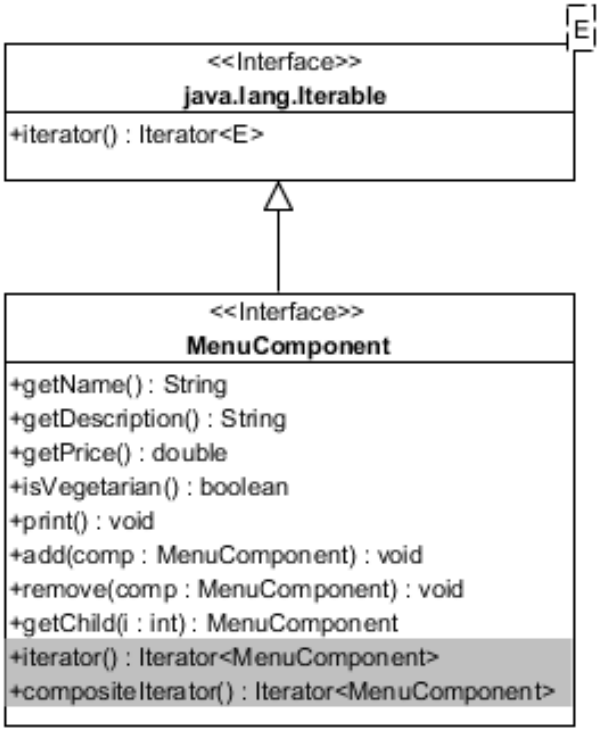


Composite-Muster

Composite Muster und Iteratoren

Bei der Verwendung des Composite Musters sollte ein Methode zum Abfragen eines Iterators vorgesehen werden. Zwei Iteratoren sind dabei sinnvoll:

- Ein Iterator, der nur die direkten Kinder durchläuft. Hier sollte am Besten der Iterator der Knotenliste zurückgegeben werden.
- Ein Iterator, der alle direkten und untergeordneten Kinder durchläuft. Hier muss ein eigener Iterator implementiert werden.



Composite-Muster

Null-Iteratoren

Sogenannte Null-Iteratoren werden verwendet, wenn ein Objekt keinen Durchlauf zur Verfügung stellen kann, jedoch eine entsprechende Methode besitzt. Beim Composite-Muster können dadurch Knoten und Blätter einheitlich verwendet werden.

```
    public class NullIterator implements Iterator<MenuComponent> {

    @Override
    public boolean hasNext() {
        return false;
    }

    @Override
    public MenuComponent next() {
        return null;
    }

    @Override
    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```

Composite-Muster

Beispiel – Menüverwaltung

```
public class MenuItem implements MenuComponent {

    ...

    @Override
    public Iterator<MenuComponent> iterator() {
        return new NullIterator();
    }

    @Override
    public Iterator<MenuComponent> compositeIterator() {
        return new NullIterator();
    }
}
```

Composite-Muster

Beispiel – Menüverwaltung

```
public class Menu implements MenuComponent {

    private ArrayList<MenuComponent> menuComponents = new ArrayList<MenuComponent>();

    ...

    @Override
    public Iterator<MenuComponent> iterator() {
        return menuComponents.iterator();
    }

    @Override
    public Iterator<MenuComponent> compositeIterator() {
        return new CompositeIterator(this);
    }
}
```

Composite-Muster

Beispiel – Menüverwaltung

```
public class CompositeIterator implements Iterator<MenuComponent> {

    Stack<MenuComponent> toVisit = new Stack<MenuComponent>();

    public CompositeIterator(MenuComponent menuComponent) {
        toVisit.push(menuComponent);
    }

    public MenuComponent next() {
        if (hasNext()) {
            MenuComponent menuComponent = toVisit.pop();
            int index = 0;
            for (MenuComponent tmp : menuComponent) {
                toVisit.insertElementAt(tmp, index);
                index++;
            }
            return menuComponent;
        } else {
            return null;
        }
    }

    public boolean hasNext() {
        return !toVisit.isEmpty();
    }
    ...

    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```

Composite-Muster

Beispiel – Menüverwaltung

```
public class CompositeIterator implements Iterator<MenuComponent> {

    LinkedList<MenuComponent> toVisit = new LinkedList<MenuComponent>();

    public CompositeIterator(MenuComponent menuComponent) {
        toVisit.add(menuComponent);
    }

    public MenuComponent next() {
        if (hasNext()) {
            MenuComponent menuComponent = toVisit.poll();
            for (MenuComponent tmp : menuComponent) {
                toVisit.add(tmp);
            }
            return menuComponent;
        } else {
            return null;
        }
    }

    public boolean hasNext() {
        return !toVisit.isEmpty();
    }
    ...
                                        public void remove() {
                                            throw new UnsupportedOperationException();
                                        }
                                    }
```

Composite-Muster

Fazit

Das Composite Muster bietet an Struktur, um einzelne und gruppierte Objekte hierarchisch verwalten zu können.

Für den Anwender (Client) ist es nicht ersichtlich, ob ein einzelnes oder gruppiertes Objekt verwendet wird. Es entsteht somit eine einheitliche Verwendung.

Bei der Implementierung müssen jedoch Aspekte der Konsistenz, Transparenz und Sicherheit beachtet werden.

Iteratoren können zum Durchlauf von Composite-Strukturen verwendet werden.