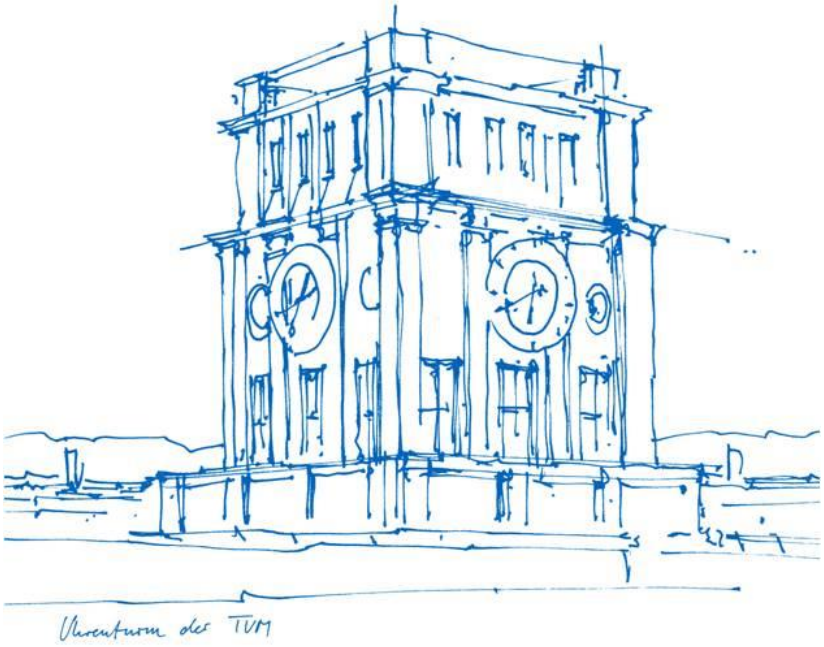
PGdP Woche 8

Daniel Krüger, Sebastian Oßner

Technische Universität München

09.12.2019



Folien Zusammenfassung

1. Kurze Wiederholung zu Abstract Classes
2. Keyword *final*
3. Interfaces
4. Generics
5. P01 - Polymorphie
    1. Statischer / Dynamischer Typ
    2. Methodensignatur
    3. Überschreiben
    4. Überladen
    5. Dynamischer Dispatch

Kurze Wiederholung von abstract class

- Kann nicht instanziiert werden
- Hat Attribute, Konstruktor und Methoden (Wie eine normale Klasse)
- Hat zusätzlich abstrakte Methoden: werden aber nur in den Unterklassen implementiert

```
public abstract class Animal {
    private int age;

    public Animal(int age) {
        this.age = age;
    }

    public abstract void eat(Animal food);
}
```

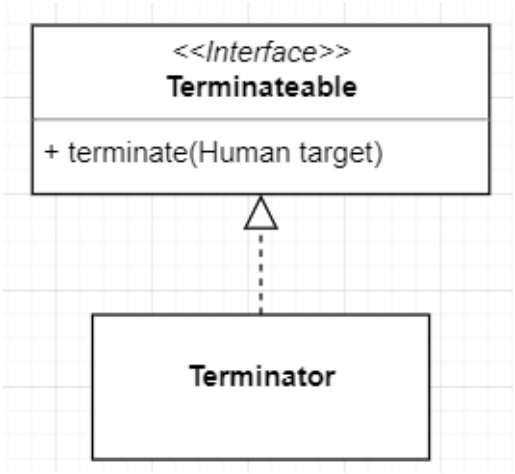
Keyword *final*

- final Klassen: können keine Unterklassen haben
- final Methode: dürfen in Unterklassen nicht geändert (overwritten) werden
- Final Attribute und Variablen: dürfen einmal zugewiesen werden und danach nicht mehr verändert werden

    Zum Beispiel Integer.MAX_VALUE (all caps, nicht veränderbar)

Interfaces

Wir wissen, wie abstract Methoden in abstract classes funktionieren. Ein Interface ist sehr ähnlich.
Eine Klasse kann nur von **einer** Klasse erben. Aber sie kann beliebig viele Interfaces implementieren.



Erinnerung: obwohl es terminatable heißt, bedeutet es nicht, dass Terminator terminated wird, Sondern dass er terminaten **kann**

Interfaces

Interface:

```
public interface Cookable {
    public void boil();
    public void fry();
    public void steam();
}
```

Alle Klassen, die Cookable implementieren, können gekocht, fritiert oder gedampft werden. Klassen, die von Interfaces implementieren, **müssen** diese Methoden haben

Interfaces

```java
public class Stew implements Cookable {
    @Override
    public void boil() {
        System.out.println("Now boiling stew...");
    }

    @Override
    public void fry() {
        System.out.println("Now frying stew...");
    }

    @Override
    public void steam() {
        System.out.println("Now steaming stew...");
    }
}
```

Stew implementiert Cookable. Stew kann jetzt gekocht, fritiert oder gedampft werden.

(Es wird in späteren Vorlesungen dann Iteratables, Comparables, usw. Das Prinzip funktioniert sehr ähnlich wie Cookable)

Generics

Sinn: Wenn eine Klasse für viele verschiedene Datentypen auch funktioniert soll, benutzt man Generics. Vorteil davon ist, dass man die Klasse nur einmal schreiben und nicht mehrmals (Liste die dann viele Datentypen speichern kann, statt einer int-Liste, double-Liste, String-Liste)

Der Header sieht dann so aus:

```
public class ListOfObjects<T> {
```

T ist unser Generic

Generics

Wir behandeln unsere Generic-typ einfach wie ein Datentyp

```
public class ListOfObjects<T> {
    private T info;
    private ListOfObjects<T> next;

    public ListOfObjects(T info, ListOfObjects<T> next) {
        this.info = info;
        this.next = next;
    }

    public void insert(T info) {
        next = new ListOfObjects<T>(info, next);
    }
}
```

Generics

Was wenn ich meine Generics etwas einschränken will?
Man kann Generics auf **eine Klasse** und **alle Unterklassen** einschränken

```
public class Terminator<T extends Animal>{
    public void terminate(T target) {
        System.out.println("Hasta la vista, baby");
        target.setAge(-1);
    }
}
```

Wenn ein Terminator alle Tiere töten kann, dann kann er Menschen, Säugetiere, Vögel und alles, was von Animal erbt töten. Dafür kann er nicht diese Methode auf Integers oder Strings ausüben.

Polymorphie - Terminologie

Statischer Typ: wird bei Compile-Zeit festgelegt, nicht veränderbar
Dynamischer Typ: wird zur Laufzeit festgelegt, kann sich während Laufzeit ändern

Methodensignatur: Definition der Methode - Name und Parametertypen

Überschreiben: Überschreiben einer definierten Methode in einer Unterklasse

Überladen: Methode mit dem selben Namen, aber einer anderen Signatur

Dynamic Dispatch: Determinierung, welche Methode aus mehreren möglichen ausgewählt wird aufgrund der Typen der Variablen

P-Aufgabe 1: Polymorphie

Betrachte einen Aufruf $e_0.f(e_1, \ldots, e_k)$.

Bestimme die statischen Typen $T_0, \ldots, T_k$ der Ausdrücke $e_0, \ldots, e_k$.

Suche in einer Oberklasse von T_0 nach einer Methode mit Namen f, deren Liste von Argumenttypen bestmöglich zu der Liste $T_1, \ldots, T_k$ passt.

Der Typ I dieser rein statisch gefundenen Methode ist die Signatur der Methode f an dieser Aufrufstelle im Programm.

Der dynamische Typ D des Objekts, zu dem sich e_0 auswertet, gehört zu einer Unterklasse von T_0.

Die Methode f wird nun aufgerufen, deren Typ I ist und die in der nächsten Oberklasse von D implementiert wird.

Betrachte einen Aufruf $e_0.f(e_1, \ldots, e_k)$.

Bestimme die statischen Typen $T_0, \ldots, T_k$ der Ausdrücke $e_0, \ldots, e_k$.

Suche in einer Oberklasse von T_0 nach einer Methode mit Namen f, deren Liste von Argumenttypen bestmöglich zu der Liste $T_1, \ldots, T_k$ passt.

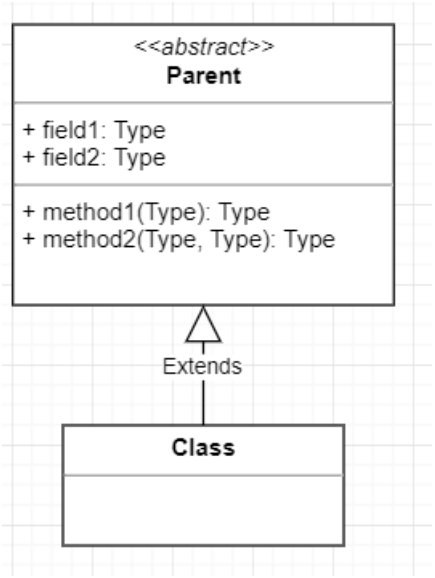
Der Typ I dieser rein statisch gefundenen Methode ist die Signatur der Methode f an dieser Aufrufstelle im Programm.

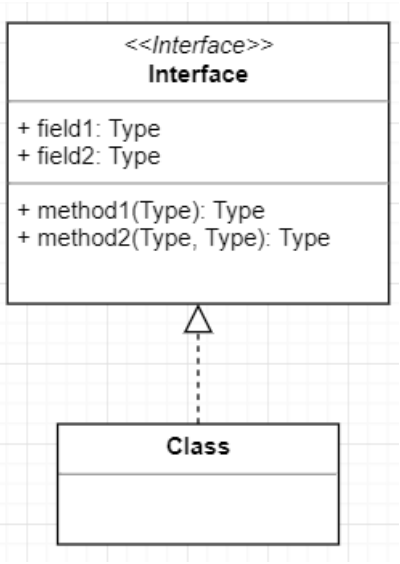
Der dynamische Typ D des Objekts, zu dem sich e_0 auswertet, gehört zu einer Unterklasse von T_0.

Die Methode f wird nun aufgerufen, deren Typ I ist und die in der nächsten Oberklasse von D implementiert wird.

Auf-ruf	Zei-le	stat. Typ	kompati-ble Methoden	statisch gewählte Methode	Si-gna-tur	Begründung	dyn. Typ	zur Lauf-zeit aus-geführt	Begründung	Ausgabe
1	62	B	41, 14, 36, 9	41	f(B)	Speziellste Signatur, niedrigste Unterklasse	B	41	static, kein Dispatch	9-

P-Aufgabe 2: Fabelwesen





P-Aufgabe 3: Bäume

Zu dieser Aufgabe, erstelle ich eine Lösung mit Erklärung. Siehe dann W8P3Explained.pdf