

FOP Recap #5



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Interfaces mit Senf





Hej! Wir sind nicht Marc!



Hej! Wir sind nicht Marc!

Wir möchten trotzdem mit euch über die FOP sprechen!

Das steht heute auf dem Plan



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Methoden

- Überladen von Methoden

- Überladen von Konstruktoren
- super in Methoden

Interfaces

Live-Coding: Algomon



Eine Methode heißt *überladen*, wenn für den selben Typ¹ (mindestens) eine weitere gleichnamige Methode existiert

- keine zwei gleichnamigen Methoden mit selben Parametern in selber Reihenfolge
- aufrufbar wie „normale“ Methoden
- es wird immer die Methode gewählt, deren formale Parameter die aktuellen Parameter am besten darstellen

Überladen \neq Überschreiben

¹Klasse, Interface, ...

Methoden

Überladen von Methoden — Beispiel Klasse MyPrinter



```
1 public void print(String message) {  
2     System.out.println("string: " + message);  
3 }  
4  
5 public void print(Object object) {  
6     System.out.println("object: " + object);  
7 }  
8  
9 public void print(int number) {  
10     System.out.println("number: " + number);  
11 }
```



Was liefern folgende Aufrufe?

```
1 MyPrinter printer = new MyPrinter();  
2 printer.print(1337);  
3 printer.print("Hello Darmstadt!");  
4 printer.print(4.2);
```



Was liefern folgende Aufrufe?

```
1 MyPrinter printer = new MyPrinter();  
2 printer.print(1337);  
3 printer.print("Hello Darmstadt!");  
4 printer.print(4.2);
```

number: 1337

string: Hello Darmstadt!

object: 4.2



Konstruktoren sind nur *spezielle* Methoden

- beim Überladen gleiche Eigenschaften wie bei „normalen“ Methoden



```
1 public MyRobot(int x, int y, int numberOfCoins) {  
2     this.x = x;  
3     this.y = y;  
4     this.numberOfCoins = numberOfCoins;  
5 }  
6  
7 public MyRobot(int x, int y) {  
8     this.x = x;  
9     this.y = y;  
10    this.numberOfCoins = 0;  
11 }  
12  
13 public MyRobot() {  
14     this(0, 0); // calls constructor with two parameters  
15 }
```



super bereits von Konstruktoren bekannt

- super-Aufruf ruft überschriebene Methode aus Basis-Klasse auf

Syntax

```
super .Methodenname ( aktuale Parameter ) ;
```

Methoden

super in Methoden



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1  class A {  
2  
3      void a() {  
4          println("A.a()");  
5      }  
6  
7      void b() {  
8          println("A.b()");  
9      }  
10 }
```

```
1  class B extends A {  
2  
3      @Override  
4      void a() {  
5          super.a();  
6          println("B.a()");  
7      }  
8  
9      @Override  
10     void b() {  
11         println("B.b()");  
12     }  
13 }
```



Was liefern folgende Aufrufe?

```
1 B b = new B();  
2 b.a(); // B overrides a with super  
3 b.b(); // B overrides b without super
```



Was liefern folgende Aufrufe?

```
1 B b = new B();  
2 b.a(); // B overrides a with super  
3 b.b(); // B overrides b without super
```

A.a()

B.a()

B.b()

Das steht heute auf dem Plan



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Methoden

Interfaces

- implements und extends

- Deklaration

- Beispiel

- Weitere Vorteile

Live-Coding: Algomon



„An interface is a contract between a class and the outside world. When a class implements an interface, it promises to provide the behavior published by that interface.“

– Oracle

Idee

Trennung zwischen *Deklaration* und *Implementation*

- Interfaces *deklarieren* Methoden
- Klassen *implementieren* Methoden

Interfaces

Deklaration – Interface



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Syntax

<Modifiers> interface <Interface Name> extends <Parent Interfaces>

```
1 interface A {  
2     // ...  
3 }
```

```
1 interface B {  
2     // no content  
3 }
```

```
1 interface C extends A, B {  
2     // ...  
3 }
```

- Interfaces sind automatisch *immer* `public` – muss nicht angegeben werden



Syntax

```
<Modifiers> interface <Interface Name> extends <Parent Interfaces> { ... }
```

- enthalten Methoden-*Deklarationen*
- Methoden in Interfaces sind auch *immer public*
- Methoden aus abgeleiteten Interfaces (*Parent Interfaces*) müssen nicht neu deklariert werden

Interfaces

Deklaration – Methoden



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Beispiel

```
1 interface A {  
2     void doMagic(int n);  
3     int doBad();  
4 }
```

```
1 interface C extends A, B {  
2     String getBehavior();  
3 }
```

- folgende Deklarationen sind identisch:

```
1 public int doMagic(int n);  
2 int doMagic(int n);
```

Interfaces

Deklaration – default-Methoden



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Syntax

```
default <Modifiers> <Return Type> <Method Name>(<Parameters>) { ... }
```

- können *immer* in implementieren Klassen überschrieben werden



Klassiker

- Hinweis auf fehlende Implementation

```
1 default void setZ() {  
2     throwError();  
3 }
```

- Unabhängigkeit von Implementation

```
1 default int getXPlusY() {  
2     return getX() + getY();  
3 }
```



- Interface können keine *Objektattribute* haben → Implementation
- Interfaces können *nur public-final* Klassenattribute (`public + final + static`) haben
- müssen *direkt* initialisiert werden
- folgende Deklarationen + Initialisierungen sind identisch

```
1 public static final int MAGIC_NUMBER = 42;  
2 int MAGIC_NUMBER = 42;
```

Interfaces

Deklaration – Attribute



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 interface Connectable {  
2  
3     int DEFAULT_MAX_NUMBER_OF_CONNECTIONS = 42;  
4  
5     void connect();  
6  
7     default int maxNumberOfConnections() {  
8         return DEFAULT_MAX_NUMBER_OF_CONNECTIONS;  
9     }  
10 }
```



Zwei Klassen mit Position

```
1 public class Person {  
2  
3     public int getX() {  
4         return x;  
5     }  
6  
7     public int getY() {  
8         return y;  
9     }  
10    // ...  
11 }
```

```
1 public class Car {  
2  
3     public int getX() {  
4         return x;  
5     }  
6  
7     public int getY() {  
8         return y;  
9     }  
10    // ...  
11 }
```




Klasse zum Berechnen des Abstands zwischen zwei Personen

```
1 public PersonEuclidianDistanceCalculator {  
2  
3     public double calcDistance(Person p1, Person p2) {  
4         int dx = p2.getX() - p1.getX();  
5         int dy = p2.getY() - p1.getY();  
6         return Math.sqrt(dx * dx + dy * dy);  
7     }  
8     // ...  
9 }
```



Der `PersonEuclidianDistanceCalculator` berechnet zwischen zwei Personen die euklidische Distanz.

Was ist, wenn weitere Klassen (und Distanzen) unterstützt werden sollen?

- je weiterer Klasse (und weiterer Distanz) *doppelt* so viele `Calculator`-Klassen
- für Kombinationen werden es noch mehr ...

Lösung

- alle Klassen, die Position haben, implementieren gemeinsames Interface
- alle Klassen, die Distanz berechnen können, implementieren gemeinsames Interface



Interface für Klassen mit Position

```
1 interface WithPosition {  
2  
3     int getX();  
4     int getY();  
5 }
```



Interface für Distanzberechnung

```
1 interface DistanceCalculator {  
2  
3     double calcDistance(WithPosition p1, WithPosition p2);  
4 }
```



Interface WithPosition deklariert und implementiert Methode getXPlusY()

```
1 interface WithPosition {  
2  
3     int getX();  
4     int getY();  
5  
6     default int getXPlusY() {  
7         return getX() + getY();  
8     }  
9 }
```

Interfaces

Beispiel – Klasse zum Berechnen des Abstands zwischen zwei Objekten mit Position



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public class Person implements
   ↳ WithPosition {
2
3     @Override
4     public int getX() {
5         return x;
6     }
7
8     @Override
9     public int getY() {
10        return y;
11    }
12    ...
```

```
1 public class Car implements
   ↳ WithPosition {
2
3     @Override
4     public int getX() {
5         return x;
6     }
7
8     @Override
9     public int getY() {
10        return y;
11    }
12    ...
```

Interfaces

Beispiel – DistanceCalculator mit Interfaces



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public EuclidianDistanceCalculator {  
2  
3     public double calcDistance(WithPosition p1, WithPosition p2) {  
4         int dx = p2.getX() - p1.getX();  
5         int dy = p2.getY() - p1.getY();  
6         return Math.sqrt(dx * dx + dy * dy)  
7     }
```



Reminder: Jede Klasse kann nur *eine* Klasse erweitern.

- jede Klasse kann *mehrere* Interfaces implementieren
- „Module“ (hier: *DistanceCalculator*) können an zentraler Stelle ausgetauscht werden
- Implementation können einfach ausgetauscht werden

Das steht heute auf dem Plan



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Methoden

Interfaces

Live-Coding: Algomon



Live-Coding!