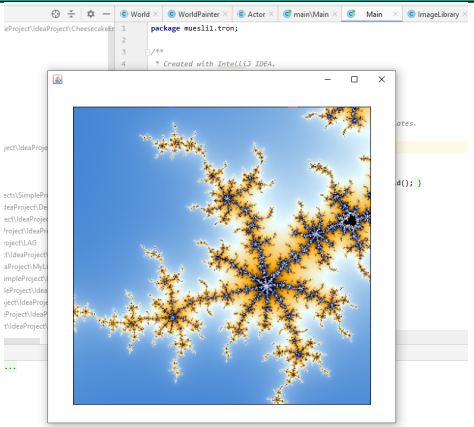
# FOP Recap #5

## Interfaces mit Senf

TECHNISCHE UNIVERSITÄT DARMSTADT



TECHNISCHE
UNIVERSITÄT
DARMSTADT

# 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

# Methoden

## Überladen von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

Eine Methode heißt *überladen*, wenn für den selben Typ[1] (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 aktualen Parameter am besten darstellen

> Überladen $\neq$ Überschreiben

[1] Klasse, Interface, ...

# Methoden

## Überladen von Methoden — Beispiel Klasse `MyPrinter`

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public void print(String message) {
    System.out.println("string: " + message);
}

public void print(Object object) {
    System.out.println("object: " + object);
}

public void print(int number) {
    System.out.println("number: " + number);
}
```

# Methoden

## Überladen von Methoden — Beispiel Klasse `MyPrinter`

TECHNISCHE UNIVERSITÄT DARMSTADT

**Was liefern folgende Aufrufe?**

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

# Methoden

## Überladen von Methoden — Beispiel Klasse `MyPrinter`

TECHNISCHE UNIVERSITÄT DARMSTADT

**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

# Methoden

## Überladen von Konstruktoren — Beispiel Klasse `MyRobot`

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public MyRobot(int x, int y, int numberOfCoins) {
    this.x = x;
    this.y = y;
    this.numberOfCoins = numberOfCoins;
}

public MyRobot(int x, int y) {
    this.x = x;
    this.y = y;
    this.numberOfCoins = 0;
}

public MyRobot() {
    this(0, 0); // calls constructor with two parameters
}
```

`super` bereits von Konstruktoren bekannt

- `super`-Aufruf ruft überschiebene Methode aus Basis-Klasse auf

**Syntax**

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

# Methoden

## super in Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
class A {

    void a() {
        println("A.a()");
    }

    void b() {
        println("A.b()");
    }
}
```

```java
class B extends A {

    @Override
    void a() {
        super.a();
        println("B.a()");
    }

    @Override
    void b() {
        println("B.b()");
    }
}
```

TECHNISCHE UNIVERSITÄT DARMSTADT

**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
```

TECHNISCHE UNIVERSITÄT DARMSTADT

**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

# Interfaces

TECHNISCHE UNIVERSITÄT DARMSTADT

> „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>*

```java
interface A {
  // ...
}
```

```java
interface B {
    // no content
}
```

```java
interface C extends A, B {
  // ...
}
```

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

# Interfaces

## Deklaration — Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

**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

TECHNISCHE UNIVERSITÄT DARMSTADT

### Beispiel

```java
interface A {
    void doMagic(int n);
    int doBad();
}
```

```java
interface C extends A, B {
    String getBehavior();
}
```

- folgende Deklarationen sind identisch:

```java
public int doMagic(int n);
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

TECHNISCHE UNIVERSITÄT DARMSTADT

### Klassiker

- Hinweis auf fehlende Implementation

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

- Unabhängigkeit von Implementation

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

# Interfaces

## Deklaration — Attribute

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

```java
interface Connectable {

    int DEFAULT_MAX_NUMBER_OF_CONNECTIONS = 42;

    void connect();

    default int maxNumberOfConnections() {
        return DEFAULT_MAX_NUMBER_OF_CONNECTIONS;
    }
}
```

TECHNISCHE UNIVERSITÄT DARMSTADT

### Zwei Klassen mit Position

```java
public class Person {

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }
    // ...
}
```

```java
public class Car {

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }
    // ...
}
```

TECHNISCHE UNIVERSITÄT DARMSTADT

### 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  }
```

# Interfaces
## Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

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

TECHNISCHE UNIVERSITÄT DARMSTADT

### Interface für Klassen mit Position

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

TECHNISCHE UNIVERSITÄT DARMSTADT

### Interface für Distanzberechnung

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

TECHNISCHE UNIVERSITÄT DARMSTADT

### 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

```java
public class Person implements
    ↪ WithPosition {

    @Override
    public int getX() {
        return x;
    }

    @Override
    public int getY() {
        return y;
    }
    ...
```

```java
public class Car implements
    ↪ WithPosition {

    @Override
    public int getX() {
        return x;
    }

    @Override
    public int getY() {
        return y;
    }
    ...
```

# 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      }
```

# Interfaces
## Weitere Vorteile

TECHNISCHE UNIVERSITÄT DARMSTADT

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: Algomon

TECHNISCHE UNIVERSITÄT DARMSTADT

# Live-Coding!