Karlsruher Institut für Technologie

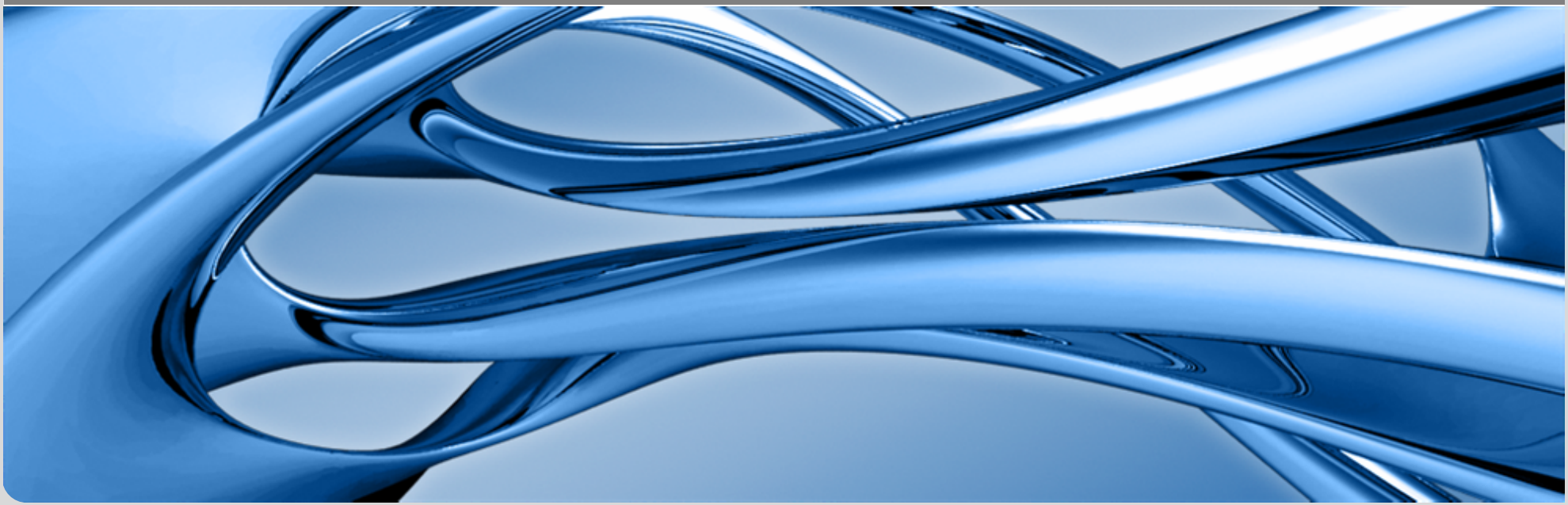
SDQ Software Design and Quality

Tutorium 4

Konstruktoren und Methoden

Christian Zielke | 20. November 2018

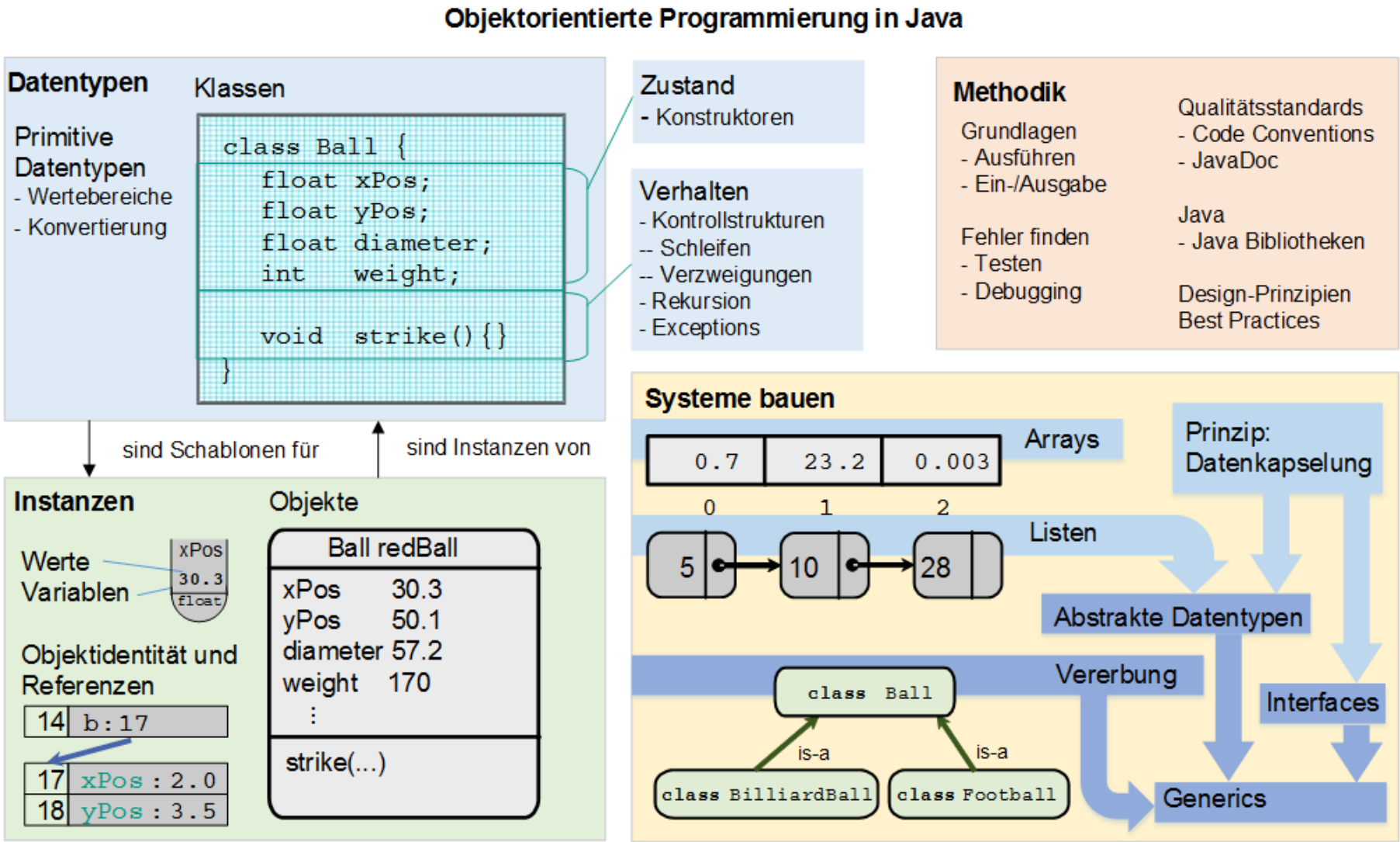
CHAIR FOR SOFTWARE DESIGN AND QUALITY



Gliederung

Gliederung

Wiederholung



Warum Konstruktoren? - Grundlagen

Bisher:

```
1   class Circle {
2     int radius;
3   }
```

```
1   Circle circle1 = new Circle();
2   circle1.radius = 5;
```

- Objekterstellung und Initialisierung der Attribute meist gleichzeitig
- Deshalb: Verwendung von Konstruktoren, die beides erledigen
- Vermeidet Vergessen der Initialisierung
- Ermöglicht Überprüfen des Werts auf Gültigkeit

Syntax

- ```
  Klassenname(param1,..., paramN) {
     //Konstruktorrumpf
  }
  ```
- Eine Methode ohne Rückgabetyp
  - keine return Anweisung
- Es kann mehrere Konstruktoren geben, die sich in den Parametern unterscheiden
  ```

this-Referenz

- Zur Vereinfachung verwendet man gerne für Parameter den gleichen Namen wie für das zugehörige Attribut
- Problem:

```
1     class Point {
2       int x;
3       int y;
4
5       Point(int x; int y) {
6         x = x; //Keine Zuweisung an Attribut
7         y = y;
8       }
9     }
```

this-Referenz

- Lösung: this-Referenz

```
1       class Point {
2         int x;
3         int y;
4
5         Point(int x; int y) {
6           this.x = x;
7           this.y = y;
8         }
9       }
```

- `this` ermöglicht Unterscheidung zwischen Parameter und Attribut

Beispiele und Verwendung

- Konstruktor ohne Parameter: `Point() {}`
  - Default-Konstruktor
  - Wird automatisch eingefügt, wenn kein Konstruktor definiert ist
  - Sobald ein Konstruktor definiert wird, exisitiert der Default-Konstruktor nicht mehr!

```
1     Point(int x; int y) {
2       this.x = x;
3       this.y = y;
4     }
```

- Verwendung: `Point q = new Point(5, 4);`
- Es können auch mehrere Konstruktoren definiert werden
  - z.B. ein Konstruktor für Eingabe in Polarkoordinaten

Default-Konstruktor Attributwerte

Typ	Default-Wert
`boolean`	`false`
`byte, short, int`	`0`
`long`	`0L`
`float`	`0.0f`
`double`	`0.0`
`char`	`'\u0000'`
Objekt-Referenz	`null`

Methoden - Grundlagen

- Verhindern Code Wiederholungen
- Geben einem Objekt “Fähigkeiten” und Dynamik
- Führen Berechnungen, Algorithmen etc. aus
- Beispiel:

```
1    int calculateArea(int a, int b) {
2      return a * b;
3    }
```

Syntax

- `Rückgabetyp MethodenName(param1,..., paramN) {`
  `//Methodenrumpf`
  `}`
- Parameter-Syntax: <Datentyp> ParameterName
- Am Ende jedes Pfads durch die Methode muss eine return Anweisung stehen
  - z.B. am Ende des Methodenrumpfs
- Diese gibt einen Wert passend zum Rückgabetyp der Methode zurück
- Spezialfall: Rückgabetyp `void`
  - Hier gibt es keinen Rückgabetyp $\rightarrow$ kein return notwendig

Signatur

- Die Signatur macht eine Methode einzigartig
- Es kann immer nur eine Methode mit einer spezifischen Signatur geben
- Sie besteht aus:
  - Name der Methode
  - Anzahl der Parameter
  - Typen der Parameter
  - Reihenfolge der Parameter
  - Rückgabetyp der Methode

Überladen

- Die Signaturverwaltung ermöglicht es, mehrere Methoden mit dem gleichem Namen zu definieren
- Dies ermöglicht **Überladen**:

```
1       int calculateArea(int a, int b) {
2         return a * b;
3       }
4
5       //Ueberladung
6       double calculateArea(double a, double b) {
7         return a * b;
8       }
```

Statische Methoden

- Ist eine Methode unabhängig vom Zustand des Objekts, kann sie **statisch** gemacht werden
- Statisch: Unabhängig von Objekten, nur Klassenabhängigkeit
- Syntax:
  `static Rückgabetyp name(param1,...,paramN) {}`
- Haben keinen Zugriff auf Attribute
- Verwendung von `this` nicht möglich

Statische Methoden

- Beispiel:

```
1    class WeatherStation {
2      static int convertToFahrenheit(int celsius) {
3        return (celsius * 9) / 5 + 32;
4      }
5    }
```

- Zugriff per `Klassenname.statischeMethode();`

```
1    WeatherStation.convertToFahrenheit(30);
```

Exkurs: Statische Attribute

- Selbiges wie für statische Methoden gilt auch für statische Attribute
- Syntax: `static Datentyp attributName;`
- Wie bei Methoden auch über
  `Klassenname.statischesAttribut;`

Exkurs: Statische Attribute

Sinnvolle Anwendungsmöglichkeit ist der mehrfache Gebrauch einer Klassenkonstante:

```
1   class SenselessCountPrinting {
2   //Was private bedeutet lernen wir noch
3   private static final String NEXT_NUMBER =
4   "The next number in order is ";
5
6   public static void main(String[] args) {
7   System.out.println(NEXT_NUMBER + 1);
8   System.out.println(NEXT_NUMBER + 2);
9   System.out.println(NEXT_NUMBER + 3);
10
11  /* Ausgabe
12  The next number in order is 1
13  The next number in order is 2
14  The next number in order is 3 */
15  }
16  }
```

Die main-Methode

- Wird gebraucht, um eine Klasse ausführbar zu machen
- Höchstens eine main-Methode pro Klasse

Analyse:

```
1   public static void main(String[] args) {}
```

- `public`: Ist nach außen sichtbar
- `static`: Unabhängig von Objekten
- `void`: Kein Rückgabetyp
- `main`: Name der Methode
- `String[] args`: Von der Konsole übergebene Parameter

Lokale Variablen

- Werden für Zwischenwerte in Berechnungen verwendet
- Existieren nur für die Dauer der Methode
- Verbessert die Übersichtlichkeit
- Beispiel:

```
1    int calculateVolume(int length, int width, int height) {
2      int area = length * width;
3      return area * height;
4    }
```

Lokale Variablen vs. Attribute

	Lokale Variable	Attribut
Deklaration	innerhalb von Methoden	außerhalb von Methoden
Lebensdauer	Methoden-Aufruf	Lebensdauer des zugehörgien Objekts
Zugänglichkeit	nur innerhalb einer Methode	für alle Methoden der Klasse
Zweck	Zwischenspeicher für Werte	Zustand des Objekts

Aufgabe: Konstruktoren

- Füge zu den vorgegebenen Klassen einen oder mehrere Konstruktoren hinzu.
- Achte darauf, dass nur sinnvolle Werte übergeben werden können.

Aufgabe: get und set Methoden

- Füge zu den vorgegebenen Klassen get und set Methoden hinzu.
- Achte darauf, dass nur sinnvolle Werte übergeben werden können.

Aufgabe: Fußballspiel