

3. Tutorium

Enums , Strings, Konstruktoren, Methoden, Code Stil

Tutorium 14

Péter Bohner | 16.11.2022



Inhaltsverzeichnis

1. Wiederholung

2. Datentypen III

3. Strings

3.1 Vergleiche

4. Konstruktoren

5. Methoden

6. Sauber Programmieren

Wiederholung
○

Datentypen III
○○

Strings
○○○○○
○○

Konstruktoren
○○○○○

Methoden
○○○○○○○○○○○○○

Sauber Programmieren
○○○

Ende
○

- Falls ihr SSH für die Abgaben nutzen wollt, müsst ihr bei Artemis einen Passwort-Reset machen.
- <https://artemis.praktomat.cs.kit.edu/account/reset/request>

Wiederholung
○

Datentypen III
○○

Strings
○○○○
○○

Konstruktoren
○○○○

Methoden
oooooooooooo

Sauber Programmieren
○○

Ende
○

Wiederholung

Mit welchem Operator berechnet man den Rest der Ganzzahldivision?

Modulo: % Beispiel: 5 % 3 ergibt 2

y = ++x entspricht...

x = x + 1; y = x;

Welches Schlüsselwort steht für eine Referenz auf "kein Objekt"?

null

Was passiert bei folgender Code-Ausführung: Rectangle rec = null; rec.a = 20f;?

NullPointerException

Aufzählungs-Datentyp

Eine Liste der möglichen Werte des Typs wird explizit angegeben Schema: **enum** Name { Liste der Werte }

Beispiele

```
enum CompassDirection { NORTH, EAST, SOUTH, WEST}
```

Color.java

```
enum Color {
    RED,
    GREEN,
    BLUE
}
```

Enums können in eine eigene Java-Datei oder innerhalb einer Klasse definiert werden.

Operationen: Müssen vom Benutzer angegeben werden analog zu Klassenmethoden

Wiederholung
○

Datentypen III
●○

Strings
○○○○○
○○

Konstruktoren
○○○○○

Methoden
○○○○○○○○○○○○○

Sauber Programmieren
○○○

Ende
○

Konstanten

Variablen mit festem, unveränderlichen Wert werden in Java durch das Schlüsselwort **final** gekennzeichnet:

lokale Variablen und Attribute

```
final Typ variablenName;
```

oder mit Initialisierung:

```
final Typ variablenName = Wert;
```

Klassenkonstanten

Schema: **static final** Typ NAME = Wert;

Beispiel: **static final float** GRAVITATION_EARTH = 9.80665f;

Wiederholung Zeichenkette

String

- Zusammensetzung von chars
- Ist kein primitiver Datentyp, sondern eine Klasse
- Außerdem sind ein paar Besonderheiten z.B. beim Vergleichen zu beachten
- Operation: Konkatination

Konkatenation

Strings werden mit dem + Operator konkateniert (verkettet):

```
1 String s1 = "Dies ist ";
2 String s2 = "eine Konkatenation";
3 String s3 = s1 + s2 + "!";
4 String s4 = "Dies ist + keine Konkatenation!";
5 System.out.println(s3);
6 System.out.println(s4);
```

Ausgabe:

```
Dies ist eine Konkatenation!
Dies ist + keine Konkatenation!
```


Was wird Ausgegeben?

```
class References {  
    public static void main(String[] args) {  
        String s1 = "Vorsicht mit ";  
        String s2 = s1;  
        s2 = s2 + "Strings";  
        System.out.println("s1: " + s1);  
        System.out.println("s2: " + s2);  
    }  
}
```

```
$ java References  
s1: Vorsicht mit  
s2: Vorsicht mit Strings
```

Warum ist das so?

Konkatenation mit +

- Bei der Nutzung von + wird ein neues Objekt erstellt
- Die Referenz zu „s1“ wird dabei gelöscht
- Deshalb ändert sich nur „s2“

immutable

- String ist immutable, also unveränderbar

Was wird Ausgegeben?

```
class References2 {  
    public static void main(String[] args) {  
        String s1 = "Vorsicht mit ";  
        String s2 = s1;  
        s2.concat("Strings");  
        System.out.println("s1: " + s1);  
        System.out.println("s2: " + s2);  
    }  
}
```

```
$ java References2  
s1: Vorsicht mit  
s2: Vorsicht mit
```

String ist immutable: `.concat()` verändert String nicht, es wird ein neuer String zurückgegeben

Strings vergleichen

Vergleiche von primitiven Datentypen (==, >=, ...)

- Können direkt verglichen werden
- Es wird direkt der im Speicher hinterlegte Wert verglichen

Vergleiche von Strings

- == vergleicht nur die im Speicher hinterlegten Werte beider Variablen
- Bei Objekten werden also nur die Referenzen verglichen
- Attributwerte werden ignoriert
- Benutzung der Funktion `equals(String string)`
- Für Klasse `String` ist `equals` bereits implementiert
- Später: Nutzung von `equals` für eigene Objekte

```
class StringComparator {  
    public static void main(String[] args) {  
        String string1 = "string";  
        String string2 = new String("string");  
        System.out.println(string1 == string2);  
        System.out.println(string1.equals(string2));  
    }  
}
```

```
$ java StringComparator  
false  
true
```

Konstrukturen - Einführung

- Dienen der Initialisierung eines Objekts (Setzen von Anfangswerten der Attribute)
- legen damit den Startzustand eines Objekts fest
- Können Anfangswerte des Objekts als Parameter erhalten

Schema:

```
KlassenName(Parameter-Liste) {
    // Konstruktor-Rumpf
}
```

Parameter-Liste: Datentyp₁ parameter₁, Datentyp₂ parameter₂, ..., Datentyp_n parameter_n

Aufruf: **new** KlassenName(aktuelle Parameter);

Wiederholung
○

Datentypen III
○○

Strings
○○○○
○○

Konstrukturen
●○○○

Methoden
○○○○○○○○○○○○

Sauber Programmieren
○○○

Ende
○

Konstruktor - Beispiel

Bisher:

```
class Person {
    String name;
    int age;
}
```

```
Person person = new Person();
person.name = "Hans Peter";
person.age = 68;
```

Mit Konstruktor:

```
class Person {
    String name;
    int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

```
Person person = new Person("Hans Peter", 68);
```

this erlaubt Unterscheidung zwischen Parameter und Attribut

Mehrere Konstruktoren

- festlegen mehrer Konstruktoren möglich
- müssen unterschiedliche Parameteranzahl und/oder unterschiedliche Parametertypen haben

Ok

```
Person(String forname, String lastname) { .. }  
Person(String name, int age) { .. }  
Person(String name) { .. }
```

Nicht Ok

```
Person(String forname, int age) { .. }  
Person(String lastname, int age) { .. }
```


Mehrere Konstruktoren - Beispiel

Intuition:

```
class Person {
    String name;
    int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    Person(String name) {
        this.name = name;
        this.age = 0;
    }
}
```

Besser:

```
class Person {
    String name;
    int age;

    Person (String name, int age) {
        this.name = name;
        this.age = age;
    }

    Person (String name) {
        this(name, 0);
    }
}
```

Default-Konstruktor

- ist kein Konstruktor angegeben, wird ein leerer Konstruktor automatisch ergänzt
- `KlassenName() { }`
- dieser initialisiert Attribute auf ihre Default-Werte
- sobald ein Konstruktor angegeben wird, verschwindet der Default-Konstruktor

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

Methoden - Einführung

Methoden realisieren das dynamische Verhalten von Objekten und führen Berechnungen durch.

Schema:

```
Rückgabety methodName(Parameterliste) {
    // Methodenrumpf
}
```

Der Rückgabety `void` steht dabei für keine Rückgabe!

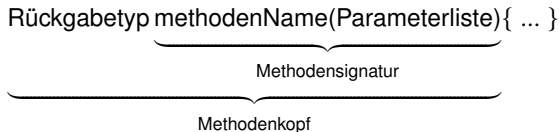
- Parameterliste: Datentyp₁ parameter₁, Datentyp₂ parameter₂, ..., Datentyp_n parameter_n

Methoden - Aufbau

Der Methodenkopf besteht aus dem Rückgabebetyp und der Methodensignatur.

Die Methodensignatur besteht aus:

- dem Namen der Methode
- der Anzahl der Parameter
- der Reihenfolge der Parameter
- der Typen der Parameter



Methoden - Zugriff auf Attribute

In Methoden kann auch auf Attribute des Objekts zugegriffen werden:

```
class Person {  
    String name = "Sven";  
  
    void greet(String name) {  
        System.out.println("Hallo " + name + ", ich heiße " + this.name + ".");  
    }  
}
```

this erlaubt Unterscheidung zwischen Parameter und Attribut

Methoden - Aufruf

- Methoden können nur auf Objekten aufgerufen werden!
 - Dazu wird die Syntax `objekt.methode()` verwendet.

Beispiel

```
Person person = new Person();  
person.greet("Peter");
```

Ausgabe:

```
Hallo Peter, ich heiße Sven.
```

Methoden - Rückgabe

Wird ein anderer Rückgabetyt als **void** gewählt, muss die Methode einen Wert vom Rückgabetyt mit dem Schlüsselwort **return** zurückgeben.

Beispiel

```
int sum(int a, int b) {  
    return a + b;  
}
```

Methoden - return bei void

return kann aber auch bei Methoden mit Rückgabetyt **void** benutzt werden:
z.B. um eine Methode gezielt zu verlassen

Beispiel

```
class Job {  
    boolean started = false;  
    void startJob() {  
        if (started)  
            return;  
        started = true;  
        // Do the Job  
    }  
}
```


Klassenmethoden

Methoden die mit dem Modifier **static** beginnen nennt man Klassenmethoden. Diese gehören zu der Klasse statt zu einem Objekt.

```
class Math {
    static int sum(int a, int b) {
        return a + b;
    }
}
```

```
int c = Math.sum(5, 8); // Nach Ausführung: c = 13
```

Es wird **kein** Objekt benötigt, um die Methode auszuführen.

Schema: Klasse.methodName();

this kann in Klassenmethoden nicht genutzt werden

Wiederholung
○

Datentypen III
○○

Strings
○○○○
○○

Konstruktoren
○○○○

Methoden
○○○○○●○○○○○

Sauber Programmieren
○○○

Ende
○

Hilfsmethoden

Was sind Hilfsmethoden?

- Methoden, die häufig verwendete Funktionalität ausführen
- Können statisch oder nicht statisch sein

statisch

- Klassenmethoden
- hängt nicht von Attributen einer Klasse ab
- `static int sum(int a, int b)`

nicht statisch

- gliedern innerhalb einer Klasse Funktionalität aus
- hängt von Attributen ab
- `int sum()` (Summe zweier Attribute)

Überladen

Es können mehrere Methoden mit dem gleichen Namen existieren, wenn sich diese in den Datentypen, Reihenfolge und/oder der Anzahl ihrer Parameter(d.h. in ihrer Signatur) unterscheiden.

Ok

```
int sum (int a, int b) { ... }
double sum (double a, double b) { ... }
int sum (int a, int b, int c) { ... }
```

Nicht Ok

```
int sum (int a, int b) { ... }
long sum (int a, int b) { ... }
int sum (int a, int c) { ... }
```

Methoden gleichen Namens, die sich nur im Rückgabetyt unterscheiden, sind nicht möglich.

Wiederholung
○

Datentypen III
○○

Strings
○○○○
○○

Konstruktoren
○○○○○

Methoden
○○○○○○○○●○○○

Sauber Programmieren
○○○

Ende
○

Methoden - Übung

Aufgabe

Schreibe eine Methode die eine Ganzzahl an nimmt und zurück gibt ob diese gerade ist.

Tipp

- 1 Mit dem Modulo Operator % erhaltet ihr den Rest der Ganzzahldivision.
- 2 Der Rückgabetyt euer Methode muss **boolean** sein.

Lösung

```
static boolean isEven(int number) {  
    return (number % 2 == 0);  
}
```

Wiederholung
○

Datentypen III
○○

Strings
○○○○○
○○

Konstrukturen
○○○○○

Methoden
○○○○○○○○○●○○

Sauber Programmieren
○○○

Ende
○

Methoden - JavaDoc

- beschreibt Zweck der Methode
- beschreibt in welchem Kontext die Methode verwendet werden kann
 - Vor- und Nachbedingungen der Ausführung
 - Spezialfälle, die das Verhalten betreffen
- alle Parameter werden beschrieben (Zweck und Wertebereich)
- der Rückgabotyp wird beschrieben (Inhalt)

Syntax

```
/**
 * Kurze aber aussagekräftige Beschreibung von Zweck und Kontext der Methode.
 * @param parameter Beschreibung des Parameters
 * @return Beschreibung Rückgabewert
 */
```

```
RückgabeDatentyp methodenName(DatenTyp parameter) {
```

Wiederholung
○

Datentypen III
○○

Strings
○○○○○
○○

Konstruktoren
○○○○○

Methoden
○○○○○○○○○○●○

Sauber Programmieren
○○○

Ende
○

Methoden - JavaDoc

```
/**
 * Returns the absolute value of an int value.
 * If the argument is not negative, the argument is returned.
 * If the argument is negative, the negation of the argument is returned.
 *
 * Note that if the argument is equal to the value of Integer.MIN_VALUE,
 * the most negative representable int value, the result is that same value, which is negative.
 *
 * @param a the argument whose absolute value is to be determined
 * @return the absolute value of the argument.
 */
static int abs(int a) {
    return (a < 0) ? -a : a;
}
```

Wiederholung
○

Datentypen III
○○

Strings
○○○○
○○

Konstruktoren
○○○○

Methoden
○○○○○○○○○○●

Sauber Programmieren
○○○

Ende
○

Sauber Programmieren

Negativ Beispiel

```
public class DemoBad{public static void main(String[]  
args){System.out.println("Hello world!");System.  
out.println("auto-generated text");}}
```

Warum sauber programmieren?

- erhöht Softwarequalität, Lesbarkeit und Wiederverwendbarkeit
- Kosten der Entwicklung und Wartung der Software werden reduziert

Wiederholung
○

Datentypen III
○○

Strings
○○○○
○○

Konstruktoren
○○○○

Methoden
○○○○○○○○○○○○

Sauber Programmieren
●○○

Ende
○

Sauber Programmieren

Aber wie?

- Namen und Bezeichner sind aussagekräftig und in Englisch (Benennungskonventionen beachten)
- Kommentare existieren und sind sinnvoll (JavaDoc)
- Einheitliche Sprache in Kommentaren (nur Englisch oder Deutsch)
- Quelltext enthält sinnvolle Einrückungen und Leerzeichen

Unterstützung

- Checkstyle!
- Automatische Überprüfung, abgaberelevant!
- Auch nach Programmieren sinnvoll, vor allem in Team Projekten

Wiederholung
○

Datentypen III
○○

Strings
○○○○
○○

Konstruktor
○○○○

Methoden
○○○○○○○○○○○○

Sauber Programmieren
●●○

Ende
○

Sauber Programmieren - Beispiel

```
/**
 * The class DemoNice looks way better!
 * @author Gregor Lucka
 * @see DemoBad
 */
public class DemoNice {
    /**
     * This main-method greets the world.
     * @params args This methods does not need any arguments.
     */
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Wiederholung
○

Datentypen III
○○

Strings
○○○○○
○○

Konstrukturen
○○○○○

Methoden
○○○○○○○○○○○○○

Sauber Programmieren
○○●

Ende
○

Bis zum nächsten Tutorium am 23.11.2022!

Wiederholung
○

Datentypen III
○○

Strings
○○○○
○○

Konstruktoren
○○○○

Methoden
○○○○○○○○○○○○

Sauber Programmieren
○○○

Ende
●