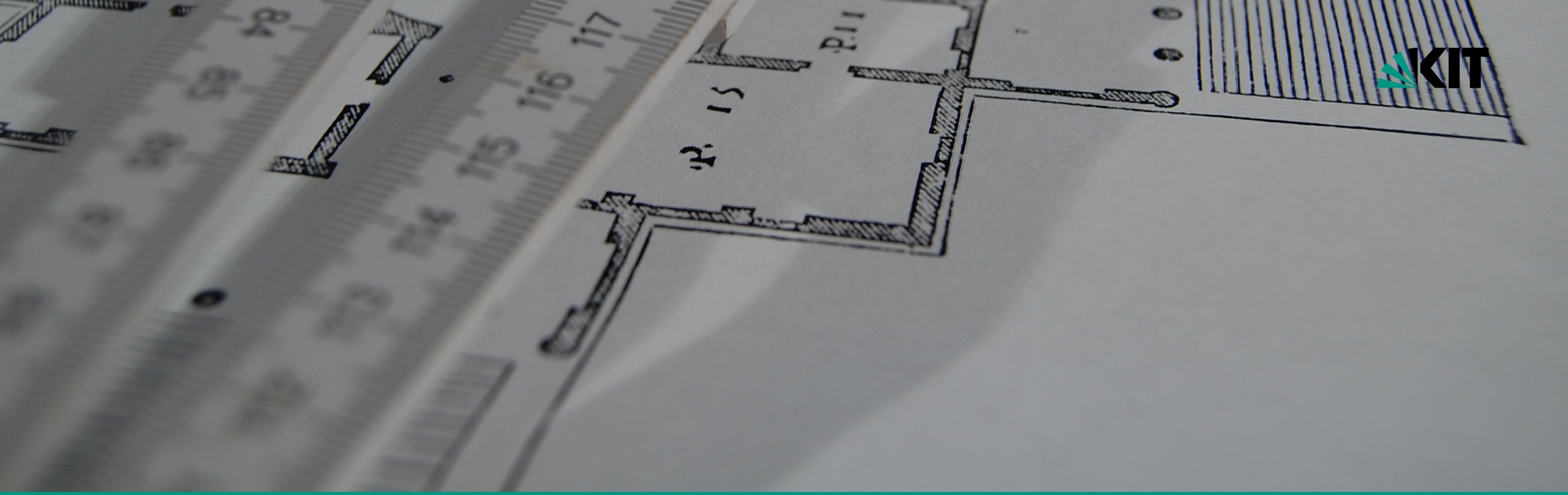


Methoden, Konstruktoren und Sichtbarkeit

Programmieren Tutorium Nr.12

Aleksandr Zakharov | 2. Dezember 2025

Organisatorisches

Übungsblatt 2 Abgabe

26.11.2025 12:00 Uhr – 04.12.2025 06:00 Uhr

Übungsschein

- Anmeldung: 01.10.2025 bis 10.12.2025, jeweils 12:00 Uhr
- Im Campussystem
- Meldet euch so bald wie möglich an und erspart euch Stress :)

Organisatorisches

Übungsblatt 2 Abgabe

26.11.2025 12:00 Uhr – 04.12.2025 06:00 Uhr

Übungsschein

- Anmeldung: 01.10.2025 bis 10.12.2025, jeweils 12:00 Uhr
- Im Campussystem
- Meldet euch so bald wie möglich an und erspart euch Stress :)

1. Übungsblatt

Anmerkungen

- Keine unnötigen Sachen ausgeben
- Aufgabenstellung genau lesen
- Nicht angeben (außer ihr seid 200% sicher, dass ihr es könnt :D)
  - Ihr dürft Sachen benutzen, die noch nicht dran kamen, aber wenn es falsch ist gibt es eben Punktabzug
  - Nicht unnötig flexen und Punkte verlieren
- Achtet auf Variablennamen! Nur Englisch und aussagekräftig!
- Keine unnötig komplexen Sachen machen.
  Wenn man etwas einfach aber gut lösen kann, ist das besser als kompliziert und gut.
- Ab dem zweiten Blatt ist der Checkstyle sehr relevant!

Wiederholung

Was bisher geschah …

Bisherige Inhalte

- Klassen
- Objekte
- Attribute
- Variablen
- Datentypen
- enum
- Methoden
- Konstruktoren
- main-Methode
- Kontrollstrukturen
  - Bedingte Ausführung
  - Schleifen

Wer weiss es?

In eigenen Worten, mit Beispiel?

Wiederholung

`static`

- Was war nochmal gleich der Unterschied zwischen mit/ohne `static`?

Wiederholung

`static`

- Was war nochmal gleich der Unterschied zwischen mit/ohne `static`?
- Wann `static` und wann nicht?

Wiederholung

`static`

- Was war nochmal gleich der Unterschied zwischen mit/ohne `static`?
- Wann `static` und wann nicht?
- Wie ist das bei Methoden?

Wiederholung

`static`

- Was war nochmal gleich der Unterschied zwischen mit/ohne `static`?
- Wann `static` und wann nicht?
- Wie ist das bei Methoden?
  `new Math().abs(-20)` oder `Math.abs(-20)`?

Wiederholung

`static`

- Was war nochmal gleich der Unterschied zwischen mit/ohne `static`?
- Wann `static` und wann nicht?
- Wie ist das bei Methoden?
  `new Math().abs(-20)` oder `Math.abs(-20)`?

Kommentare?

Wiederholung

`static`

- Was war nochmal gleich der Unterschied zwischen mit/ohne `static`?
- Wann `static` und wann nicht?
- Wie ist das bei Methoden?
  `new Math().abs(-20)` oder `Math.abs(-20)`?

Kommentare?

- Zeilenkommentar: `// Ich bin ein Kommentar`
- Blockkommentar: `/* Mehrere Zeilen möglich! */`
- Javadoc: `/** Beschreibung einer Klasse, Methode, ...*/`

Wiederholung

Kommentare - Hinweise

- Knapp aber präzise
- Keine offensichtlichen Banalitäten kommentieren, die direkt ersichtlich sind
- Zum Verständnis des Programms beitragen – nicht dieses unleserlich machen

Wiederholung

Was ist das jeweilige Ergebnis?

```
1 int  i=+5+-100000000*1000/1000-1000%17;
2
3 boolean b=5*8>=9!=6+8*2<8;
```

Wiederholung

Was ist das jeweilige Ergebnis?

```
1 int  i=+5+-100000000*1000/1000-1000%17;
2
3 boolean b=5*8>=9!=6+8*2<8;
```

Gefällts?

Schön, so ganz ohne Leerzeichen, nicht?
Würdet Ihr solchen Code korrigieren *lesen* wollen?

*(Just because you can doesn't mean you **should**.)*

Wiederholung

Was ist das jeweilige Ergebnis? *(mit Hilfe)*

```
1 int  i  = (  (+5)  + ((–100000000 * 1000) / 1000) )
2            – (1000 % 17);
3 boolean b = ((5 * 8)  >= 9)  !=  ((6  + (8  *  2))  < 8);
```

Ergebnis . . .

Wiederholung

Was ist das jeweilige Ergebnis? *(mit Hilfe)*

```
1 int  i  = (  (+5)  + ((–100000000 * 1000) / 1000) )
2            – (1000 % 17);
3 boolean b = ((5 * 8)  >= 9)  !=  ((6  + (8  *  2))  < 8);
```

Ergebnis . . .

i == -1215761 (Überlauf!)
b == true

Wiederholung

Was ist das Ergebnis für *j1, c1, j2, c2*?

```
1 int  j1  = 0;
2 boolean c1 = 5 < 8 ||  j1++ < 1 && ++j1 < 2;
3 c1 = (5  < 8)  ||  (((j1++) < 1)  && ((++j1) < 2));
4 int  j2  = 0;
5 boolean c2 = 5 < 8 | j2++ < 1 && ++j2 < 2;
6 c2 = ((5  < 8)  |  ((j2++) < 1))  && ((++j2) < 2);
```

Ergebnis . . .

Wiederholung

Was ist das Ergebnis für *j1, c1, j2, c2*?

```
1 int  j1  = 0;
2 boolean c1 = 5 < 8 ||  j1++ < 1 && ++j1 < 2;
3 c1 = (5  < 8)  ||  (((j1++) < 1)  && ((++j1) < 2));
4 int  j2  = 0;
5 boolean c2 = 5 < 8 | j2++ < 1 && ++j2 < 2;
6 c2 = ((5  < 8)  |  ((j2++) < 1))  && ((++j2) < 2);
```

Ergebnis . . .

j1 = 0
c1 = true
j2 = 4
c2 = false

WARUM?

Wiederholung

Was ist das Ergebnis für *j1, c1, j2, c2*?

```
1 int  j1  = 0;
2 boolean c1 = 5 < 8 ||  j1++ < 1 && ++j1 < 2;
3 c1 = (5  < 8)  ||  (((j1++) < 1)  && ((++j1) < 2));
4 int  j2  = 0;
5 boolean c2 = 5 < 8 | j2++ < 1 && ++j2 < 2;
6 c2 = ((5  < 8)  |  ((j2++) < 1))  && ((++j2) < 2);
```

Ergebnis . . .

j1 = 0
c1 = true
j2 = 4
c2 = false

WARUM? (Stichwort: Kurzauswertung)

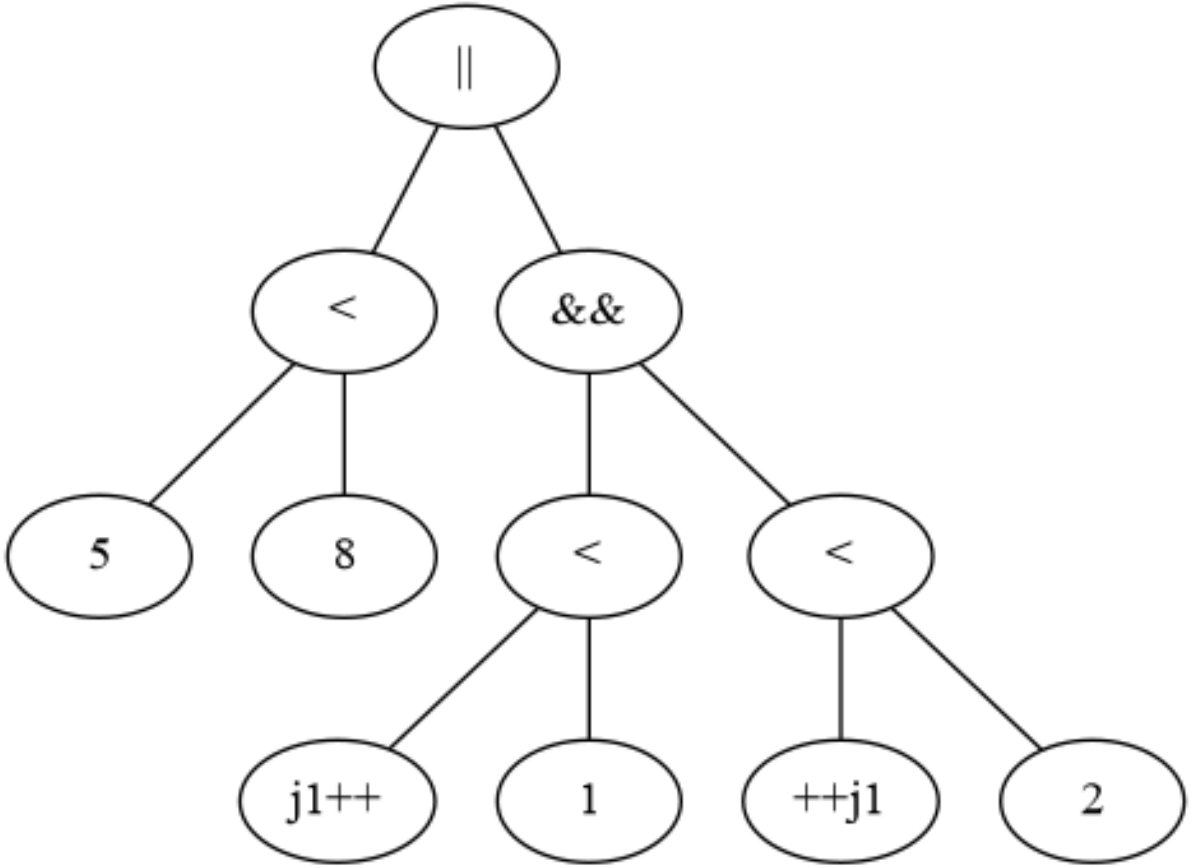
Aber warum? Was ist mit Präzedenz??

Auswertung von Unterausdrücken

Die Auswertung von Unterausdrücken ist unabhängig von der Präzedenz der einzelnen Operatoren und wird meistens von links nach rechts evaluiert. Dabei ist garantiert, dass beide Argumente von Operatoren vor diesen ausgewertet werden (außer bei Kurzschlussauswertung).

Aber warum? Was ist mit Präzedenz??

Durch die von links nach rechts ablaufende Auswertung wird die Rechte Seite des || – Operators nicht ausgewertet.



Wiederholung

Was ist das Ergebnis für *s*?

```
1 String s = "Test: " + 5 + 7 * 5;
```

Ergebnis . . .

Wiederholung

Was ist das Ergebnis für *s*?

```
1 String s = "Test: " + 5 + 7 * 5;
```

Ergebnis . . .

’Test: 535’

WARUM?

Wiederholung

Was ist das Ergebnis für *s*?

```
1 String s = "Test: " + 5 + 7 * 5;
```

Ergebnis . . .

'Test: 535'

WARUM?

Präzedenz führt zu folgender Auswertung:

```
1 String s = ("Test: " + 5) + (7 * 5);
```

Was müssen wir tun, um 'Test: 40' zu erhalten?

Checkstyle

Was ist das?

Programm in der IDE um automatisch Programmierstil zu checken

Checkstyle

Was ist das?

Programm in der IDE um automatisch Programmierstil zu checken

Diesmal neu:

- Dokumentation
- instanceof
- Komplexität
- Leerer Konstruktor
- Nur Main
- Schlechter Bezeichner
- Schwieriger Code
- Unbenutztes Element

Sichtbarkeit

Sichtbarkeit

Sichtbarkeit

Es ist wichtig, Attribute und Methoden vor externen Zugriffen zu schützen, bzw. diesen Zugriff bewusst zuzulassen.
Dafür gibt es die folgenden Schlüsselworte:

Sichtbarkeit

Sichtbarkeit

Es ist wichtig, Attribute und Methoden vor externen Zugriffen zu schützen, bzw. diesen Zugriff bewusst zuzulassen.
Dafür gibt es die folgenden Schlüsselworte:

Modifier	Klasse	Paket	Unterklasse	Welt
public	Y	Y	Y	Y
protected	Y	Y	Y	N
<no modifier>	Y	Y	N	N
private	Y	N	N	N

Sichtbarkeit - Merkregeln

Sichtbarkeit - Merkregeln für unser Modul

- Alles hat einen Modifier (nein, <no modifier> ist kein Modifier)
- *private* für Attribute, Hilfsmethoden, private Konstruktoren
- *public* definiert Schnittstellen zu anderen Klassen
- *protected* falls notwendig

Sichtbarkeit - Merkregeln

Sichtbarkeit - Merkregeln für unser Modul

- Alles hat einen Modifier (nein, <no modifier> ist kein Modifier)
- *private* für Attribute, Hilfsmethoden, private Konstruktoren
- *public* definiert Schnittstellen zu anderen Klassen
- *protected* falls notwendig

- **Beginne immer mit *private* und lockere die Sichtbarkeit.**

Sichtbarkeit - Beispiel

Beispiel

```java
class Car {
    public String manufacturer; // Public
    private String name;        // nur innerhalb Klasse
}

class Test {
    Car car = new Car();          // neues Objekt erzeugen
    car.manufacturer = "Audi";    //  Attributzugriff  erlaubt
    car.name = "TT";              // FEHLER!
}
```

Und nun?

Wie können wir aber auf solche *private* Attribute / Methoden zugreifen?

Getter / Setter

Getter / Setter

Die Lösung: sogenannte Getter und Setter!
Getter helfen beim kontrollierten Auslesen eines private Attributes
Setter helfen beim kontrollierten Setzen eines private Attributes

Grundsätzlich . . .

. . . sollten Attribute einer Klasse nur über *Getter* und *Setter* zugreifbar sein.
Setter ermöglichen auch, Plausibilitätsabfragen zu implementieren und mögliche Fehlerzustände zu verhindern.

Getter / Setter - Beispiel

Beispiel

```java
class Car {
    private String manufacturer;
    private String name;
    public String getManufacturer() {
        return this.manufacturer;
    }
    public void setName(String name) {
        this.name = name;
    }
}
class Test {
    Car car = new Car();
    System.out.println("Manufacturer: " + car.getManufacturer());
    car.setName("TT");
}
```

Methoden und Konstruktoren

Methoden

Wozu?

Methoden

Wozu?

- Methoden realisieren das dynamische Verhalten von Objekten

Methoden

Wozu?

- Methoden realisieren das dynamische Verhalten von Objekten
- Methoden berechnen etwas aus dem aktuellen Zustand und liefern das Ergebnis zurück

Methoden

Wozu?

- Methoden realisieren das dynamische Verhalten von Objekten
- Methoden berechnen etwas aus dem aktuellen Zustand und liefern das Ergebnis zurück
- Methoden können auch ganz wildes Zeug machen . . .

Deklaration

```
1 Sichtbarkeit  Ergebnistyp methodenname(Parameterliste) {
2             Methodenrumpf
3 }
```

- Ergebnistyp: void (nichts) oder entsprechende(r) Datentyp / Klasse
- Parameterliste: $\text{Typ}_1\ \text{name}_1, \text{Typ}_2\ \text{name}_2, \ldots, \text{Typ}_n\ \text{name}_n$

Methoden II

Methodenaufruf

- `objectName.methodName(Arguments);`

Methoden II

Methodenaufruf

- `objectName.methodName(Arguments);`
  - Und was, wenn Ergebnistyp != *void*?
- Argumentenliste: $\text{Ausdruck}_1, \ldots, \text{Ausdruck}_n$

Mehrere Methoden mit gleichem Namen

- Es können mehrere Methoden mit dem gleichen Namen existieren.
- Dabei müssen sie sich aber in der Art und/oder Anzahl ihrer Parameter unterscheiden.
- Stichworte: Überladen, Überschreiben

Methoden - Beispiele

Methoden mit gleichem Namen

- void doSomething(int a, int b)
- void doSomething(int a, String b)
- void doSomething(int a, int b, double c)
- void doSomething(int a, long b)

Methoden - Beispiele

Methoden mit gleichem Namen

- void doSomething(int a, int b)
- void doSomething(int a, String b)
- void doSomething(int a, int b, double c)
- void doSomething(int a, long b)

toString-Methode - üblicherweise in jeder Klasse

```
1 class Point {
2     int x;
3     int y;
4      ... so_much_to_do ...
5     @Override
6     public String toString() {
7         return ("(" + x + ", " + y + ")");
8     }
9 }
```

main-Methode

Allgemein

main-Methode

Allgemein

- Einstiegspunkt in das Programm
- Jedes **ausführbare** Programm muss die main-Methode **genau einmal** enthalten
- Programm endet beim Verlassen der main-Methode

main-Methode

Allgemein

- Einstiegspunkt in das Programm
- Jedes **ausführbare** Programm muss die main-Methode **genau einmal** enthalten
- Programm endet beim Verlassen der main-Methode

Wie funktioniert das mit diesen „Kommandozeilenargumenten“?

main-Methode

Allgemein

- Einstiegspunkt in das Programm
- Jedes **ausführbare** Programm muss die main-Methode **genau einmal** enthalten
- Programm endet beim Verlassen der main-Methode

Wie funktioniert das mit diesen „Kommandozeilenargumenten“?

- ```
  public static void main(String[] args) {
  /* Anweisungen */
  }
  ```
- `String... args` ist auch erlaubt
- Aufruf mit `java Programm a 1 Text etc`
- Achtung, die Elemente sind alle Strings!
  ```

Konstruktoren

Wozu?

Mit Konstruktoren werden . . .

Konstruktoren

Wozu?

Mit Konstruktoren werden ...

- neue Objekte erzeugt

Konstruktoren

Wozu?

Mit Konstruktoren werden ...

- neue Objekte erzeugt
- Anfangswerte von Objekten festgelegt

Konstruktoren

Wozu?

Mit Konstruktoren werden ...

- neue Objekte erzeugt
- Anfangswerte von Objekten festgelegt
- Startzustände von Objekten bestimmt

Konstruktoren

Wozu?

Mit Konstruktoren werden ...

- neue Objekte erzeugt
- Anfangswerte von Objekten festgelegt
- Startzustände von Objekten bestimmt

Merkmale

Konstruktoren

Wozu?

Mit Konstruktoren werden ...

- neue Objekte erzeugt
- Anfangswerte von Objekten festgelegt
- Startzustände von Objekten bestimmt

Merkmale

- Name des Konstruktors entspricht dem Namen der Klasse

Konstruktoren

Wozu?

Mit Konstruktoren werden ...

- neue Objekte erzeugt
- Anfangswerte von Objekten festgelegt
- Startzustände von Objekten bestimmt

Merkmale

- Name des Konstruktors entspricht dem Namen der Klasse
- Objekte werden mit dem Operator '*new*' erzeugt:

Konstruktoren

Wozu?

Mit Konstruktoren werden ...

- neue Objekte erzeugt
- Anfangswerte von Objekten festgelegt
- Startzustände von Objekten bestimmt

Merkmale

- Name des Konstruktors entspricht dem Namen der Klasse
- Objekte werden mit dem Operator '*new*' erzeugt:
- Mehrere Konstruktoren mit gleichem Namen können existieren
  - Dabei müssen sie sich aber in der Art und/oder Anzahl ihrer Parameter unterscheiden.

Konstruktoren - Beispiel

Wie könnten wir ein kleines Auto der Klasse *Car* instantiieren?

Konstruktoren - Beispiel

Wie könnten wir ein kleines Auto der Klasse *Car* instantiieren?

```
Car smallCar = new Car();
```

Es könnten natürlich auch noch Parameter mit übergeben werden:

```
Car smallCar = new Car(“Smart”);
```

Konstruktoren

Konstruktoren - Wissenswertes

- Hat eine Klasse keinen Konstruktor, wird der parameterlose *default*-Konstruktor generiert.

Konstruktoren

Konstruktoren - Wissenswertes

- Hat eine Klasse keinen Konstruktor, wird der parameterlose *default*-Konstruktor generiert.
- Schreibt ihr einen Konstruktor, wird kein default-Konstruktor erzeugt.

Konstruktoren

Konstruktoren - Wissenswertes

- Hat eine Klasse keinen Konstruktor, wird der parameterlose *default*-Konstruktor generiert.
- Schreibt ihr einen Konstruktor, wird kein default-Konstruktor erzeugt.
- Konstruktoren in Java können sich gegenseitig aufrufen.
  Dies MUSS an erster Stelle im aufrufenden Konstruktor geschehen! (sonst Compiler-Fehler!)

Konstruktoren

Konstruktoren - Wissenswertes

- Hat eine Klasse keinen Konstruktor, wird der parameterlose *default*-Konstruktor generiert.
- Schreibt ihr einen Konstruktor, wird kein default-Konstruktor erzeugt.
- Konstruktoren in Java können sich gegenseitig aufrufen.
  Dies MUSS an erster Stelle im aufrufenden Konstruktor geschehen! (sonst Compiler-Fehler!)
- Aufzurufender Konstruktor wird als normale Methode angesehen und mit *this* aufgerufen.
  Compiler unterscheidet anhand der () automatisch zum this-Pointer.

Konstruktoren

Konstruktoren - ohne Verkettung

```java
public class Car {
    public String name;
    public int year;

    public Car(String name) {
        this.name = name;
    }
    public Car(String name, int year) {
        this.name = name;
        this.year = year;
    }
}
```

Konstruktoren

Konstruktoren - mit Verkettung

```java
public class Car {
    public String name;
    public int year;

    public Car(String name) {
        this.name = name;
    }
    public Car(String name, int year) {
        this(name);
        this.year = year;
    }
}
```

static

Beispiel

```
1  class Whatever {
2      static int nextNumber = 0; // Bezug zur Klasse
3      int myNumber; // Bezug zum Objekt
4
5      public Whatever() {
6          this.myNumber = nextNumber;
7          nextNumber++;
8          // Oder Whatever.nextNumber++; als direkte
9          // Referenz der Attribute
10         // DON'T: nextNumber = nextNumber + 1;
11         // oder nextNumber +=1;
12     }
13 }
```

static

Beispiel

```
1 // funktioniert nur wenn x ebenfalls als `static`
2 // deklariert ist
3
4 int x = 0;
5
6 static void doSomething() {
7     x++;
8 }
```

JavaDoc

Javadoc - Wie sieht das aus?

```java
/**
 * Dies ist eine kurze Beschreibung der Klasse.
 * In diesem ersten Paragraphen steht eine Kurzzusammenfassung.
 *
 * Hier folgt eine genauere Beschreibung der Klasse,
 * Methode oder des Feldes.
 *
 * @author Sonic    - Beschreibt den Autor
 * @since version   - Dokumentiert die Version,
 *                    bei der das Element eingeführt wurde
 */
public class JavadocExample {
```

Javadoc - Wie sieht das aus?

```java
/**
 * Addiert zwei Zahlen und gibt das Ergebnis zurück.
 *
 * Arithmetische Überläufe werden nicht behandelt.
 *
 * @param a der erste Summand
 * @param b der zweite Summand
 * @return die Summe der beiden Summanden
 */
public static int add(int a, int b) {
    return a + b;
}
```

Javadoc - Wie sieht das aus?

```
/**
 * Außerdem:
 *
 * In Javadoc sind <strong>alle</strong> HTML-Tags erlaubt.
 * <ul>
 *   <li>Dies eignet sich z.B. für Listen</li>
 * </ul>
 * Oder <a href="https://example.com">Links</a>
 * und <img src="https://cat.jpg"> Bilder!
 */
```

Javadoc - Was ist das?

Javadoc...

- ... beschreibt Klassen, Methoden und Felder
- ... ist unkompliziert zu schreiben und direkt im Code
- ... erleichtert das Verständnis

Das `javadoc` Programm

- Generiert HTML(5) code
- Seit einigen Java-Versionen sogar mit Suche!
- Online Javadoc

Javadoc - Etiquette

Regeln und Ziele

- **EINHEITLICH** (in unserem Fall Englisch)
- Beschreiben das Wie und Warum
- Beschreiben Zusicherungen und Bedingungen der Ein- und Ausgabe

Javadoc - Etiquette

```java
/**
 * Divides two integers, returning the result of the
 * integer division.
 *
 * The result is an integer, so any fractional part is
 * discarded: {@code divide(20, 3)} is 6,
 * {@code divide(2, 4)} is 0.
 *
 * @param numerator the numerator, i.e. the number to divide
 * @param denominator i.e. the number to divide by.
 *        Must not be zero
 * @return the result of the integer division
 * @throws ArithmeticException if the denominator is zero
 */
public static int divide(int numerator, int denominator) {
    return numerator / denominator;
}
```

F R A G E N ? ? ?