PSE - Vorkurs Tag 4

Linus, Philipp, Tillmann, Tobias

FIUS - Fachgruppe Informatik Universität Stuttgart

09.10.2025

Recap Tag 3 - Funktionen & Scopes

Recap Tag 3 - Funktionen & Scopes

- Funktionen helfen Code auszulagern und mehrfach zu nutzen

```java
public static double avg(double a, double b, double c) {
    return (a + b + c) / 3;
}
```

Recap Tag 3 - Funktionen & Scopes

- Funktionen helfen Code auszulagern und mehrfach zu nutzen

```java
public static double avg(double a, double b, double c) {
    return (a + b + c) / 3;
}
```

- `return` gibt einen Wert zurück und beendet die Funktion

Recap Tag 3 - Funktionen & Scopes

- Funktionen helfen Code auszulagern und mehrfach zu nutzen

```java
public static double avg(double a, double b, double c) {
    return (a + b + c) / 3;
}
```

- `return` gibt einen Wert zurück und beendet die Funktion
- Scopes: Variablen nur im jeweiligen Block sichtbar

```java
int x = 5;
if (x > 0) {
    int y = 10; // y nur hier sichtbar
}
System.out.println(y); // Fehler!
```

Recap Tag 3 - Arrays

Recap Tag 3 - Arrays

Recap Tag 3 - Arrays

- Arrays speichern mehrere Werte gleichen Typs

```
1  int[] zahlen = new int[5];
2  zahlen[0] = 42;
```

Recap Tag 3 - Arrays

- Arrays speichern mehrere Werte gleichen Typs

```
1  int[] zahlen = new int[5];
2  zahlen[0] = 42;
```

- Zugriff über Index, Start bei 0

Recap Tag 3 - Arrays

- Arrays speichern mehrere Werte gleichen Typs

```
1  int[] zahlen = new int[5];
2  zahlen[0] = 42;
```

- Zugriff über Index, Start bei 0
- Initialisierung auch direkt möglich:

```
1  String[] grill = {"Wurst", "Steak", "Mais"};
```

- Mit `.length` über alle Elemente iterieren:

```
1  for (int i = 0; i < grill.length; i++) {
2      System.out.println(grill[i]);
3  }
```

Angenommen . . .

Angenommen . . .

- Wir wollen z.B. Informationen über Studierende speichern
  - Name

Angenommen . . .

- Wir wollen z.B. Informationen über Studierende speichern
  - Name
  - Matrikelnummer

Angenommen . . .

- Wir wollen z.B. Informationen über Studierende speichern
  - Name
  - Matrikelnummer
  - . . .

Angenommen . . .

- Wir wollen z.B. Informationen über Studierende speichern
  - Name
  - Matrikelnummer
  - . . .
- Informationen verarbeiten

Angenommen . . .

- Wir wollen z.B. Informationen über Studierende speichern
  - Name
  - Matrikelnummer
  - . . .
- Informationen verarbeiten
  - testen ob Prüfung bestanden

Angenommen . . .

- Wir wollen z.B. Informationen über Studierende speichern
  - Name
  - Matrikelnummer
  - . . .
- Informationen verarbeiten
  - testen ob Prüfung bestanden
  - . . .

Direkt umgesetzt

```
1  String name1 = "Melanie";
2  int matrikelnummer1 = 12345;
```

Direkt umgesetzt

```
1  String name1 = "Melanie";
2  int matrikelnummer1 = 12345;
3
4  String name2 = "Paul";
5  int matrikelnummer2 = 54321;
```

Direkt umgesetzt

```
1  String name1 = "Melanie";
2  int matrikelnummer1 = 12345;
3
4  String name2 = "Paul";
5  int matrikelnummer2 = 54321;
6
7  lernen(name1);
8  lernen(name2);
9
10 public static void lernen(String name) {
11     System.out.println(name + " sagt: \"Ich hasse mein Leben\"");
12 }
```

Direkt umgesetzt

```java
String name1 = "Melanie";
int matrikelnummer1 = 12345;

String name2 = "Paul";
int matrikelnummer2 = 54321;

lernen(name1);
lernen(name2);

public static void lernen(String name) {
    System.out.println(name + " sagt: \"Ich hasse mein Leben\"");
}
```

```
Melanie sagt: "Ich hasse mein Leben"
Paul sagt: "Ich hasse mein Leben"
```

Warum so nicht?!

Warum so nicht?!

- **Zusammengehörige Eigenschaften sind getrennt**
  `name1`, `matriklenummer1`

Warum so nicht?!

- **Zusammengehörige Eigenschaften sind getrennt**
  `name1`, `matriklenummer1`
- **Redundanz**
  Immer wieder dieselben zwei Variablen - für jede Person

Warum so nicht?!

- **Zusammengehörige Eigenschaften sind getrennt**
  `name1`, `matriklenummer1`
- **Redundanz**
  Immer wieder dieselben zwei Variablen - für jede Person
- **Kaum skalierbar**
  Für 2 Studenten okay - aber bei 200? 20.000?

Warum so nicht?!

- **Zusammengehörige Eigenschaften sind getrennt**
  `name1`, `matriklenummer1`
- **Redundanz**
  Immer wieder dieselben zwei Variablen - für jede Person
- **Kaum skalierbar**
  Für 2 Studenten okay - aber bei 200? 20.000?
- **Keine klare Struktur**
  Was genau ist ein Studierender im Code?

Warum so nicht?!

- **Zusammengehörige Eigenschaften sind getrennt**
  `name1`, `matriklenummer1`
- **Redundanz**
  Immer wieder dieselben zwei Variablen - für jede Person
- **Kaum skalierbar**
  Für 2 Studenten okay - aber bei 200? 20.000?
- **Keine klare Struktur**
  Was genau ist ein Studierender im Code?

$\longrightarrow$ Klassen schaffen Ordnung im Chaos

Was sind eigentlich Klassen?

- eine Klasse ist ein **Bauplan** für Objekte

Was sind eigentlich Klassen?

- eine Klasse ist ein **Bauplan** für Objekte
- Eine Klasse definiert:
  - welche Eigenschaften ein Student hat (Attribute)
  - was ein Student `"kann"` (Funktionen)

Was sind eigentlich Klassen?

- eine Klasse ist ein **Bauplan** für Objekte
- Eine Klasse definiert:
  - welche Eigenschaften ein Student hat (Attribute)
  - was ein Student "kann" (Funktionen)

```java
public class Student {
    // Eigenschaften (Attribute)
    String name;
    int matrikelnummer;

    // Verhalten (Funktionen)
    void lernen() {
        System.out.println(name + " sagt: \"Ich hasse mein Leben\"");
    }
}
```

Klasse(n): Objekte

- ein Objekt ist eine Instanz einer Klasse
  - z.B. Paul ist ein Objekt der Klasse Student

Klasse(n): Objekte

- ein Objekt ist eine Instanz einer Klasse
  - z.B. Paul ist ein Objekt der Klasse Student
- Objekte werden mit einem **Konstruktor** erzeugt

Klasse(n): Objekte

- ein Objekt ist eine Instanz einer Klasse
  - z.B. Paul ist ein Objekt der Klasse Student
- Objekte werden mit einem **Konstruktor** erzeugt
- das ist der Konstruktor von `Student`:

```
1  public Student(String name, int matrikelnummer) {
2          this.name = name;
3          this.matrikelnummer = matrikelnummer;
4  }
```

Klasse(n): Objekte

- ein Objekt ist eine Instanz einer Klasse
  - z.B. Paul ist ein Objekt der Klasse Student
- Objekte werden mit einem **Konstruktor** erzeugt
- das ist der Konstruktor von `Student`:

```
1  public Student(String name, int matrikelnummer) {
2          this.name = name;
3          this.matrikelnummer = matrikelnummer;
4  }
```

  - dieser gibt vor wie das neue Objekt initialisiert wird

Klasse(n): Objekte

- ein Objekt ist eine Instanz einer Klasse
  - z.B. Paul ist ein Objekt der Klasse Student
- Objekte werden mit einem **Konstruktor** erzeugt
- das ist der Konstruktor von `Student`:

```
1  public Student(String name, int matrikelnummer) {
2          this.name = name;
3          this.matrikelnummer = matrikelnummer;
4  }
```

  - dieser gibt vor wie das neue Objekt initialisiert wird
  - hat keinen Rückgabewert

Klasse(n): Objekte

- ein Objekt ist eine Instanz einer Klasse
  - z.B. Paul ist ein Objekt der Klasse Student
- Objekte werden mit einem **Konstruktor** erzeugt
- das ist der Konstruktor von `Student`:

```
1  public Student(String name, int matrikelnummer) {
2          this.name = name;
3          this.matrikelnummer = matrikelnummer;
4  }
```

  - dieser gibt vor wie das neue Objekt initialisiert wird
  - hat keinen Rückgabewert
  - heißt wie die Klasse

Let's build `paul`

- Klasse inklusive Konstruktor:

```java
public class Student{
  String name;
  int matrikelnummer;

  public Student(String name, int matrikelnummer) {
    this.name = name;
    this.matrikelnummer = matrikelnummer;
  }

  void lernen() {
    System.out.println(name + " sagt: \"Ich hasse mein Leben\"");
  }
}
```

Let's build `paul`

- Klasse mit Konstruktor (gekürzt, nicht kompilierbar):

```
1  // ... Attribute
2  public Student(String name, int matrikelnummer) {
3    this.name = name;
4    this.matrikelnummer = matrikelnummer;
5  }
6  // ... Funktionen
```

Let's build `paul`

- Klasse mit Konstruktor (gekürzt, nicht kompilierbar):

```java
// ... Attribute
public Student(String name, int matrikelnummer) {
  this.name = name;
  this.matrikelnummer = matrikelnummer;
}
// ... Funktionen
```

- `new Student(...)` ruft den oben erstellten Konstruktor auf:

```java
Student paul = new Student("Paul", 12345);
```

paul ein Highperformer

```java
// Objekt erzeugen
Student paul = new Student("Paul", 12345);

// Attribute auslesen
System.out.println(paul.name);
System.out.println(paul.matrikelnummer);

// Funktion aufrufen
paul.lernen();
```

```
Paul
12345
Paul sagt: "Ich hasse mein Leben"
```

Wie sehen Klassen in Java aus?

Wie sehen Klassen in Java aus?

- Klassen werden in eigenen Dateien gespeichert

Wie sehen Klassen in Java aus?

- Klassen werden in eigenen Dateien gespeichert
  - Dateiname = Klassenname + `.java`

Wie sehen Klassen in Java aus?

- Klassen werden in eigenen Dateien gespeichert
  - Dateiname = Klassenname + `.java`
- Beispiel: `Student.java` enthält die Klasse `Student`

Wie sehen Klassen in Java aus?

- Klassen werden in eigenen Dateien gespeichert
  - Dateiname = Klassenname + `.java`
- Beispiel: `Student.java` enthält die Klasse `Student`
- Attribute und Funktionen werden innerhalb der Klasse definiert

Sichtbarkeit in Klassen

Sichtbarkeit in Klassen

- `private`: nur innerhalb der Klasse sichtbar

Sichtbarkeit in Klassen

- `private`: nur innerhalb der Klasse sichtbar
- `public`: überall sichtbar (auch von außen)

Sichtbarkeit in Klassen

- `private`: nur innerhalb der Klasse sichtbar
- `public`: überall sichtbar (auch von außen)

```
1  public class Student {
2      private String name;
3      public int matrikelnummer;
4
5      public void lernen() {
6          System.out.println(name + " sagt: \"Ich hasse mein
           ↪  Leben\"");
7      }
8  }
```

Sichtbarkeit in Klassen

Sichtbarkeit in Klassen

- `matrikelnummer` und `lernen` ist `public` → kann man von außen verwenden

```
1  Student paul = new Student("Paul", 12345);
2
3  System.out.println(paul.matrikelnummer);
4  paul.lernen();
```

Sichtbarkeit in Klassen

- `matrikelnummer` und `lernen` ist `public` → kann man von außen verwenden

```java
Student paul = new Student("Paul", 12345);

System.out.println(paul.matrikelnummer);
paul.lernen();
```

```
12345
Paul sagt: "Ich hasse mein Leben"
```

Sichtbarkeit in Klassen

- `matrikelnummer` und `lernen` ist `public` → kann man von außen verwenden

```java
Student paul = new Student("Paul", 12345);

System.out.println(paul.matrikelnummer);
paul.lernen();
```

```
12345
Paul sagt: "Ich hasse mein Leben"
```

- `name` ist `private` → kann nur innerhalb der Klasse verwendet werden

```java
Student paul = new Student("Paul", 12345);

System.out.println(paul.name);
```

Sichtbarkeit in Klassen

- `matrikelnummer` und `lernen` ist `public` → kann man von außen verwenden

```
1  Student paul = new Student("Paul", 12345);
2
3  System.out.println(paul.matrikelnummer);
4  paul.lernen();
```

```
12345
Paul sagt: "Ich hasse mein Leben"
```

- `name` ist `private` → kann nur innerhalb der Klasse verwendet werden

```
1  Student paul = new Student("Paul", 12345);
2
3  System.out.println(paul.name);
```

```
java: name hat private-Zugriff in Student
```

Attribute können auch `final` sein

Attribute können auch `final` sein

- mit `final` kann Attribut nach Initialisierung nicht mehr geändert werden

```
1   <Sichtbarkeit> final <Datentyp> <Name> = <Wert>;
```

Attribute können auch `final` sein

- mit `final` kann Attribut nach Initialisierung nicht mehr geändert werden

```
<Sichtbarkeit> final <Datentyp> <Name> = <Wert>;
```

- Beispiel:

```
public class Student {
    private final String name;
    public int matrikelnummer;

  public Student(String name, int matrikelnummer) {
    this.name = name; //kann jetzt nie wieder geändert werden
    this.matrikelnummer = matrikelnummer; //kann beliebig oft
     ↪ geändert werden
  }
}
```

Attribute können auch `final` sein

- mit `final` kann Attribut nach Initialisierung nicht mehr geändert werden

```
<Sichtbarkeit> final <Datentyp> <Name> = <Wert>;
```

- Beispiel:

```
public class Student {
    private final String name;
    public int matrikelnummer;

  public Student(String name, int matrikelnummer) {
    this.name = name; //kann jetzt nie wieder geändert werden
    this.matrikelnummer = matrikelnummer; //kann beliebig oft
     ↪ geändert werden
  }
}
```

- Wird versucht `name` später zu ändern → Fehler

Fuiz los geht’s!

Frage 1: Wie nennt man ein Objekt, das von einer Klasse erstellt wurde?

- `Konstruktor`
- `Instanz`
- `Attribut`
- `Funktion`

Frage 1: Wie nennt man ein Objekt, das von einer Klasse erstellt wurde?

- Konstruktor
- Instanz
- Attribut
- Funktion

Frage 2: Was beschreibt Klassen am besten?

- Eine fertige Instanz von Daten
- Eine Gruppe von ähnlichen Funktionen
- Ein Bauplan für Objekte
- Eine Gruppe von Variablen

Frage 2: Was beschreibt Klassen am besten?

- Eine fertige Instanz von Daten
- Eine Gruppe von ähnlichen Funktionen
- Ein Bauplan für Objekte
- Eine Gruppe von Variablen

Frage 3: Welche Aussage trifft auf Konstruktoren zu?

- Ein Konstruktor hat immer den Rückgabetyp void
- Ein Konstruktor initialisiert Objekte
- Ein Konstruktor kann beliebig benannt werden
- Konstruktoren dürfen keine Parameter haben

Frage 3: Welche Aussage trifft auf Konstruktoren zu?

- Ein Konstruktor hat immer den Rückgabetyp void
- Ein Konstruktor initialisiert Objekte
- Ein Konstruktor kann beliebig benannt werden
- Konstruktoren dürfen keine Parameter haben

Frage 4: Eine Funktion innerhalb einer Klasse kann auf alle Attribute dieser Klasse zugreifen.

wahr

falsch

Frage 4: Eine Funktion innerhalb einer Klasse kann auf alle Attribute dieser Klasse zugreifen.

wahr

falsch

Frage 5: Was ist notwendig für eine Klasse?

- Ein Bezeichner
- Attribute
- Ein definierter Konstruktor
- Funktionen

Frage 5: Was ist notwendig für eine Klasse?

- Ein Bezeichner
- Attribute
- Ein definierter Konstruktor
- Funktionen

Frage 6: Welche Aussage ist wahr?

```java
public class Student {
    private final String name;
    public int matrikelnummer;

    public Student(String name, int matrikelnummer) {
        this.name = name;
        this.matrikelnummer = matrikelnummer;
    }

    public void lernen() {
        System.out.println(name + " sagt: \"Ich hasse mein Leben\"");
    }
}
```

Name kann nur im Konstruktor gesetzt werden

Frage 7: Was bedeutet es wenn ein Attribut private ist?

- Es kann nur innerhalb der Klasse darauf zugegriffen und verändert werden
- Es kann nur innerhalb der Klasse Verändert werden
- Es kann nur innerhalb der Klasse darauf zugegriffen werden
- Es kann nicht verändert werden

Frage 7: Was bedeutet es wenn ein Attribut private ist?

- Es kann nur innerhalb der Klasse darauf zugegriffen und verändert werden
- Es kann nur innerhalb der Klasse Verändert werden
- Es kann nur innerhalb der Klasse darauf zugegriffen werden
- Es kann nicht verändert werden

Klassen in Klassen

Klassen in Klassen

- Klassen können auch Objekte von anderen Klassen enthalten

Klassen in Klassen

- Klassen können auch Objekte von anderen Klassen enthalten
- Beispiel: `Universitaet` enthält viele `Studenten` (Objekte der `Student` Klasse)

Klassen in Klassen

- Klassen können auch Objekte von anderen Klassen enthalten
- Beispiel: `Universitaet` enthält viele `Studenten` (Objekte der `Student` Klasse)

```
1  public class Universitaet {
2      Student[] alleStudis;
3
4      public Universitaet(int anzahl) {
5          alleStudis = new Student[anzahl];
6          for (int i = 0; i < anzahl; i++) {
7              alleStudis[i] = new Student("Paul", i);
8          }
9      }
10 }
```

Alle Studenten lernen

Alle Studenten lernen

- Neue Methode in `Universitaet`:

```
1  public void pruefungsvorbereitung() {
2          for (Student s : alleStudis) {
3                  s.lernen();
4          }
5  }
```

so schnell kanns gehen

so schnell kanns gehen

- Was passiert hier?

```
1  Universitaet uniStuttgart = new Universitaet(3);
2  uniStuttgart.pruefungsvorbereitung();
```

so schnell kanns gehen

- Was passiert hier?

```
1  Universitaet uniStuttgart = new Universitaet(3);
2  uniStuttgart.pruefungsvorbereitung();
```

  - uniStuttgart wird mit anzahl 3 erstellt

so schnell kanns gehen

- Was passiert hier?

```
1  Universitaet uniStuttgart = new Universitaet(3);
2  uniStuttgart.pruefungsvorbereitung();
```

  - uniStuttgart wird mit anzahl 3 erstellt
  - der `Universitaet` Konstruktor erstellt 3 Studenten

so schnell kanns gehen

- Was passiert hier?

```
1  Universitaet uniStuttgart = new Universitaet(3);
2  uniStuttgart.pruefungsvorbereitung();
```

  - uniStuttgart wird mit anzahl 3 erstellt
  - der `Universitaet` Konstruktor erstellt 3 Studenten
  - durch `pruefungsvorbereitung` lernen alle Studis

so schnell kanns gehen

- Was passiert hier?

```
1  Universitaet uniStuttgart = new Universitaet(3);
2  uniStuttgart.pruefungsvorbereitung();
```

  - uniStuttgart wird mit anzahl 3 erstellt
  - der `Universitaet` Konstruktor erstellt 3 Studenten
  - durch `pruefungsvorbereitung` lernen alle Studis

```
Paul sagt: "Ich hasse mein Leben"
Paul sagt: "Ich hasse mein Leben"
Paul sagt: "Ich hasse mein Leben"
```

Ende

- Folien und Aufgaben:
  `https://fius.de/index.php/pse-vk-folien/`

- Feedback:
  `https://forms.gle/JasKFVgR2ARoFVTEA`

Wenn ihr Fragen habt, sagt Bescheid!