

Tutorium Programmierung 1

| 04.02.2026

Jannes Kurzke und Fabian Bauriedl

Inhalt

1. Array und ArrayList
2. Hilfsklassen - Math, Random, Scanner
3. Objektorientierung
4. Access Modifier
5. Method Overloading
6. Constructor
7. Static Modifier

Arrays

- Speichert mehrere Datenobjekte des selben Typs
- Fest definierte Anzahl an Datenobjekten

Array - Deklaration

- Datentyp der zu speichernden Elemente + []

Eckige Klammern [Länge]

```
public static void main(String[] args) {  
    int[] int_array = new int[6];  
}
```

Array - Initialisierung

- Datentyp der zu speichernden Elemente + []

Geschweifte Klammern {Werte}

```
public static void main(String[] args) {  
    int[] int_array = {1,2,3,4,5,6};  
}
```

Array - Zugriff

- Mit eckigen Klammern
- Index beginnend bei 0!

```
public static void main(String[] args) {  
    int[] int_array = {1,2,3,4,5,6};  
    int num_1 = int_array[0];  
    int num_2 = int_array[5];  
  
    System.out.println(num_1);  
    // --> 1  
    System.out.println(num_2);  
    // --> 6  
}
```

Array - Zuweisung / Speichern

- Angabe des Index in eckigen Klammern
- Zuweisung mit Zuweisungsoperator '='

```
public static void main(String[] args) {  
    int[] int_array = {1,2,3,4,5,6};  
  
    System.out.println(int_array[3]);  
    // --> 4  
  
    int_array[3] = 9;  
    System.out.println(int_array[3]);  
    // --> 9  
}
```

Array - Länge

```
public static void main(String[] args) {  
    int[] int_array = {1,2,3,4,5,6};  
  
    System.out.println(int_array.length);  
    // --> 6  
}
```

Array - Werte Anhängen 1/2

- Problem: Arrays haben feste Größe
- Lösung: Neues Array mit neuer Größe erstellen

```
public static void main(String[] args) {  
    int[] int_array = {1,2,3,4,5,6};  
  
    int[] int_array_long = new int[int_array.length + 1];  
  
    for (int i = 0; i < int_array.length; i++) {  
        int_array_long[i] = int_array[i];  
    }  
  
    int_array_long[6] = 7;  
  
    for (int i : int_array_long) System.out.print(i + " ");  
    // --> 1 2 3 4 5 6 7  
}
```

Array - Werte Anhängen 2/2

```
public static void main(String[] args) {  
    int[] int_array = {1,2,3,4,5,6};  
  
    int[] int_array_long = addToArray(int_array, 7);  
  
    for (int i : int_array_long) System.out.print(i + " ");  
    // --> 1 2 3 4 5 6 7  
}  
  
public static int[] addToArray(int[] array, int newElement){  
    int[] returnArray = new int[array.length + 1];  
  
    for (int i = 0; i < array.length; i++) {  
        returnArray[i] = array[i];  
    }  
  
    returnArray[returnArray.length - 1] = newElement;  
  
    return returnArray;  
}
```

ArrayList - Das besserer Array

- Bessere Verison des Standard Arrays
- Kommt mit vielen nützlichen Funktionen
- Benötigt mehr Speicherplatz

```
public static void main(String[] args) {  
    ArrayList<Integer> int_list = new ArrayList<>();  
    System.out.println(int_list);  
    // --> []  
  
    int_list.add(1);  
    int_list.add(2);  
    int_list.add(3);  
  
    System.out.println(int_list);  
    // --> [1, 2, 3]  
}
```

ArrayList - Länge

- Länge einer ArrayList ist dynamisch --> muss berechnet werden
- Zugriff auf Länge über Funktion '`.size()`', **nicht** über Attribut '`.length`'

```
public static void main(String[] args) {  
    ArrayList<Integer> int_list = new ArrayList<>();  
    System.out.println(int_list.size());  
    // --> 0  
  
    int_list.add(1);  
    int_list.add(2);  
    int_list.add(3);  
  
    System.out.println(int_list.size());  
    // --> 3  
}
```

Hilfsklassen - Math

- Standard Operatoren mit '+ - * /'
- Fortgeschrittenere Operatoren mit Funktionen von Math

```
public static void main(String[] args) {  
    double num_1 = -5;  
    double num_2 = 5;  
    double num_3 = 10;  
    double result;  
  
    result = Math.abs(num_1);  
    System.out.println(result);           // --> 5.0  
  
    result = Math.sqrt(num_2);  
    System.out.println(result);           // --> 2.23606797749979  
  
    result = Math.pow(num_2, num_3);  
    System.out.println(result);           // --> 9765625.0  
}
```

Hilfsklassen - Random

- Zufällige Erzeugung von Ganzzahlen und Gleitkommazahlen

```
public static void main(String[] args) {  
    Random random = new Random();  
    int rnd_num ;  
  
    // Zahlen von 0 bis 99 --> 100 als exklusive Grenze  
    rnd_num = random.nextInt(100);  
    System.out.println(rnd_num);  
  
    // Trick um Zahlen von 1 bis 100 zu erhalten  
    rnd_num = random.nextInt(100);  
    System.out.println(rnd_num+1);  
  
    double rnd_double = random.nextDouble(100);  
    System.out.println(rnd_double);  
}
```

Hilfsklassen - Scanner

- Scanner wird verwendet um Eingaben in der Konsole zu ermöglichen

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
  
    System.out.println("Wie lautet dein Name?");  
    String benutzerEingabe = sc.nextLine();  
  
    System.out.println("Hallo " + benutzerEingabe + ", schön dich kennen zu lernen.");  
    System.out.println("Wie alt bist du?");  
  
    int alter = sc.nextInt();  
    System.out.println("Du bist also " + alter + " Jahre alt");  
}
```

Objektorientierung

- Gängiges und leicht verständliches Programmier Paradigma
- Abbildung realer Gegenstände in digitale Objekte
- Objekte verfügen über spezifische Eigenschaften und Funktionen

Beispiel **Auto**

Eigenschaften: Farbe, Leistung, Sitze, Türen, etc.

Funktionen: beschleunigen, bremsen, schließen, öffnen, etc.

Objektorientierung - Auto Implementierung 1/4



```
public class auto {  
    String farbe;  
    int leistungInPS;  
    boolean isOpen;  
    double aktuelleGeschwindigkeit;  
    double maxGeschwindigkeit;  
  
    public Auto(String farbe, int leistungInPS, double maxGeschwindigkeit){  
        ...  
    }  
  
    public boolean beschleunigen(double extraSpeed){  
        ...  
    }  
  
    public boolean bremsen(double lowerSpeed){  
        ...  
    }  
  
    public boolean schließen(){  
        ...  
    }  
  
    public boolean öffnen(){  
        ...  
    }  
}
```

Objektorientierung - Auto Implementierung 2/4



```
public class auto {
    String farbe;
    int leistungInPS;
    boolean isOpen;
    double aktuelleGeschwindigkeit;
    double maxGeschwindigkeit;

    public Auto(String farbe, int leistungInPS, double maxGeschwindigkeit){
        this.farbe = farbe;
        this.leistungInPS = leistungInPS;
        this.maxGeschwindigkeit = maxGeschwindigkeit;
        this.aktuelleGeschwindigkeit = 0;
        this.isOpen = false;
    }

    ...
}
```

Objektorientierung - Auto Implementierung 3/4



```
public class auto {
    String farbe;
    int leistungInPS;
    boolean isOpen;
    double aktuelleGeschwindigkeit;
    double maxGeschwindigkeit;

    ...

    public boolean beschleunigen(double extraSpeed){
        if (this.aktuelleGeschwindigkeit + extraSpeed <= maxGeschwindigkeit) {
            this.aktuelleGeschwindigkeit += extraSpeed;
            return true;
        }
        return false;
    }

    public boolean bremsen(double lowerSpeed){
        if (this.aktuelleGeschwindigkeit - lowerSpeed >=0) {
            this.aktuelleGeschwindigkeit -= lowerSpeed;
            return true;
        }
        return false;
    }

    ...
}
```

Objektorientierung - Auto Implementierung 4/4



```
public class auto {
    String farbe;
    int leistungInPS;
    boolean isOpen;
    double aktuelleGeschwindigkeit;
    double maxGeschwindigkeit;

    ...

    public boolean schließen(){
        if (this.isOpen) {
            this.isOpen = false;
            return true;
        }
        return false;
    }

    public boolean öffnen(){
        if (!this.isOpen) {
            this.isOpen = true;
            return true;
        }
        return false;
    }
}
```

Objektorientierung - Auto Verwendung

```
public class main {  
    public static void main(String[] args) {  
        Auto auto1 = new Auto("grün", 100, 130);  
  
        System.out.println("Beschleunige um 100km/h");  
        System.out.println("Erfolg: " + auto1.beschleunigen(100));  
        System.out.println("Aktuelle Geschwindigkeit: " + auto1.aktuelleGeschwindigkeit);  
  
        // Beschleunige um 100km/h  
        // Erfolg: true  
        // Aktuelle Geschwindigkeit: 100.0  
    }  
}
```

Access Modifier

- Standard: jede Variable, jede Methode, jede Klasse kann von überall verwendet werden
- Access Modifier limitieren den Zugriff auf Variablen, Methoden und Klassen

Wir kennen: public und private

Access Modifier - Public

- Zugriff kann direkt und von überall erfolgen

```
public class Mensch {  
    public String name;  
    public int alter;  
    ...  
}
```

```
public class main {  
    public static void main(String[] args) {  
        Mensch thomas = new Mensch("Thomas", 21);  
  
        System.out.println(thomas.name);  
        System.out.println(thomas.alter);  
    }  
}
```

Access Modifier - Private

- Zugriff nur innerhalb des Objektes möglich
- Variablen Abfrage über Getter/ Setter

```
public class Mensch {  
    private String name;  
    private int alter;  
  
    public Mensch(String name, int alter){  
        this.name = name;  
        this.alter = alter;  
    }  
  
    public String getName(){  
        return this.name;  
    }  
  
    public int getAlter(){  
        return this.alter;  
    }  
}
```

```
public class main {  
    public static void main(String[] args) {  
        Mensch thomas = new Mensch("Thomas", 21);  
  
        System.out.println(thomas.getName());  
        System.out.println(thomas.getAlter());  
    }  
}
```

Method Overloading

- Definition der gleichen Methode **aber** mit unterschiedlichen Argumenten
- Java ist clever und wählt die Implementierung, welcher den gegebenen Argumenten entspricht

```
public class Calculator {  
    public int addNums(int num1, int num2){  
        System.out.println("double integer implementation");  
        return num1 + num2;  
    }  
  
    public int addNums(int num1, int num2, int num3){  
        System.out.println("triple integer implementation");  
        return num1 + num2 + num3;  
    }  
  
    public double addNums(double num1, double num2){  
        System.out.println("double implementation");  
        return num1 + num2;  
    }  
}
```

```
public class main {  
    public static void main(String[] args) {  
        Calculator calc = new Calculator();  
  
        System.out.println(calc.addNums(1, 1));  
        // double integer implementation  
        // 2  
  
        System.out.println(calc.addNums(1, 1, 1));  
        // tripple integer implementation  
        // 3  
  
        System.out.println(calc.addNums(1.0, 1.0));  
        // double implementation  
        // 2.0  
    }  
}
```

Constructor

- Anleitung wie man die Instanz einer Klasse baut
- Definition als Methode ohne Return-Typ

```
public class Mensch {  
    private String name;  
    private int alter;  
  
    public Mensch(String name, int alter){  
        this.name = name;  
        this.alter = alter;  
    }  
}
```

Constructor - Overloading

```
public class Mensch {  
    public String name;  
    public int alter;  
  
    public Mensch(String name, int alter){  
        this.name = name;  
        this.alter = alter;  
    }  
  
    public Mensch (String name){  
        this.name = name;  
        this.alter = 20;  
    }  
  
    public Mensch(int alter){  
        this.name = "John Doe";  
        this.alter = alter;  
    }  
  
    public Mensch(){  
        this.name = "John Doe";  
        this.alter = 20;  
    }  
}
```

Constructor - Overloading Best Practice

```
public class Mensch {  
    public String name;  
    public int alter;  
  
    public Mensch(String name, int alter){  
        this.name = name;  
        this.alter = alter;  
    }  
  
    public Mensch (String name){  
        this(name, 20);  
    }  
  
    public Mensch(int alter){  
        this("John Doe", alter);  
    }  
  
    public Mensch(){  
        this("John Doe", 20);  
    }  
}
```

Static Modifier

- Unveränderliche Variable
- In allen Instanzen einer Klasse gleich

```
public class Auto {  
    private boolean isOpen = false;  
    private double aktuelleGeschwindigkeit = 0;  
    private double maxGeschwindigkeit;  
  
    private static int anzahlRäder = 4;    // <-- Static Modifier  
  
    public Auto(double maxGeschwindigkeit){  
        this.maxGeschwindigkeit = maxGeschwindigkeit;  
  
        this.anzahlRäder = 6;                // --> fehler  
    }  
    ...  
}
```

Übungen

git fetch

git checkout <branch_name>

- Array & ArrayList: semester1/array-list
- Klassen: semester1/classes
- Access modifier: semester1/access-modifier
- Methoden Überladung: semester1/method-overloading