DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Tutorium Programmierung 1

12.02.2026

Jannes Kurzke und Fabian Bauriedl

Inhalt

1. Java API (Wrapper, Datum, IO)
2. Final Modifier
3. Enumeration
4. Klassen Diagramme
5. Vererbung
6. Finale Klassen
7. Protected Modifier
8. Polymorphie
9. Abstract
10. Interface
11. Komparator

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Wrapper Klassen

- Wir kennen **Primitive** Datentypen und **Komplexe** Datentypen
- Komplexe Datentypen bringen fortgeschrittene Funktionen mit sich

Numerische Wrapper Klassen

```
public static void main(String[] args) {
    // Numerische Wrapper funktionieren mit Integer, Long und Double
    int intResult;
    String binResult;

    intResult = Integer.parseInt("1");              // --> 1

    intResult = Integer.compare(1, 2);              // --> -1
    intResult = Integer.compare(2, 2);              // --> 0
    intResult = Integer.compare(2, 1);              // --> 1

    intResult = Integer.min(2, 1);                  // --> 1
    intResult = Integer.max(2, 1);                  // --> 2
    intResult = Integer.sum(2, 1);                  // --> 3

    binResult = Integer.toBinaryString(69);         // --> "1000101"
}
```

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Character Wrapper

```
public static void main(String[] args) {
    //Char Wrapper

    boolean boolResult;

    boolResult = Character.isDigit('1');         // --> True
    boolResult = Character.isLetter('A');        // --> True
    boolResult = Character.isWhitespace(' ');    // --> True
    boolResult = Character.isUpperCase('A');     // --> True
    boolResult = Character.isLowerCase('a');     // --> True
}
```

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Boolean Wrapper

```
public static void main(String[] args) {
    boolean boolResult;

    boolResult = Boolean.parseBoolean("TrUe");     // --> true (ignoriert Groß-/Kleinschreibung)
    boolResult = Boolean.logicalXor(true, false);  // --> true
    boolResult = Boolean.logicalAnd(true, true);   // --> true
}
```

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Datums Objekte

Datums Objekte

DHBW
Duale Hochschule

```
public static void main(String[] args) {
    LocalDateTime now = LocalDateTime.now();

    System.out.println(now.getYear());           // --> 2026
    System.out.println(now.getMonth());          // --> FEBRUARY
    System.out.println(now.getMonthValue());     // --> 2
    System.out.println(now.getDayOfMonth());     // --> 18

    System.out.println(now.plusWeeks(2));
            // --> 2026-03-04T22:52:18.034961767
    System.out.println(now.minusHours(5));
            // --> 2026-02-18T17:53:21.239400596
    System.out.println(now.withHour(12).withMinute(0));
            // --> 2026-02-18T12:00:19.056285733
}
```

Datums Objekte

```java
public static void main(String[] args) {
    LocalDate today = LocalDate.now();
    LocalDate examDate = LocalDate.of(2026, 3, 10);

    long daysUntilExam;
    daysUntilExam = ChronoUnit.DAYS.between(today, examDate);       //--> 20

    boolean isLeap = today.isLeapYear();
    boolean isBefore = today.isBefore(examDate);
    DayOfWeek dayOfWeek = today.getDayOfWeek();                     // -> WEDNESDAY
    String dayString = today.getDayOfWeek().toString();             // -> WEDNESDAY
    int daysInMonth = today.lengthOfMonth();                        // --> 28
}
```

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Dateien Lesen

- Wir kennen bereits den **Scanner** --> Nutzereingabe über die Konsole
- Scanner kann nun ebenso verwendet werden, um Dateien zu lesen
- Bei der Arbeit mit Dateien muss auf korrekte Fehler Behandlung geachtet werden
- Verwendung von **Try-Ressource** Blöcken wird empfohlen

Dateien Lesen


```java
public static void main(String[] args) {
    File file = new File("HelloWorld.txt");

    try (Scanner sc = new Scanner(file)) {
        while (sc.hasNextLine()) {
            String currentLine = sc.nextLine();
            System.out.println(currentLine);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

> Dateien Speichern wird mit dem **FileWriter** erledigt --> nicht im ersten Semester

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Final Modifier

- Anwendbar auf: Klassen, Methoden, Attribute, Variablen
- Zuweisung eines Wertes erfolgt **final**
  - Das bedeutet er lässt sich nicht mehr verändern
- Finale Attribute müssen im Konstruktur gesetzt oder direkt initialisiert werden

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Final Modifier

```
public class Car {
    final int wheelCount = 4;
    final String make;
    final String model;

    public Car(String make, String model) {
        this.make = make;
        this.model = model;
    }
}
```

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Enumerations

- Feste Gruppe von bekannten Konstanten
- Sehr komfortable und schnelle Art mehrere, ähnliche Objekte anzulegen
- Können Attribute, Funktionen und Konstruktor verwenden

  > Konstruktur ist hier **private**

- **Best Practice:** Namen der Enum-Objekte in Caps-Lock und Singluar

Enumerations - Car-Klasse Upgrade

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

```
public class Car {
    final int wheelCount = 4;
    final String make;
    final String model;
    final EngineEnum engine;

    public Car(String make, String model, EngineEnum engine) {
        this.make = make;
        this.model = model;
        this.engine = engine;
    }
}
```

Enumerations


```
public enum EngineEnum {
    Diesel,
    Petrol,
    Electric,
    Hydrogen;
}
```

```
public static void main(String[] args) {
    Car Id3 = new Car("VW", "ID.3", EngineEnum.Electric);

    System.out.println(Id3.engine);     // --> Electric
}
```

Enumerations - Upgrade


```
public enum EngineEnum {
    Diesel(0.1),
    Petrol(0.2),
    Electric(1.0),
    Hydrogen(0.9);

    private double ecoRating;

    private EngineEnum(double ecoRating){
        this.ecoRating = ecoRating;
    }

    public double getEcoRating(){
        return this.ecoRating;
    }
}
```

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Enumerations - Upgrade

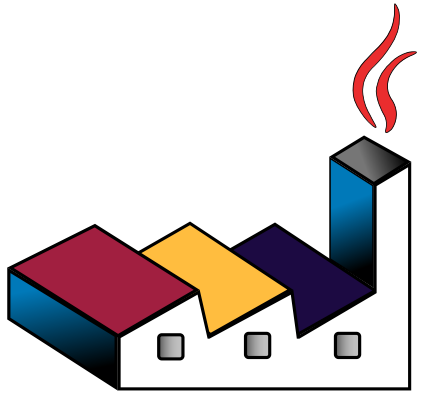
```
public static void main(String[] args) {
    Car Id3 = new Car("VW", "ID.3", EngineEnum.ELECTRIC);
    Car P911 = new Car("Porsche", "9/11", EngineEnum.PETROL);

    System.out.println(Id3.engine);                    // --> ELECTRIC
    System.out.println(Id3.engine.getEcoRating());     // --> 1.0
    System.out.println(P911.engine);                   // --> PETROL
    System.out.println(P911.engine.getEcoRating());    // --> 0.2
}
```

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Unified Modelling Language

- Standardisierte grafische Sprache
- Modellierung und Spezifikation von Software
- Visualisierung, Konstruktion und Dokumentation von Software-Architektur
- Unabhängig von der verwendeten Programmiersprache
- Strukturdiagramme z.B. Klassendaigramme
- Verhaltensdiagramme z.B. Sequenzdiagramme

UML- Tooling

Draw.io	Eclipse-Papyrus	Plant-UML
		
Einfache Browser Anwendung	Profi Werkzeug mit Projekt Imports und Exports	UML-As-Code Lösung

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Klassendiagramme

- Verwendet den UML Standard
- Grafische modellierung von Klassen und deren Beziehung
- Empfiehlt sich beim Entwurf der Software-Architektur
  - **bevor** man Code schreibt

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Klassendaigramme - UML Syntax

UML Syntax / Symbol	Bedeutung	Beispiel
+	public	`+ name: String`
-	private	`- alter: int`
#	protected	`# id: int`
name: T	Attribut	`- gehalt: double`
name(p: Typ): Typ	Methode	`+ getGehalt(bonus: int): double`

Klassendaigramme - UML Syntax

UML Syntax / Symbol	Bedeutung	Beispiel
Unterstrichen	static	`- MAX_SPEED: int`
Kursiv	abstract	`+ berechnePreis(): double`
<<interface>>	Interface	`<<interface>> Runnable`
<<enumeration>>	Enumeration	`<<enumeration>> Engine`
Leere Raute	Aggregation	`Parkplatz` ◇——— `Auto`
Gefüllte Raute	Komposition	`Auto` ♦——— `Motor`
Durchgezogene Linie, leerer Pfeil	Vererbung	`PKW` ———▷ `Fahrzeug`
Gestrichelte Linie, leerer Pfeil	Implementierung	`Auto` ----▷ `Fahrbar`

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Klassendiagramme - Beispiel


Example UML Class-Diagramm

Vererbung

- Wiederverwendung von Attributen und Funktionen
- Vererbung über **extends** Kezyword im Klassen-Kopf
- Spiegelt eine **ist-ein** Beziehung wieder --> ein **Auto** ist ein **Fahrzeug**
- Die **Sub-Klasse** (Child) erbt von der **Super-Klasse** (Parent)
- Java erlaubt nur Einfach-Vererbung --> nur eine Super-Klasse
- Konstruktoren werden nicht verebt --> Aufruf über **super** Keyword

> **Ziel:** Hohes Maß an Generalisierung erhalten --> Code Recycling

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Vererbung - Modelling


Inheritance UML Modell

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Vererbung - Code

> Super Klasse

```
public class Vehicle {
    final double topSpeed;

    public Vehicle(double topSpeed){
        this.topSpeed = topSpeed;
    }
}
```

Vererbung - Code

> Sub-Klasse

```
public class Car extends Vehicle {
    int wheelCount = 4;
    String make;
    String model;
    EngineEnum engine;

    public Car(String make, String model, EngineEnum engine, double topSpeed) {
        super(topSpeed);
        this.make = make;
        this.model = model;
        this.engine = engine;
    }
}
```

- **Car** verfügt nun über alle Eigenschaften und Funktionen der **Vehicle** Klasse

Vererbung - Code

> Verwendung

```
public static void main(String[] args) {
    Car Id3 = new Car("VW", "ID.3", EngineEnum.ELECTRIC, 150);

    System.out.println(Id3.engine);                          // --> ELECTRIC
    System.out.println(Id3.engine.getEcoRating());           // --> 1.0
    System.out.println(Id3.topSpeed);                        // --> 150.0
}
```

- Die **topSpeed** Eigenschaft auf **Vehicle** kann nun von **Car** verwendet und gesetzt werden

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Protected Modifier

- Access modifier, so wie public und private
- Anwendbar auf Attribute, Methoden und Konstruktoren
- Sichtbarkeits-Bereich
  - Eigene Klasse
  - Klassen im selben Package
  - Alle **Unterkassen** --> hier bietet sich die Verwendug an
- Strenger als **public**, offener als **private**

Protected Modifier - Code


```
public class Vehicle {

    protected int wheelCount;
    protected int seatCount;
}
```

```
public class Car extends Vehicle {
    String make;
    String model;
    EngineEnum engine;

    public Car(String make, String model, EngineEnum engine, int wheelCount, int seatCount){
        this.make = make;
        this.model = model;
        this.engine = engine;
        this.wheelCount = wheelCount;
        this.seatCount = seatCount;
    }
}
```

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Protected Modifer - Code

```
public static void main(String[] args) {
    Car Id3 = new Car("VW", "ID.3", EngineEnum.ELECTRIC, 4, 5);

    System.out.println(Id3.wheelCount);     // --> 4
    System.out.println(Id3.seatCount);      // --> 5
}
```

Polymorphie

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

- Objekt kann als Instanz seiner Klasse oder einer seiner Superklassen behandelt werden
- Wichtig: Trennung zwischen **Referenztyp** und **Objekttyp**
- Variable vom Typ Super-Klasse kann Objekte der Sub-Klasse speichern
- **Upcast:** Umwandlung in die Superklasse
  - **Explizit**/Manuell oder **Implizit**/ Automatisch möglich
- **Downcast:** Rückumwandlung in die Subklasse
  - Erfolgt immer **Explizit**
  - Downcast eines falschen Objekts führt zu Fehler
- **instanceof:** Prüfung ob Objekt einen bestimmten Typ hat

Polymorphie - Code

DHBW
Duale Hochschule
Baden-Württemberg

```
public static void main(String[] args) {
    Car Id3 = new Car("VW", "ID.3", EngineEnum.ELECTRIC, 150);

    System.out.println(Id3.engine);                          // --> ELECTRIC

    Vehicle vehicle = (Vehicle) Id3;          // <-- upcast (explizit)
    Vehicle vehicle2 = Id3;                   // <-- upcast (implizit)
    System.out.println(vehicle.engine);                      // --> FEHLER

    Car car = (Car) vehicle;                  // <-- downcast (explizit)
    System.out.println(car.engine);                          // --> ELECTRIC

    Boolean resultBool;
    resultBool = Id3 instanceof Vehicle;                     // --> true
}
```

Abstract Modifier

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

- Anwendbar auf Klassen und Methoden
- Dient alls Kennzeichnung eines "leeren Bauplans"
- Abstrakte Klassen
  - Können nicht instanziiert werden
  - Liefert struktur sowie Methoden für Unterklassen --> Basis Klasse
  - Kann auch nicht abstrakte Methoden beeinhalten
- Abstrakte Methoden
  - Kein Methodenrumpf --> endet mit Semikolon, ohne {}
  - Eigene Implementierung in Unterklassen wird erzwungen
  - Wenn abstrakte Methoden verwendet werden, muss die Klasse auf abstrakt sein

Abstract Modifier - Code


```
public abstract class Vehicle {
    protected int wheelCount;
    protected int seatCount;
    abstract public int getWheelCount();
    abstract public int getSeatCount();
}
```

```
public class Car extends Vehicle {
    public Car(int wheelCount, int seatCount){
        this.wheelCount = wheelCount;
        this.seatCount = seatCount;
    }
    public int getWheelCount() {
        return this.wheelCount;
    }
    public int getSeatCount() {
        return this.seatCount;
    }
}
```

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Abstract Modifier - Code

```
public static void main(String[] args) {
    Car Id3 = new Car(4, 5);

    System.out.println(Id3.getWheelCount());     // --> 4
    System.out.println(Id3.getSeatCount());      // --> 5
}
```

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Final Modifier - Klassen

- Funktioniert ähnlich wie bei Methoden und Variablen
- Es können **keine** Sub-Klassen gebildet werden
- **Final** Keyword nach **Access**-Modifier (public, private) in Klassen Kopf

Finale Klassen - Code

```
public class Cat{
    final public void makeSound() {
        System.out.println("Miau");
    }
}
```

```
public final class BetterCat extends Cat {          // --> Final Modifier
    public void makeBetterSound(){
        System.out.println("Miau Miau");
    }
}
```

```
public class WayBetterCat extends BetterCat{        // --> Fehler
    public void makeWayBetterSound(){
        System.out.println("Miau Miau Miau");
    }
}
```

Interface

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

- Kann man als **Vertrag** sehen, dass eine Klasse bestimmte Funktionen hat
- Funktionen eines Interface sollten **abstract** sein
- Verwendung als Referenzvariable möglich, kann jedoch nicht instanziiert werden (kein `new Interface()`)
- **Interface** Keyword anstelle des **Class** Keywords
- Verwendung in anderer Klasse mit **implements** Keyword
- **Schnittstelle** zwischen Entwickler einer Klasse und dem Verwender
- Eine Klasse kann **mehrere** Interfaces implementieren

Interface - Code


```java
public interface AnimalInterface {
    public void makeSound();
}
```

```java
public class Cat implements AnimalInterface{
    final public void makeSound() {
        System.out.println("Miau");
    }
}
```

```java
public class Dog implements AnimalInterface {
    public void makeSound() {
        System.out.println("Wau");
    }
}
```

Interface - Code

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

```
public static void main(String[] args) {
    ArrayList<AnimalInterface> animalList = new ArrayList<AnimalInterface>();

    AnimalInterface animal1 = new Dog();
    AnimalInterface animal2 = new Cat();

    animalList.add(animal1);
    animalList.add(animal2);

    for (AnimalInterface animal : animalList) {
        animal.makeSound();
    }
}
```

Komparator

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

- Vergleich von zwei Klassen
- Benutzerdefiniertes Sortier Verhalten möglich
- Wird ausserhalb der zu sortierenden Klasse implementiert
- Implementiert das Interface `java.util.Comparator<T>`
- Implementiert dann dann die Methode `int compare(T o1, T o2`
- Wenn o1 kleiner als 02 --> return -1
- Wenn o1 größer als o2 --> return 1
- Wenn o1 und o2 gleich sind --> return 0

Komparator - Code


```
public class CatComparator implements Comparator<Cat> {
    @Override
    public int compare(Cat cat1, Cat cat2) {
        if (cat1.age < cat2.age) {
            return -1;
        } else if (cat1.age > cat2.age) {
            return 1;
        } else {
            return 0;
        ...
```

```
public class Cat{
    public int age;

    public Cat(int age ){
        this.age = age;
    }
}
```

Komparator - Code

```
import java.util.ArrayList;

public class main {
    public static void main(String[] args) {
        ArrayList<Cat> catList = new ArrayList<Cat>();

        catList.add(new Cat(2));
        catList.add(new Cat(7));
        catList.add(new Cat(1));
        catList.add(new Cat(10));
        catList.add(new Cat(1));

        for (Cat cat : catList) {                       // --> nicht sortiert
            System.out.println(cat.age);
        }

        System.out.println("~~~~~~~~~~~~~~~");
        catList.sort(new CatComparator());

        for (Cat cat : catList) {                       // --> sortiert
            System.out.println(cat.age);
        }
    }
}
```