



Woche 2 - Java

Agenda

Woche 2

- Tag 1: -
- Tag 2: Einführung in Java
- Tag 3: Einführung in Java
- Tag 4: Java Projekt
- Tag 5: Java Projekt

Einführung in die Programmiersprache Java

Was ist Java?

- Entwickelt von James Gosling 1995 bei Sun Microsystems
- Objektorientierte und plattformunabhängige Programmiersprache
- Weit verbreitet für plattformübergreifende Programmierung
- Bietet automatische Speicherverwaltung (Garbage Collection)
- Sicher und robust

Java

Einsatzgebiete



- Enterprise-Anwendungen
 - Viele große Unternehmen nutzen Java für geschäftskritische Anwendungen aufgrund seiner Plattformunabhängigkeit und Skalierbarkeit
- Web-Entwicklung
 - Java wird häufig für serverseitige Anwendungen (Backend) verwendet
- Cloud-basierte Systeme
 - Java wird in Cloud-Computing-Umgebungen wie AWS und Google Cloud eingesetzt, da es für verteilte Systeme gut geeignet ist

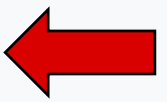
Einführung in Java

Java Development Kit

<https://adoptium.net/temurin/releases/>

Operating System	Architecture	Package Type	Version
Windows	x64	JDK	8 - LTS

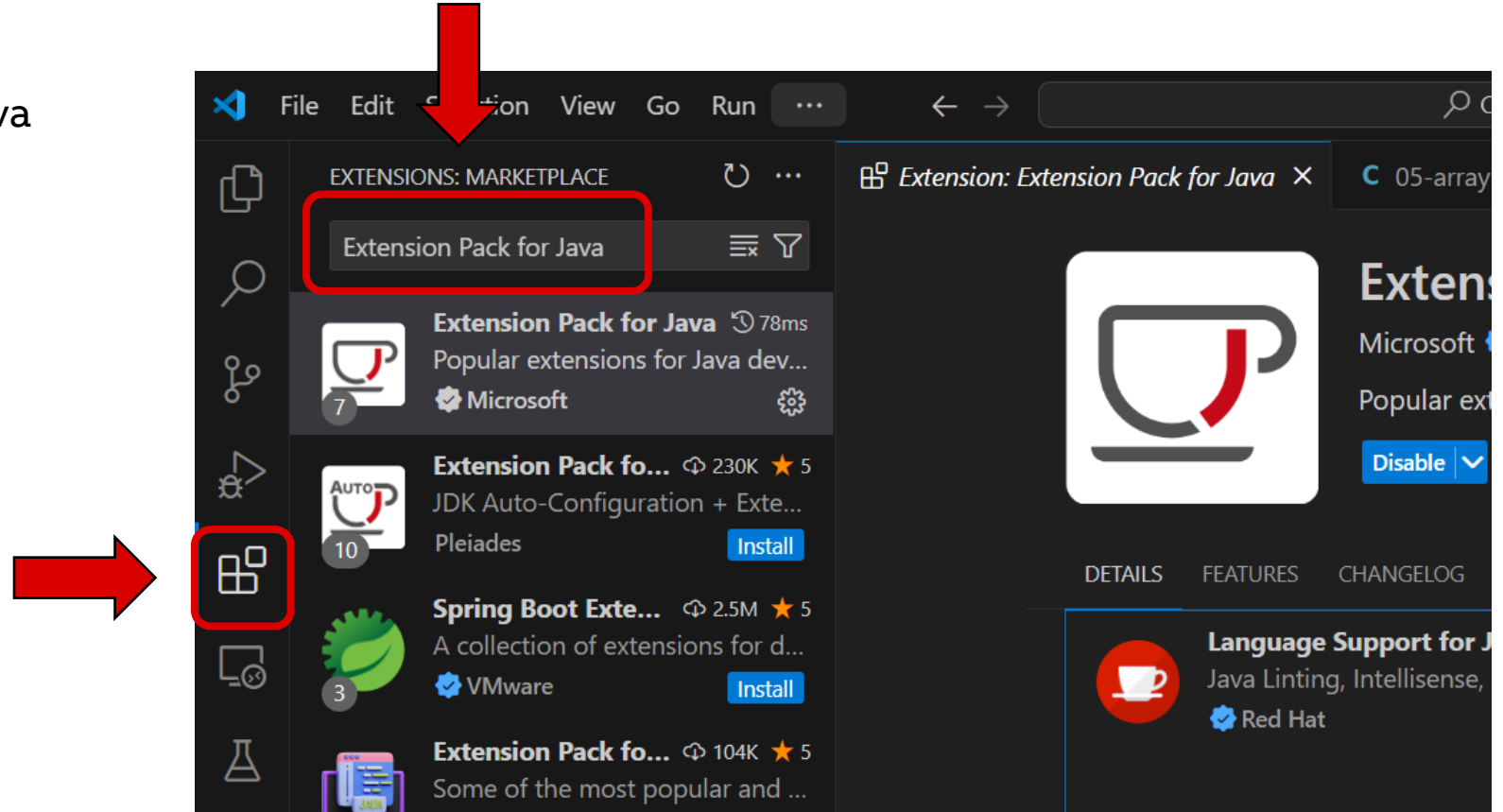
<p>8.0.422+5</p> <p>Temurin </p> <p>July 19, 2024</p>	Windows	x64	<p>JDK - 89 MB</p> <p>Checksum</p> <p></p>
			<p>JDK - 106 MB</p> <p>Checksum</p> <p></p>



Einführung in Java

VS Code Extension Pack for Java

- VS Code
 - Extensions -> Extension Pack for Java installieren



Einführung in Java

HelloWorld.java

- **Dateiendung:** .java
- **Kompilieren**
 - javac HelloWorld.java
- **Ausführen**
 - java HelloWorld
- Klassenname im Code muss denselben Namen wie die Datei haben + Dateiendung

```
// In Java muss jedes Programm in einer Klasse definiert sein
public class HelloWorld {

    // Hauptmethode
    public static void main(String[] args) {

        // Gibt den Text auf dem Bildschirm aus
        System.out.println("Hello, World!");
    }
}
```

Einführung in Java

Unterschiede zu C

Java-Programme:

- Kompilierung in Bytecode (plattformunabhängig)
- Benötigt Java Virtual Machine (JVM) zur Ausführung
- Just-in-Time (JIT)-Kompilierung zur Laufzeit
- Automatische Speicherverwaltung (Garbage Collection)
- Plattformunabhängig: Einmal schreiben, überall ausführen (JVM notwendig)

C-Programme

- Direkte Kompilierung in Maschinencode (plattformabhängig)
- Kein zusätzliches Laufzeitsystem wie JVM notwendig
- Höhere Effizienz, direkte Ausführung auf der Hardware
- Manuelle Speicherverwaltung (malloc/free)
- Plattformabhängig: Erfordert plattformspezifische Kompilierung

Einführung in Java

Benutzereingaben einlesen

- **Scanner**

- Die Klasse Scanner wird verwendet, um Benutzereingaben einzulesen
- Sie ist Teil des Pakets `java.util`, daher muss sie importiert werden.

```
import java.util.Scanner;

public class BenutzerEingabe {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Gib deinen Namen ein: ");
        String name = scanner.nextLine();

        System.out.print("Gib dein Alter ein: ");
        int alter = scanner.nextInt();

        System.out.println("Hallo " + name + ", du
bist " + alter + " Jahre alt.");
    }
}
```

Einführung in Java

Listen in Java

- Gehört zum util package
- Eine der gängigsten Implementierungen von Listen ist die ArrayList
- Wichtige Methoden für Listen
 - add(element): Fügt ein Element zur Liste hinzu
 - get(index): Gibt das Element an der angegebenen Position zurück
 - size(): Gibt die Anzahl der Elemente in der Liste zurück
 - remove(index): Entfernt das Element an der angegebenen Position
- Im Gegensatz zu reinen Arrays passen sich Listen in ihrer Größe automatisch an

```
import java.util.ArrayList;
import java.util.List;

public class ListenBeispiel {
    public static void main(String[] args) {
        List<String> namen = new ArrayList<>();
        namen.add("Anna");
        namen.add("Ben");

        System.out.println(namen.get(0));
        namen.remove(1);

        for (String name : namen) {
            System.out.println(name);
        }
    }
}
```

Einführung in Java

Prozedurale vs. Objektorientierte Programmierung

Prozedurale Programmierung

- Programmablauf ist eine Abfolge von Befehlen
- Funktionen werden zur Ausführung bestimmter Aufgaben aufgerufen
- Daten und Funktionen sind voneinander getrennt

```
void printTime(int hours, int minutes) {  
    System.out.println(hours + ":" +  
minutes);  
}  
  
int hours = 10;  
int minutes = 30;  
printTime(hours, minutes);
```

Objektorientierte Programmierung

- Programm ist eine Sammlung von Objekten
- Objekte repräsentieren Daten und deren Verhalten
- Daten und Methoden sind in Klassen gekapselt.

```
class Clock {  
    int hours = 10;  
    int minutes = 30;  
  
    void printTime() {  
        System.out.println(hours + ":" +  
minutes);  
    }  
}  
  
Clock clock = new Clock();  
clock.printTime();
```

Einführung in Java

Objektorientierte Programmierung (OOP)

Was bedeutet "objektorientiert"?

- In OOP dreht sich alles um **Objekte**.
- Ein Objekt ist eine Einheit, die Daten (Attribute) und Funktionen (Methoden) kombiniert
- Objekte repräsentieren Elemente, wie z. B. ein Auto, einen Benutzer oder auch technisches wie eine Verbindung zu einem Webserver.
- **Klassen** sind die Baupläne für Objekte. Sie definieren, welche Eigenschaften (Attribute) und Verhaltensweisen (Methoden) Objekte haben.

```
public class Auto {  
    String modell;  
    int geschwindigkeit;  
  
    public Auto(String modell) {  
        this.modell = modell;  
        this.geschwindigkeit = 0;  
    }  
  
    public void beschleunigen(int wert) {  
        geschwindigkeit += wert;  
    }  
}
```

```
public class Program {  
    public static void main(String[] args)  
    {  
        Auto meinAuto = new Auto("BMW");  
        meinAuto.beschleunigen(50);  
    }  
}
```

Einführung in Java

Vorteile der Objektorientierung

- **Wiederverwendbarkeit**
 - Code lässt sich durch Klassen und Vererbung leicht wiederverwenden
- **Modularität**
 - Objekte und Klassen sind voneinander getrennt, was die Wartung und das Verständnis vereinfacht
- **Erweiterbarkeit**
 - Neue Funktionen lassen sich leicht durch Vererbung und Polymorphismus hinzufügen
- **Kapselung**
 - Daten und Funktionen sind in Objekten gekapselt, wodurch sie vor ungewolltem Zugriff geschützt werden

```
public class Auto {  
    String modell;  
    int geschwindigkeit;  
  
    public Auto(String modell) {  
        this.modell = modell;  
        this.geschwindigkeit = 0;  
    }  
  
    public void beschleunigen(int wert) {  
        geschwindigkeit += wert;  
    }  
}
```

```
public class Program {  
    public static void main(String[] args)  
    {  
        Auto meinAuto = new Auto("BMW");  
        meinAuto.beschleunigen(50);  
    }  
}
```

Einführung in Java

Konstruktor

Was ist ein Konstruktor?

- Eine spezielle Methode, die beim Erstellen eines Objekts aufgerufen wird
- Hat denselben Namen wie die Klasse
- Kein Rückgabewert, nicht einmal void

Zweck des Konstruktors

- Initialisiert die Attribute des Objekts beim Erstellen
- Wird automatisch aufgerufen, wenn ein neues Objekt erstellt wird

```
public class Auto {  
    String modell;  
  
    // Konstruktor  
    public Auto(String modell) {  
        this.modell = modell;  
    }  
}
```

```
// Erstellen eines Auto-Objekts  
Auto meinAuto = new Auto("BMW");
```


Einführung in Java

Methoden

Was sind Methoden?

- Funktionen, die innerhalb einer Klasse definiert sind und das Verhalten eines Objekts bestimmen

Zweck von Methoden

- Ermöglichen die Manipulation von Objektattributen und führen Aufgaben durch

```
public void beschleunigen(int wert) {  
    this.geschwindigkeit += wert;  
}
```

Einführung in Java

Getter und Setter (Zugriffsmethoden)

- **Getter**


- Methoden, die es ermöglichen, private Attribute von Objekten außerhalb der Klasse sicher abzurufen

- **Setter**

- Methoden, die es ermöglichen, private Attribute sicher zu ändern

- **Zweck**

- Kapselung, um den Zugriff auf Attribute zu kontrollieren und Daten zu schützen



```
public String getModel() {  
    return this.modell;  
}  
  
public void setModell(String modell) {  
    this.modell = modell;  
}
```

Einführung in Java

Zugriffsmodifizierer in Java

- Zugriffsmodifizierer steuern die Sichtbarkeit und den Zugriff auf Klassen, Attribute und Methoden
- Ziel: Kapselung, um die Integrität und Sicherheit der Daten zu schützen
- `public`
 - Zugriff von überall (innerhalb und außerhalb der Klasse und des Pakets)
- `private`
 - Zugriff nur innerhalb derselben Klasse
- `protected`
 - Zugriff innerhalb derselben Klasse, in Unterklassen und innerhalb des gleichen Pakets

```
public int anzahl;  
  
private String name;  
  
protected void setName(String name) {  
    this.name = name;  
}
```

Einführung in Java

Schlüsselwort `this`

- Was ist **`this`**?
 - Verweist auf das aktuelle Objekt der Klasse
 - Wird verwendet, um auf Instanzvariablen und Methoden zuzugreifen
 - Nützlich bei Namenskonflikten zwischen lokalen und Instanzvariablen
- **Instanzvariablen-Zuweisung** mit `this`
 - Im Konstruktor wird `this.name` und `this.alter` verwendet, um die Instanzvariablen des aktuellen Objekts zu referenzieren
- **Konstruktorverkettung** mit `this()`: Der ein-Parameter-Konstruktor ruft den zwei-Parameter-Konstruktor auf

```
public class Hund {  
    private String name;  
    private int alter;  
  
    public Hund(String name) {  
        this(name, 1); // Ruft den Konstruktor mit zwei Parametern auf  
    }  
  
    public Hund(String name, int alter) {  
        this.name = name; // `this` verweist auf die Instanzvariable  
        this.alter = alter;  
    }  
}
```



Aufgabenblatt 3 (Java)

Aufgaben 1 bis 12

<https://github.com/koenig101/vorpraktikum2024>

Vertiefung in Java

Vererbung

- **Was ist Vererbung?**

- Vererbung ist ein Prinzip der Objektorientierung, bei dem eine Klasse Eigenschaften (Attribute) und Verhalten (Methoden) von einer anderen Klasse erbt
- Die übergeordnete Klasse nennt man Oberklasse, Superklasse oder Basisklasse
- Die untergeordnete Klasse nennt man Unterklasse, Subklasse oder abgeleitete Klasse

- **Vorteile**

- Wiederverwendbarkeit: Gemeinsamer Code wird in der Superklasse definiert und von den Subklassen wiederverwendet
- Erweiterbarkeit: Subklassen können die Superklasse erweitern, indem sie neue Methoden und Attribute hinzufügen

Vertiefung in Java

Vererbung

```
// Superklasse
class Tier {
    String name;

    public void schlafen() {
        System.out.println(name + " schläft.");
    }
}
```

```
// Subklasse
class Hund extends Tier {
    public void bellen() {
        System.out.println(name + " bellt: Wuff!");
    }
}
```

```
// Verwendung der Subklasse
public class Main {
    public static void main(String[] args) {
        Hund hund = new Hund();
        hund.name = "Bello";
        hund.schlafen(); // geerbte Methode
        hund.bellen();   // eigene Methode
    }
}
```

Vertiefung in Java

Abstrakte Klassen

– Was sind abstrakte Klassen?

- Eine abstrakte Klasse ist eine Klasse, die nicht instanziiert werden kann
- Sie dient als Vorlage für andere Klassen
- Können sowohl abstrakte Methoden (ohne Implementierung) als auch konkrete Methoden (mit Implementierung) enthalten

– Zweck

- Definiert gemeinsame Eigenschaften und Methoden für alle Unterklassen
- Zwingt Unterklassen, die abstrakten Methoden zu implementieren
- Konkrete Methoden können optional verwendet oder überschrieben werden

```
abstract class Tier {  
    // Abstrakte Methode (ohne Implementierung)  
    abstract void lautGeben();  
  
    // Normale Methode (mit Implementierung)  
    void bewegen() {  
        System.out.println("Tier bewegt sich");  
    }  
}
```

Vertiefung in Java

Abstrakte Klassen

– Vorteile

- Code-Wiederverwendung: Gemeinsame Methoden in der Oberklasse vermeiden doppelten Code
- Flexibilität: Abstrakte Methoden bieten Raum für spezifische Implementierungen
- Struktur: Sicherstellen, dass alle Unterklassen wichtige Methoden implementieren

– Fazit

- Abstrakte Klassen sind ideal, wenn gemeinsame Funktionalität vorhanden ist, aber die genaue Implementierung von Unterklassen abhängt

```
abstract class Tier {  
    abstract void lautGeben();  
}
```

```
class Hund extends Tier {  
    public void lautGeben() {  
        System.out.println("Wuff!");  
    }  
}
```

Vertiefung in Java

Methodenüberschreiben mit @Override

- **Was ist Methoden überschreiben?**
 - Erlaubt es Unterklassen, spezialisierte Implementierungen zu definieren und das Verhalten der geerbten Methode anzupassen
 - Beim Überschreiben einer Methode wird eine Methode der Oberklasse in der Unterklasse neu definiert
 - Die Methode in der Unterklasse hat dieselbe Signatur (Name, Parameter)
 - Mithilfe des @Override-Schlüsselworts wird die Methode in der Unterklasse explizit als Überschreibung markiert

```
// Oberklasse Tier
class Tier {
    void geraeuschiMachen() {
        System.out.println("Das Tier macht ein Geräusch.");
    }
}

// Unterklasse Hund
class Hund extends Tier {
    // Methode wird überschrieben
    @Override
    void geraeuschiMachen() {
        System.out.println("Der Hund bellt: Wuff!");
    }
}
```

Vertiefung in Java

Polymorphismus

– Was ist Polymorphismus

- Fähigkeit von Objekten, je nach Kontext unterschiedliche Formen anzunehmen
- Erlaubt es, dass dieselbe Methode in verschiedenen Klassen unterschiedlich implementiert wird

– Vorteile

- Flexibilität bei der Arbeit mit Objektgruppen
- Neue Klassen können leicht hinzugefügt werden, ohne bestehende Logik zu ändern (Erweiterbarkeit)

– Beispiele

- Die Methode `lautGeben()` verhält sich abhängig vom tatsächlichen Objekttyp
- List Interface von Java

```
class Tier {
    void lautGeben() {
        System.out.println("Das Tier gibt einen Laut.");
    }
}

class Hund extends Tier {
    @Override
    void lautGeben() {
        System.out.println("Wuff!");
    }
}

class Katze extends Tier {
    @Override
    void lautGeben() {
        System.out.println("Miau!");
    }
}
```

Vertiefung in Java

Interfaces

– Interfaces in Java

- Ein Interface legt Methoden fest, die eine Klasse implementieren muss
- Enthält nur Methodendeklarationen, keine Implementierungen
- Eine Klasse kann mehrere Interfaces implementieren
- Interfaces definieren „Was“ eine Klasse tun soll, aber nicht „Wie“ es umgesetzt wird

– Vorteile von Interfaces

- Fördert die Trennung von Schnittstelle und Implementierung
- Ermöglicht flexiblere und erweiterbare Architekturen

```
interface Kaempfer {  
    void angreifen();  
}  
  
class Krieger implements Kaempfer {  
    @Override  
    public void angreifen() {  
        System.out.println("Krieger greift mit Schwert an!");  
    }  
}  
  
class Magier implements Kaempfer {  
    @Override  
    public void angreifen() {  
        System.out.println("Magier zaubert einen Feuerball!");  
    }  
}
```


Vertiefung in Java

Übersicht

- **Vererbung** (Schlüsselwort `extends`)
 - Wiederverwendung von Code, indem Klassen Eigenschaften und Methoden einer übergeordneten Klasse erben
 - Macht den Code effizienter und flexibler
- **Abstrakte Klassen** (Schlüsselwort `abstract class`)
 - Definieren gemeinsame Funktionalitäten und Strukturen für mehrere Klassen
 - Lassen Raum für spezialisierte Implementierungen in abgeleiteten Klassen
- **Polymorphismus**
 - Erlaubt es, verschiedene Objekte über eine gemeinsame Schnittstelle oder Basisklasse gleich zu behandeln
 - Verbessert die Erweiterbarkeit und Flexibilität des Codes
- **Interfaces** (Schlüsselwort `implements`)
 - Trennen die Definition von Funktionalitäten von ihrer Implementierung
 - Fördern lose Kopplung, was den Code modularer und wartbarer macht



Aufgabenblatt 4 (Java)

Aufgaben 1 bis 12

<https://github.com/koenig101/vorpraktikum2024>

ABB



Pause

bis 13:00

