

Java Programmierung – Leitfaden mit Beispiel: Autovermietung

Ein kompakter und verständlicher Leitfaden zur objektorientierten Programmierung in Java mit einem vollständigen Praxisbeispiel (Autovermietung).


Inhalt

-  Java Programmierung – Leitfaden mit Beispiel: Autovermietung
 -  Inhalt
 -  Java IDEs – Übersicht
 -  Projekt erstellen in IntelliJ IDEA
 - Grundlagen: Klassen & Objekte
 -  Lernpfad: Vom einfachen Auto zur vollständigen Vermietung
 - Einfaches Auto
 - Mit Konstruktor und Methoden
 - Zugriffsmodifizierer in Java
 - UML-Konventionen
 - Abstrakte Klassen vs Interfaces
 - Abstrakte Klasse
 - Interface
 -  Nutzung abstrakter Klasse und Interface
 - Java Namenskonventionen
 -  Java Schlüsselwörter: `implements`, `extends`, `@Override`
 - `extends`
 - `implements`
 - `@Override`
 - UML-Diagramm zur Autovermietung
 - Java-Code – Autovermietung
 - OOP-Konzepte im Beispiel
 - Erweiterungsideen

Java IDEs – Übersicht

IDE	Vorteile
IntelliJ IDEA	Sehr leistungsstark, intelligente Vorschläge, Debugger, GUI-Designer
Eclipse	Open Source, viele Plugins
VS Code	Leichtgewichtig mit Java Plugin

Projekt erstellen in IntelliJ IDEA

1. Öffne **IntelliJ IDEA**
2. Klicke auf "**New Project**"
3. Wähle "**Java**" und SDK (z.B. Java 17)
4. Projektname: **Autovermietung**
5. Verzeichnis wählen und **Finish** klicken
6. Unter **src** Datei **Autovermietung.java** anlegen
7. Code einfügen und mit  ausführen

Grundlagen: Klassen & Objekte

- Eine **Klasse** ist ein Bauplan für Objekte.
- Ein **Objekt** ist eine konkrete Instanz dieser Klasse.

```
// Beispiel einer einfachen Klasse mit Konstruktor und Methode
public class Auto {
    private String marke;

    public Auto(String marke) {
        this.marke = marke;
    }

    public String getMarke() {
        return marke;
    }
}
```

Lernpfad: Vom einfachen Auto zur vollständigen Vermietung

Einfaches Auto

```
public class Auto {
    String marke;
    int ps;

    public void starten() {
        System.out.println(marke + " startet mit " + ps + " PS.");
    }
}
```

Mit Konstruktor und Methoden

```
public class Auto {
    String marke;
    int ps;
```

```

public Auto(String marke, int ps) {
    this.marke = marke;
    this.ps = ps;
}

public void starten() {
    System.out.println(marke + " startet.");
}
}

```

Zugriffsmodifizierer in Java

Modifizierer	Bedeutung
public	Überall sichtbar
private	Nur innerhalb der Klasse sichtbar
protected	Für Unterklassen & im Paket sichtbar

UML-Konventionen

Symbol	Bedeutung
+	public
-	private
#	protected

Abstrakte Klassen vs Interfaces

Abstrakte Klasse

- Kann Attribute & Methoden enthalten
- Kann Methoden **mit** und **ohne** Implementierung enthalten
- Kann nicht direkt instanziiert werden

```

abstract class Tier {
    public abstract void gibLaut();

    public void atmen() {
        System.out.println("Tier atmet");
    }
}

```

Interface

- Enthält nur Methodensignaturen (abstrakte Methoden)
- Wird mit `implements` eingebunden
- Alle Methoden sind `public abstract`

```
interface Fahrzeug {  
    String getMarke();  
    double getTagespreis();  
}
```

3 Nutzung abstrakter Klasse und Interface

- Interface: `Fahrzeug`
- Abstrakte Klasse: `AbstraktesFahrzeug`
- Konkrete Klasse: `Auto` erbt und überschreibt Methoden

Java Namenskonventionen

Typ	Beispiel
Klasse	<code>Auto</code> , <code>Kunde</code>
Methode	<code>getName()</code>
Variable	<code>anzSitze</code> , <code>preis</code>
Konstante	<code>MAX_SPEED</code>

🤖 Java Schlüsselwörter: `implements`, `extends`, `@Override`

`extends`

Wird verwendet, wenn eine Klasse **von einer anderen Klasse erbt** (Vererbung).

```
class Auto extends Fahrzeug {  
    // Auto übernimmt Eigenschaften von Fahrzeug  
}
```

`implements`

Wird verwendet, um ein **Interface zu implementieren**.

```
class Auto implements Fahrbar {  
    public void fahren() {  
        System.out.println("Auto fährt");  
    }  
}
```

```
}  
}
```

@Override

Diese Annotation zeigt an, dass eine Methode **überschrieben** wird – also in einer Unterklasse neu definiert wird.

```
@Override  
public void fahren() {  
    System.out.println("Auto fährt schneller");  
}
```

UML-Diagramm zur Autovermietung



```

+-----+
| Kunde          |
+-----+
| -name: String  |
| -fuehrerscheinNr: String
+-----+
| +getName()     |
| +getFuehrerscheinNr()
+-----+

+-----+
| Vermietung     |
+-----+
| -kunde: Kunde  |
| -fahrzeug: Fahrzeug
| -tage: int     |
+-----+
| +berechnePreis():double
| +zeigeDetails():void
+-----+

```

Java-Code – Autovermietung

```

// Autovermietung.java

import java.util.*;

interface Fahrzeug {
    String getMarke();
    String getModell();
    double getTagespreis();
}

abstract class AbstraktesFahrzeug implements Fahrzeug {
    protected String marke;
    protected String modell;
    protected double tagespreis;

    public AbstraktesFahrzeug(String marke, String modell, double tagespreis) {
        this.marke = marke;
        this.modell = modell;
        this.tagespreis = tagespreis;
    }

    public String getMarke() { return marke; }
    public String getModell() { return modell; }
    public double getTagespreis() { return tagespreis; }
}

class Auto extends AbstraktesFahrzeug {

```

```

        private int anzSitze;

        public Auto(String marke, String modell, double tagespreis, int anzSitze) {
            super(marke, modell, tagespreis);
            this.anzSitze = anzSitze;
        }

        public int getSitze() {
            return anzSitze;
        }
    }

    class Transporter extends AbstraktesFahrzeug {
        private int ladeVolumen;

        public Transporter(String marke, String modell, double tagespreis, int
ladeVolumen) {
            super(marke, modell, tagespreis);
            this.ladeVolumen = ladeVolumen;
        }

        public int getLadeVolumen() {
            return ladeVolumen;
        }
    }

    class Kunde {
        private String name;
        private String fuehrerscheinNr;

        public Kunde(String name, String fuehrerscheinNr) {
            this.name = name;
            this.fuehrerscheinNr = fuehrerscheinNr;
        }

        public String getName() {
            return name;
        }

        public String getFuehrerscheinNr() {
            return fuehrerscheinNr;
        }
    }

    class Vermietung {
        private Kunde kunde;
        private Fahrzeug fahrzeug;
        private int tage;

        public Vermietung(Kunde kunde, Fahrzeug fahrzeug, int tage) {
            this.kunde = kunde;
            this.fahrzeug = fahrzeug;
            this.tage = tage;
        }
    }

```

```

    public double berechnePreis() {
        return fahrzeug.getTagespreis() * tage;
    }

    public void zeigeDetails() {
        System.out.println("=== Mietvertrag ===");
        System.out.println("Kunde: " + kunde.getName());
        System.out.println("Fahrzeug: " + fahrzeug.getMarke() + " " +
fahrzeug.getModell());
        System.out.println("Tage: " + tage);
        System.out.println("Gesamtpreis: " + berechnePreis() + " EUR");
        System.out.println();
    }
}

public class Autovermietung {
    public static void main(String[] args) {
        Kunde kunde1 = new Kunde("Max Mustermann", "B1234567");
        Auto auto1 = new Auto("VW", "Golf", 45.0, 5);
        Transporter transp1 = new Transporter("Mercedes", "Sprinter", 80.0, 10);

        Vermietung miete1 = new Vermietung(kunde1, auto1, 3);
        Vermietung miete2 = new Vermietung(kunde1, transp1, 2);




        miete1.zeigeDetails();
        miete2.zeigeDetails();
    }
}


```

OOP-Konzepte im Beispiel

Konzept	Umsetzung im Code
Klasse	<code>Auto</code> , <code>Kunde</code> , <code>Vermietung</code>
Abstrakte Klasse	<code>AbstraktesFahrzeug</code>
Interface	<code>Fahrzeug</code>
Sichtbarkeit	<code>private</code> , <code>public</code> , <code>protected</code>
Polymorphismus	<code>Fahrzeug</code> als Typ für verschiedene Klassen
Vererbung	<code>Auto</code> , <code>Transporter</code> erweitern <code>AbstraktesFahrzeug</code>

Erweiterungsideen

-  Fahrzeugverwaltung über Listen (`List<Fahrzeug>`)
-  Speicherung von Mietdaten in Dateien/Datenbanken
-  GUI mit JavaFX oder Swing

-  Erweiterung mit Versicherungen, Rabatten, Rückgabeprotokollen
-