



# Objektorientierte Systementwicklung

## Vom Mockup zum langlebigen Softwaresystem

### Oberfläche analysieren · Modell entwerfen · Implementierung testen

BKWI2 und Wirtschaftsinformatik am beruflichen Gymnasium · Kapitel 1 bis 14  
Didaktisch erweiterte Kursfassung · A5 · Schuljahr 2026/2027 · v1.2 · Serienmaster R7

**Christine Janischek · Lizenz Custom License NC School-Only 1.0 ·  
[emotionalspirit.de](https://emotionalspirit.de)**

*Dieses Lernskript wurde mit Unterstützung einer KI erstellt und anschließend redaktionell geprüft.  
Trotz sorgfältiger Bearbeitung können Fehler enthalten sein. Bitte melde Fehler und  
Verbesserungsvorschläge über die angegebene Kontaktmöglichkeit.*

## Inhaltsverzeichnis

Die Einträge führen direkt zu den Kapiteln und Hauptabschnitten. In den Überschriften bringt dich „↔ Inhalt“ hierher zurück.

- 01 Warum objektorientierte Softwareentwicklung?**
- 02 Vom Mockup zum Objektmodell**
- 03 Klassen, Objekte und geschützter Zustand**
- 04 Kontrollstrukturen gehören zu Verantwortlichkeiten**
- 05 Methoden, Verträge und automatisierte Tests**

- 06** Listen, Schleifen und Objektmengen
- 07** Objektbeziehungen bewusst modellieren
- 08** Abstraktion, Interfaces und Polymorphie
- 09** MVC: Oberfläche und Fachmodell trennen
- 10** Daten verwalten und persistieren
- 11** Datenbanken und Services anbinden
- 12** Sichere Software und Datensicherheit
- 13** Softwarequalität, Refactoring und Erweiterung
- 14** Abschlussprojekt: ein langlebiges Softwaresystem
- 15** Quellen, Lizenz und Prüfnachweis

## Zwei Bildungsgänge - ein gemeinsames OOP-Fundament

### **BG-PFAD · 50 UNTERRICHTSSTUNDEN + LEISTUNGSFESTSTELLUNG**

Verbindlicher Kern: Kapitel 1–9. Vertiefungen aus Kapitel 10–14 werden nur gewählt, wenn Zeit und Lerngruppe es erlauben.

Lehrplananker: BPE 5.1 Klassen und Objekte · BPE 5.2 Kontrollstrukturen und Ausnahmen · BPE 5.3 Vererbung und Beziehungen · BPE 5.4 GUI-Projekt und Zwei-Schichten-Architektur.

### **BKWI2-PFAD · 135 UNTERRICHTSSTUNDEN**

Vollständiger Durchlauf Kapitel 1–14 mit Java als Implementierungspfad. Python und PHP dienen an ausgewählten Stellen dem Prinzipitransfer.

Lehrplananker: BPE 8.1 objektorientierter Systementwurf · BPE 8.2 grafische Oberflächen · BPE 8.3 Schichten, Persistenz, Datenbank und Sicherheit.

### **KOMPETENZNACHWEIS**

Jede bewertbare Aufgabe trägt eine eindeutige Kennung.

Abgegeben werden beobachtbare Lernprodukte: Modell, Quellcode, Testfälle, Ausführungsergebnis und kurze Begründung. Die Nummerierung innerhalb eines Aufgabenblocks beginnt bewusst wieder bei 1; die blockübergreifende Zuordnung erfolgt ausschließlich über die Aufgabenkennung.

## So arbeitest du mit diesem Skript

**OFFLINE-MATERIAL** Zu jeder Aufgaben-ID liegt im Schülerpaket ein Arbeitsblatt. Praktische Kapitel verwenden eines der zwölf Starterprojekte. Die Aufgaben-Material-Lösungsmatrix befindet sich in der Dokumentation; Lösungen, Erwartungshorizonte und Punkte bleiben ausschließlich im Lehrerpaket.

Jedes Kapitel beginnt mit einem sichtbaren Problem oder Mockup. Du untersuchst zunächst, was das System leisten soll. Erst danach modellierst du Objekte, verteilst Verantwortlichkeiten und setzt die Lösung in Java um.

**KERNPFAD** Diese Abschnitte sichern die gemeinsamen OOP-Grundlagen und eignen sich auch für den zweistündigen BG-Pfad.

**BKW12-AUFBAU** Diese Aufgaben vertiefen Modellierung, Implementierung, Test und Softwarequalität im sechstündigen Unterricht.

**KI-LERNCOACH** Die Prompts helfen dir beim Denken. Lass dir keine fertige Lösung erzeugen, die du nicht erklären, testen und verändern kannst.

## Sprachenkonzept: ein Modell - drei Ausdrucksformen

Java ist die verbindliche Arbeits- und Prüfungssprache. Python und PHP werden nicht als zusätzliche vollständige Sprachkurse behandelt. Kurze Transferfenster machen sichtbar, welche Entwurfsentscheidung erhalten bleibt, obwohl sich Syntax, Typsystem und Laufzeitumgebung unterscheiden.

| Sprache | Rolle                   | Umfang                                      | Leitfrage                                         |
|---------|-------------------------|---------------------------------------------|---------------------------------------------------|
| Java    | verbindlicher Hauptpfad | vollständige Modelle, Anwendungen und Tests | Wie wird der Entwurf robust umgesetzt?            |
| Python  | kompakter Transfer      | kurze Parallelbeispiele und Wahlaufgaben    | Was bleibt trotz weniger Syntax gleich?           |
| PHP     | Webtransfer             | kurze Parallelbeispiele und Webkontexte     | Wie lebt das Modell in einer Webanwendung weiter? |

**DIDAKTISCHE GRENZE** Ein Dreisprachenfenster erscheint nur dann, wenn es ein OOP-Prinzip schärft. Reine Syntaxdetails werden nicht dreifach wiederholt.

## Lernroute dieser Masterfassung

| Kapitel | Leitfrage                           | Lernprodukt               | Projektkontexte                |
|---------|-------------------------------------|---------------------------|--------------------------------|
| 1       | Warum OOP?                          | Qualitätslandkarte        | Lernagent, BMI-App             |
| 2       | Wie wird aus einer Idee ein Modell? | Mockup + Objektkandidaten | Parkautomat                    |
| 3       | Wie entstehen Klassen und Objekte?  | UML + Java-Modellklasse   | Vereinsmitglied                |
| 4       | Wo gehört Entscheidungslogik hin?   | drei Automatenmodelle     | Milch-, Park-, Getränkeautomat |
| 5       | Wie werden Regeln überprüfbar?      | Verträge + Tests          | BMI-App                        |

| Kapitel | Leitfrage                                  | Lernprodukt                       | Projektkontexte        |
|---------|--------------------------------------------|-----------------------------------|------------------------|
| 6       | Wie verarbeitet ein Modell Mengen?         | gefilterte Objektliste            | Volleyball-Teammanager |
| 7       | Wie arbeiten Objekte zusammen?             | Beziehungsmodell I                | Verein, Mini-Shop      |
| 8       | Wie bleibt Verhalten austauschbar?         | Strategie-Schnittstelle           | Feedback, Lernagent    |
| 9       | Wie bleibt die Oberfläche austauschbar?    | MVC-Prototyp                      | BMI-App                |
| 10      | Wie bleiben Objekte dauerhaft erhalten?    | Speicheradapter                   | Volleyball-Teammanager |
| 11      | Wie werden externe Systeme angebunden?     | Repository/Service-Adapter        | Verein, Lernagent      |
| 12      | Wie wird Software sicher?                  | Sicherheitskonzept + Negativtests | Lernagent              |
| 13      | Wie wird Fremdcode wartbarer?              | Refactoring-Nachweis              | Lernagent              |
| 14      | Wie entsteht ein langlebiges Gesamtsystem? | Abschlussinkrement                | Wahlprojekt            |

## 1 · Warum objektorientierte Softwareentwicklung? [↔ Inhalt](#)

**DEIN KAPITELSTART** Du kannst erklären, warum gute Software mehr leisten muss als nur ein korrektes Ergebnis auszugeben.

### 1.1 Aktivierung: Funktioniert - aber ist es auch gute Software?

Ein Lernagent wurde schnell programmiert. Alle Eingaben, Berechnungen, Datenbankzugriffe und Ausgaben stehen in einer einzigen Datei. Heute funktioniert das Programm. Morgen sollen ein

zweiter Lernpfad, eine mobile Oberfläche und eine neue Datenquelle ergänzt werden.

### **[OOP-K01-A01] · KERNPFAD · Die Änderungsprobe**

1. Markiere gedanklich, welche Stellen beim Ergänzen einer mobilen Oberfläche verändert werden müssten.
2. Beschreibe zwei Risiken, wenn dieselbe Berechnungsregel an mehreren Stellen vorkommt.
3. Formuliere drei Qualitätsziele für die nächste Version des Systems.

**Erwartetes Lernprodukt:** Eine begründete Qualitätslandkarte mit Wartbarkeit, Erweiterbarkeit und Redundanzvermeidung.

**Möglicher Startprompt:** Stelle mir nacheinander Fragen, mit denen ich die Änderungsrisiken einer monolithischen Anwendung selbst erkenne. Nenne die Qualitätsziele nicht vorab.

#### **Gestufte Tutorhilfe**

Hilfe 1: Was geschieht, wenn eine Regel nur an zwei von drei Stellen geändert wird?

Hilfe 2: Welche Teile ändern sich häufig – Oberfläche, Fachlogik oder Datenhaltung?

## **1.2 Java heute: Programme aus verständlichen Bausteinen**

Viele Apps bestehen aus mehreren Bausteinen. Der Teil auf dem Bildschirm ist nur die sichtbare Seite. Im Hintergrund laden andere Programmteile Daten, prüfen Regeln oder berechnen ein Ergebnis. Damit die Bausteine zuverlässig zusammenarbeiten, benötigen sie klar vereinbarte Übergabepunkte.

**ALLTAGSBILD: DIE SCHULMENSA** Stell dir eine Schulmensa vor: Am Bestellbildschirm wählst du dein Essen aus. Das ist wie die Benutzeroberfläche einer App. Die Küche arbeitet für dich unsichtbar im Hintergrund – ähnlich wie ein Backend. Eine einzelne Station, zum Beispiel die

Nudelstation, übernimmt eine klar abgegrenzte Aufgabe. Sie lässt sich mit einem Service vergleichen. Der Bestellzettel legt fest, welche Angaben die Küche erhält und welches Essen zurückgegeben wird. Er funktioniert ähnlich wie eine Schnittstelle. Eine vorbereitete Küche mit Arbeitsflächen, Geräten und festen Abläufen erinnert an ein Framework: Vieles ist schon vorbereitet, das konkrete Gericht musst du trotzdem selbst planen. Das Bild hilft beim Einstieg; ein echtes Softwaresystem ist natürlich genauer geregelt.

## Mini-Glossar für den Praxisblick

| Begriff                         | Einfach erklärt                                                                                                                           |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Benutzeroberfläche (GUI)</b> | Der sichtbare Teil einer App, den Menschen bedienen, zum Beispiel ein Eingabefeld oder ein Button. GUI ist die englische Abkürzung dafür. |
| <b>Backend</b>                  | Der meist unsichtbare Teil im Hintergrund. Er verarbeitet Eingaben, prüft Regeln und greift auf Daten zu.                                 |
| <b>Service / Dienst</b>         | Ein Programmbaustein mit einer klar abgegrenzten Aufgabe. Er kann Teil einer App sein oder als eigenes Programm laufen.                   |
| <b>Schnittstelle</b>            | Ein vereinbarter Übergabepunkt. Er legt fest, welche Daten hineingehen und welches Ergebnis oder welche Fehlermeldung zurückkommt.        |
| <b>API</b>                      | Eine dokumentierte Schnittstelle, über die Programme miteinander kommunizieren können.                                                    |
| <b>Modul</b>                    | Ein abgegrenzter Programmbereich mit einer bestimmten Verantwortung.                                                                      |
| <b>Fachmodell</b>               | Eine vereinfachte Abbildung des Anwendungsbereichs: Welche Dinge, Daten und Regeln benötigt das Programm?                                 |

| Begriff            | Einfach erklärt                                                                                                                                        |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Framework</b>   | Ein vorbereitetes Software-Gerüst mit Werkzeugen und Regeln. Es nimmt Routinearbeit ab, ersetzt aber nicht das eigene Fachmodell.                      |
| <b>Spring Boot</b> | Ein häufig genutztes Java-Framework für Programme im Hintergrund und Dienste für Webseiten. Es ist ein Werkzeug – nicht die Objektorientierung selbst. |
| <b>LTS</b>         | Abkürzung für Long-Term Support, auf Deutsch Langzeit-Unterstützung: Diese Version erhält über längere Zeit Aktualisierungen.                          |
| <b>OpenJDK</b>     | Eine freie, offen entwickelte Ausgabe der Werkzeuge, mit denen Java-Programme erstellt und ausgeführt werden.                                          |

**PRAXISBLICK** Java 25 ist eine LTS-Version. Für unseren Unterricht bedeutet das: Wir nutzen eine aktuelle freie OpenJDK-Ausgabe und lernen Grundlagen, die lange nützlich bleiben. In der Praxis arbeitet Java häufig im Hintergrund von Apps und Webseiten. Ein Modul nimmt beispielsweise eine Anfrage entgegen, ein anderes prüft eine Geschäftsregel – zum Beispiel, ob ein Rabatt gilt – und ein drittes speichert Daten. Über Schnittstellen tauschen diese Bausteine genau festgelegte Informationen aus.

**BEST PRACTICE** Beginne mit kleinen, verständlichen Klassen und eindeutigen Aufgaben. Lege zuerst fest, welches Objekt wofür verantwortlich ist und wie die Objekte zusammenarbeiten. Ein Framework wie Spring Boot kommt später hinzu und übernimmt technische Routineaufgaben. So bleibt der fachliche Kern lesbar, prüfbar und leichter erweiterbar.

**PERSPEKTIVE** Python eignet sich unter anderem für Datenanalyse und KI-Anwendungen; PHP wird häufig für Webseiten eingesetzt. Java macht Datentypen und Vereinbarungen zwischen Programmbausteinen besonders

deutlich sichtbar. In allen drei Sprachen gilt: Aufgaben sinnvoll verteilen, Abhängigkeiten begrenzen und Daten vor unkontrollierten Änderungen schützen.

### **[OOP-K01-A04] · KERNPFAD · Java-Praxis einordnen**

1. Nutze das Mini-Glossar. Ordne die Beispiele Backend-Service, Desktop-Oberfläche (GUI), Datenanalyse und PHP-Webseite einer wahrscheinlichen Aufgabe im Gesamtsystem zu.
2. Begründe, weshalb dieselbe Fachlogik nicht für jede Oberfläche neu programmiert werden sollte.
3. Formuliere zwei Regeln, nach denen ein Framework erst nach dem Fachmodell ausgewählt wird.

Erwartetes Lernprodukt: Eine kurze Praxislandkarte mit vier Einsatzkontexten, zwei Architekturregeln und einer begründeten Erweiterungsprobe.

Prüfkriterien: Java-Rolle fachlich korrekt · Sprache und Framework unterschieden · mindestens eine Service-Schnittstelle benannt · OOP-Nutzen an einer Änderung begründet.

Möglicher Startprompt: Frage mich zuerst, welche Teile einer Anwendung fachlich und welche technisch sind. Hilf mir, Java, Python und PHP als unterschiedliche Ausdrucks- und Einsatzformen desselben Objektmodells einzuordnen.

## **1.3 OOP als Denk- und Ordnungsprinzip**

Objektorientierung richtet Software an fachlich bedeutsamen Dingen, Rollen und Vorgängen aus. Ein Objekt besitzt einen Zustand und übernimmt dazu passende Verantwortung. Klassen beschreiben gemeinsame Strukturen und Verhaltensweisen ähnlicher Objekte.

- Abstraktion: Wesentliches auswählen und Unwesentliches zunächst ausblenden.
- Kapselung: Zustand schützen und nur über kontrollierte Operationen verändern.

- Modularität: Verantwortlichkeiten in verständliche, austauschbare Bausteine gliedern.
- Wiederverwendung: Bewährte Bausteine in neuen Zusammenhängen einsetzen.
- Polymorphie: Verschiedene Objekte über gemeinsame Verträge einheitlich ansprechen.

**VERBINDLICHE IMPLEMENTIERUNGSRoutine** Jedes zentrale OOP-Prinzip wird in Java selbst umgesetzt: Syntax lesen, Grundgerüst vervollständigen, Verhalten testen, eine Anforderung ergänzen und anschließend fremden oder KI-generierten Code anhand des Prinzips beurteilen.

**MERKSATZ** OOP vermeidet Redundanz nicht automatisch. Erst gut geschnittene Verantwortlichkeiten, klare Schnittstellen und bewusst gewählte Abstraktionen machen ein System langfristig wartbar.

## 1.4 Qualität sichtbar machen

### [OOP-K01-A02] · BKWI2-AUFBAU · Vorher-Nachher-Denken

1. Wähle BMI-App, Lernagent oder Vereinsverwaltung.
2. Notiere eine wahrscheinliche spätere Änderung.
3. Skizziere eine ungegliederte Lösung und eine Lösung mit mindestens drei Verantwortungsbereichen.
4. Begründe, welche Lösung sich sicherer erweitern lässt.

**Erwartetes Lernprodukt:** Eine Änderungsgeschichte mit Architekturbegründung.

**Möglicher Startprompt:** Prüfe meine Aufteilung kritisch. Frage jeweils: Wer sollte für diese Information oder Aktion verantwortlich sein? Gib keine fertige Klassenliste vor.

## 1.5 Transfer zum Lernagenten

**WARUM BRAUCHST DU DAS SPÄTER?** Ein persönlicher Lernagent wächst: Neue Fächer, Aufgabentypen, Rückmeldungen und Lernstrategien kommen hinzu. Ohne getrennte Verantwortlichkeiten würden Änderungen an einer Stelle unerwartet andere Funktionen beschädigen.

### [OOP-K01-A03] · TRANSFER LERNAGENT · Architekturkeim des Lernagenten

1. Sammle fünf Aufgaben, die ein Lernagent übernehmen soll.
2. Bündele zusammengehörige Aufgaben zu Verantwortungsbereichen.
3. Gib jedem Bereich einen vorläufigen Namen.
4. Markiere, welche Bereiche sich wahrscheinlich häufig ändern.

**Erwartetes Lernprodukt:** Eine erste Komponentenkarte, noch ohne Quellcode.

**Möglicher Startprompt:** Stelle mir Fragen zu den Verantwortlichkeiten meines Lernagenten. Hilf mir beim Sortieren, aber benenne die Komponenten erst, nachdem ich eigene Vorschläge gemacht habe.

## 1.6 Sprachtransfer: Prinzip vor Syntax

### Java - Typen und Sichtbarkeit werden ausdrücklich festgelegt

```
// Java: Zustand und Verhalten in einer Klasse
public class Mitglied {
    private String name;
    public String getName() { return name; }
}
```

## Python - weniger syntaktische Vorgaben, gleiche Modellierungsfrage

```
# Python: dasselbe Grundprinzip, kompakter notiert
class Mitglied:
    def __init__(self, name: str):
        self.name = name
```

## PHP - typisierte Modellklasse, unabhängig von HTML und Ausgabe

```
<?php
// PHP: dasselbe Modell in einer Websprache
final class Mitglied {
    public function __construct(private string $name) {}
    public function getName(): string { return $this->name; }
}
```

**ENTSCHEIDENDE FRAGE** Welche Verantwortung übernimmt ein Objekt – und welche Information benötigt es dafür?

## 1.7 Kapitel-Sicherung

HALTE FEST · Qualität und Verantwortlichkeiten. Zeige an einer Änderung, welche Klasse verantwortlich bleibt.

MINI-CHECK

1. Erkläre den Kernbegriff in höchstens zwei Sätzen.
2. Zeige ihn an einem konkreten Code-, Modell- oder Testbeleg.
3. Nenne eine typische Fehlentscheidung und ihre erkennbare Folge.

GESTUFTE HILFE

Denkimpuls: Welche fachliche Verantwortung steht im Mittelpunkt?  
Konkreter Hinweis: Zeige an einer Änderung, welche Klasse verantwortlich bleibt.

Teilstruktur: Ausgangslage → Entwurfsentscheidung → Implementierung/Modell → Test → Änderungsfolge.

[Zurück zum Inhaltsverzeichnis](#)

## 2 · Vom Mockup zum Objektmodell [← Inhalt](#)

**DEIN KAPITELSTART** Du kannst aus einer Benutzeroberfläche Anforderungen, fachliche Begriffe und erste Objektkandidaten ableiten, ohne die Oberfläche mit dem Fachmodell zu verwechseln.

### AKTIVIERUNGS-MOCKUP · Parkautomat

Abb.: Grafischer Oberflächenentwurf als Ausgangspunkt für Anforderungsanalyse und Objektmodellierung.

### 2.1 Das Mockup ist eine Gesprächsgrundlage

Ein Mockup macht Anforderungen sichtbar: Welche Eingaben gibt es? Welche Aktionen löst der Nutzer aus? Welche Ergebnisse und Fehlermeldungen werden erwartet? Es zeigt jedoch noch nicht automatisch, welche Klassen später sinnvoll sind.

#### [OOP-K02-A01] · KERNPFAD · Mockup lesen

1. Kennzeichne Eingaben, Aktionen und Ausgaben des Parkautomaten.
2. Formuliere für jede Aktion einen Anwendungsfall als Verb-Objekt-Kombination.

3. Notiere fachliche Nomen als Objektkandidaten.
4. Streiche reine Oberflächenelemente, die keine fachlichen Objekte sind.

**Erwartetes Lernprodukt:** Eine annotierte Mockup-Analyse und eine bereinigte Liste von Objektkandidaten.

**Möglicher Startprompt:** Führe mit mir eine Substantiv- und Verbanalyse durch. Frage bei jedem Kandidaten, ob er fachliche Verantwortung oder nur eine Darstellungsfunktion besitzt.

### **Gestufte Tutorhilfe**

Hilfe 1: Button ist meist kein Fachobjekt.

Hilfe 2: Parkvorgang, Tarif und Ticket verändern oder bewahren fachliche Informationen.

## **2.2 Vom Anwendungsfall zur Verantwortung**

| Anwendungsfall  | Benötigte Information | Mögliche Verantwortung            |
|-----------------|-----------------------|-----------------------------------|
| Preis berechnen | Dauer, Tarif          | Parkvorgang berechnet Preis       |
| Ticket lösen    | Kennzeichen, Endzeit  | Ticket hält Berechtigung fest     |
| Eingabe prüfen  | Dauer, Kennzeichen    | Validierung schützt Systemgrenzen |

**DENKREGEL** Verteile Verhalten möglichst an das Objekt, das die dafür notwendigen Informationen besitzt. So bleibt Fachlogik dort, wo sie verstanden und getestet werden kann.

## **2.3 Analyse - Design - Programmierung**

**OOA · WAS?** Anforderungen, Akteure, Anwendungsfälle und fachliche Objekte verstehen.

**OOD · WIE GEGLIEDERT?** Klassen, Beziehungen, Verantwortlichkeiten und Schnittstellen entwerfen.

**OOP · WIE UMGESETZT?** Das Modell in Java implementieren, testen und schrittweise verbessern.

### **[OOP-K02-A02] · BKWI2-AUFBAU · Vom Wireframe zum ersten Klassendiagramm**

1. Erstelle für Parkvorgang, Tarif und Ticket je eine kurze Verantwortungsbeschreibung.
2. Ordne Attribute und Methoden zu.
3. Zeichne ein UML-Klassendiagramm mit Sichtbarkeit, Datentypen und Beziehungen.
4. Prüfe, ob dieselbe Information unnötig in mehreren Klassen gespeichert wird.

**Erwartetes Lernprodukt:** Ein begründetes UML-Klassendiagramm als Entwurf, noch ohne GUI-Code.

## 2.4 Transfer zum Lernagenten

### AKTIVIERUNGS-MOCKUP · Mein Lernagent

Abb.: Grafischer Oberflächenentwurf als Ausgangspunkt für Anforderungsanalyse und Objektmodellierung.

#### [OOP-K02-A03] · TRANSFER LERNAGENT · Vom Lernagent-Mockup zum Fachmodell

1. Leite Eingaben, Aktionen und Ausgaben aus dem Mockup ab.
2. Unterscheide UI-Elemente von fachlichen Begriffen.
3. Prüfe die Objektkandidaten Lernziel, Lernstand, Aufgabe, Hinweis und Lernschritt.
4. Formuliere zwei Anwendungsfälle und ordne erste Verantwortlichkeiten zu.

**Erwartetes Lernprodukt:** Eine zweite Version der Komponentenkarte mit Objektkandidaten und Anwendungsfällen.

## 2.5 Kapitel-Sicherung

HALTE FEST · Mockup, Anwendungsfall und Objektmodell. Trenne sichtbare Bedienelemente von fachlichen Objekten.

### MINI-CHECK

1. Erkläre den Kernbegriff in höchstens zwei Sätzen.
2. Zeige ihn an einem konkreten Code-, Modell- oder Testbeleg.
3. Nenne eine typische Fehlentscheidung und ihre erkennbare Folge.

### GESTUFTE HILFE

Denkimpuls: Welche fachliche Verantwortung steht im Mittelpunkt?  
Konkreter Hinweis: Trenne sichtbare Bedienelemente von fachlichen Objekten.

Teilstruktur: Ausgangslage → Entwurfsentscheidung → Implementierung/Modell → Test → Änderungsfolge.

[Zurück zum Inhaltsverzeichnis](#)

## 3 · Klassen, Objekte und geschützter Zustand ↪ [Inhalt](#)

**DEIN KAPITELSTART** Du kannst aus einem Objektmodell eine Java-Klasse entwickeln, Objekte erzeugen und begründen, warum Attribute nicht unkontrolliert verändert werden sollen.

**ARBEITSSTANDARD STARTERPROJEKTE** Umfangreiche Implementierungsaufgaben beginnen mit einem lauffähigen Ausgangsprojekt. Starte zuerst AppMain und verschaffe dir einen Überblick über View, Controller und Model. View und Controller bleiben unverändert, sofern die Aufgabe nicht ausdrücklich ihre Bearbeitung verlangt. Ergänze anschließend nur die im Projekt-README und mit TODO gekennzeichneten Stellen. Prüfe das Ergebnis zuerst über die sichtbare Anwendung und danach mit dem bereitgestellten Abnahmelauf. Unit-Tests werden gezielt ergänzt, wenn sie für das Lernziel einen zusätzlichen Nutzen haben.

## 3.1 Fallbeispiel Vereinsmitglied

### AKTIVIERUNGS-MOCKUP · Vereinsverwaltung

Abb.: Grafischer Oberflächenentwurf als Ausgangspunkt für Anforderungsanalyse und Objektmodellierung.

Das Mockup zeigt mögliche Aktionen. Die Klasse Mitglied soll jedoch nicht wissen, wie Textfelder oder Buttons aussehen. Sie schützt den fachlichen Zustand und bietet verständliche Operationen an.

### Java-Modellklasse mit geschütztem Zustand

```
public class Mitglied {
    private final String mitgliedsnummer;
    private String name;
    private String team;
    private boolean aktiv;

    public Mitglied(String mitgliedsnummer, String name) {
        this.mitgliedsnummer = mitgliedsnummer;
        this.name = name;
        this.aktiv = true;
    }

    public void wechseleTeam(String neuesTeam) {
        if (neuesTeam == null || neuesTeam.isBlank()) {
            throw new IllegalArgumentException("Team
```

```

        fehlt");
    }
    this.team = neuesTeam;
}
}

```

## 3.2 Werkstatt: Klasse, Objekt und Konstruktor

### Syntaxhilfe Java · Klasse, Attribute, Konstruktor und Objekterzeugung

```

public final class Spieler {
    private final int nummer;
    private final String name;

    public Spieler(int nummer, String name) {
        if (nummer <= 0 || name == null || name.isBlank())
        {
            throw new IllegalArgumentException("Ungültige
Spielerdaten");
        }
        this.nummer = nummer;
        this.name = name;
    }
}

// zum Beispiel in einem Test:
Spieler mia = new Spieler(7, "Mia Keller");

```

### [OOP-K03-A01] · IMPLEMENTIEREN · Klassen und Objekte selbst implementieren

1. **PROJEKTGRUNDLAGE** Verwende K03\_Grundklassen\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie SpielerAbnahme aus.
2. Lege die Klasse Spieler mit Spielernummer, Name und Position an; verwende passende Datentypen und Sichtbarkeiten.
3. Implementiere einen Konstruktor, der alle Pflichtwerte

übernimmt und ungültige Werte zurückweist.

4. Erzeuge im bereitgestellten Abnahmelauf mindestens zwei verschiedene Spielerobjekte und prüfe deren Zustand über gezielte Abfragemethoden.
5. Erweitere die Klasse anschließend um ein unveränderliches Eintrittsdatum, ohne bestehenden Testcode unnötig umzubauen.
6. Lass dir alternativ eine KI-Variante erzeugen und markiere darin Klasse, Objekt, Konstruktor, Zustand sowie mindestens eine problematische Entwurfsentscheidung.

**Erwartetes Lernprodukt:** Eine kompilierbare Java-Klasse, Objekterzeugung, ein sichtbarer Testlauf und drei automatisierte Abnahmefälle und ein kurzer Codevergleich.

**Möglicher Startprompt:** Frage mich zuerst nach Pflichtwerten und ungültigen Zuständen. Gib mir höchstens ein unvollständiges Klassengerüst mit TODO-Markierungen, aber keine fertige Lösung.

### 3.3 Fachliche Entscheidungen im Code erkennen

#### [OOP-K03-A02] · KERNPFAD · Code lesen, bevor du Code schreibst

1. Ordne jede Codezeile einem OOP-Begriff zu: Klasse, Attribut, Konstruktor, Parameter, Methode oder Kontrollstruktur.
2. Begründe die Sichtbarkeit private.
3. Erkläre, warum die Mitgliedsnummer final ist.
4. Beschreibe, welche ungültige Zustandsänderung die Methode verhindert.

**Erwartetes Lernprodukt:** Eine kommentierte Codeanalyse in eigenen Worten.

**Möglicher Startprompt:** Stelle mir Verständnisfragen zur Klasse Mitglied. Bitte mich nach jeder Antwort, die passende Codezeile zu zeigen.

### 3.4 Implementierungswerkstatt: Kapselung statt beliebiger Setter

Nicht jedes Attribut benötigt automatisch einen Setter. Eine Methode wie `wechsleTeam(...)` beschreibt die Absicht besser als `setTeam(...)`, kann Regeln prüfen und hält die Klasse auch bei späteren Änderungen verständlich.

#### Syntaxhilfe Java · Privater Zustand und kontrollierte Zustandsänderung

```
private boolean aktiv = true;

public void deaktiviere() {
    if (!aktiv) {
        throw new IllegalStateException("Mitglied ist
bereits deaktiviert");
    }
    aktiv = false;
}

public boolean istAktiv() {
    return aktiv;
}
```

#### [OOP-K03-A03] · IMPLEMENTIEREN · Eine Klasse verantwortlich machen

1. **PROJEKTGRUNDLAGE** Verwende K03\_Grundklassen\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe `AppMain` sowie `SpielerAbnahme` aus.
2. Ergänze die Methode `deaktiviere()`, die eine erneute Deaktivierung sinnvoll behandelt.
3. Entscheide, welche Getter die Oberfläche wirklich benötigt.
4. Ergänze drei automatisierte Abnahmetests: gültiger Teamwechsel, leeres Team, unveränderliche

Mitgliedsnummer.

5. Reflektiere: Welche Regel lebt jetzt genau einmal im System?
6. Prüfe einen KI-generierten Entwurf auf öffentliche Attribute, pauschale Setter und umgehbare Invarianten; verbessere genau eine Schwachstelle selbst.

**Erwartetes Lernprodukt:** Implementierte Klasse mit automatisierten Tests und kurzer Entwurfsbegründung.

### Gestufte Tutorhilfe

Hilfe 1: Beginne mit den beobachtbaren Ergebnissen, nicht mit der internen Umsetzung.

Hilfe 2: Ein Test für eine Exception braucht einen klaren ungültigen Eingabewert.

## 3.5 Ein Prinzip – drei Sprachen

| Aspekt       | Java                       | Python                          | PHP                                  |
|--------------|----------------------------|---------------------------------|--------------------------------------|
| Typen        | statisch, explizit         | dynamisch;<br>Hinweise optional | statisch<br>ergänzbar,<br>deklarativ |
| Sichtbarkeit | sprachlich<br>durchgesetzt | Konventionen/<br>Properties     | sprachlich<br>durchgesetzt           |
| Konstruktion | Klassenname                | <code>__init__</code>           | <code>__construct</code>             |
| Transferwert | Verträge sehr<br>sichtbar  | Modell kompakt<br>erkennbar     | Anschluss an<br>Websysteme           |

### [OOP-K03-A04] · SPRACHTRANSFER · Sprachunabhängige Entwurfsentscheidung

1. Markiere in drei gleichwertigen Kurzimplementierungen Zustand, Konstruktion und Verhalten.
2. Notiere Unterschiede der Syntax getrennt von Unterschieden des Objektmodells.
3. Begründe, welche Invariante alle drei Klassen schützen müssen.
4. Formuliere das Modell zunächst als UML-Klasse und erst danach in einer gewählten Transfersprache.

**Erwartetes Lernprodukt:** Eine UML-Klasse, eine Vergleichsnotiz und eine kurze Python- oder PHP-

| Aspekt | Java                                                                                                                                                                                    | Python | PHP |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|-----|
|        | Implementierung.<br><b>Möglicher Startprompt:</b> Prüfe nur, ob meine drei Implementierungen dasselbe fachliche Modell ausdrücken. Korrigiere Syntax erst nach meiner Modellbegründung. |        |     |

### 3.6 Transfer zum Lernagenten

#### [OOP-K03-A05] · TRANSFER LERNAGENT · Die erste Modellklasse des Lernagenten

1. **PROJEKTGRUNDLAGE** Verwende K03\_Grundklassen\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie SpielerAbnahme aus.
2. Entwirf die Klasse Lernziel mit Kennung, Beschreibung und Status.
3. Lege fest, welche Attribute nach der Erzeugung unveränderlich sein sollen.
4. Formuliere statt pauschaler Setter fachliche Methoden, zum Beispiel starteBearbeitung() oder markiereErreicht().
5. Implementiere und teste mindestens eine Zustandsregel.

**Erwartetes Lernprodukt:** UML-Klassendiagramm, Java-Klasse und mindestens drei Tests.

**Möglicher Startprompt:** Prüfe, ob meine Methoden fachliche Absichten ausdrücken und ob ungültige Zustände möglich bleiben. Antworte zunächst nur mit Rückfragen.

## 3.7 Lehrplanwerkstatt: Datentypen, Startklasse und Gültigkeitsbereiche

### KERNWISSEN

Primitive Datentypen speichern einzelne Werte, zum Beispiel int, double und boolean. Komplexe Datentypen verweisen auf Objekte, zum Beispiel String, Mitglied oder List<Mitglied>.

Die Startklasse enthält den Programmeinstieg und erzeugt beziehungsweise verdrahtet Domänenobjekte. Fachregeln gehören in die Domänenklassen – nicht in main().

Lokale Variablen gelten im Block, Parameter während des Methodenaufrufs und Attribute während der Lebensdauer des Objekts. Methodenüberladung bedeutet: gleicher Methodenname, unterschiedliche Parameterlisten.

```
public final class App {
    public static void main(String[] args) {
        Mitglied mia = new Mitglied("Mia", 17);
        System.out.println(mia.beitrag());
    }
}

public double beitrag() { return 24.0; }
public double beitrag(double rabatt) { return 24.0 * (1 - rabatt); }
```

### [OOP-K03-A06] · IMPLEMENTIEREN · Start- und Domänenklasse trennen

**PROJEKTGRUNDLAGE** Verwende K03\_Grundklassen\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie SpielerAbnahme aus.

1. Lege App und Mitglied als getrennte Klassen an.
2. Verwende mindestens zwei primitive und einen komplexen Datentyp.
3. Überlade eine fachlich sinnvolle Methode und rufe beide Varianten auf.
4. Markiere im Code je ein Attribut, einen Parameter und eine lokale Variable.

Prüfkriterien: kompiliert · main() enthält keine Fachberechnung · beide Methodenaufrufe liefern nachvollziehbare Ergebnisse.

## 3.8 Kapitel-Sicherung

HALTE FEST · Klasse, Objekt, Konstruktor und Kapselung. Prüfe, ob ein ungültiger Zustand noch von außen erzeugt werden kann.

MINI-CHECK

1. Erkläre den Kernbegriff in höchstens zwei Sätzen.
2. Zeige ihn an einem konkreten Code-, Modell- oder Testbeleg.
3. Nenne eine typische Fehlentscheidung und ihre erkennbare Folge.

GESTUFTE HILFE

Denkimpuls: Welche fachliche Verantwortung steht im Mittelpunkt?  
Konkreter Hinweis: Prüfe, ob ein ungültiger Zustand noch von außen erzeugt werden kann.

Teilstruktur: Ausgangslage → Entwurfsentscheidung → Implementierung/Modell → Test → Änderungsfolge.

[Zurück zum Inhaltsverzeichnis](#)

## 4 · Kontrollstrukturen gehören zu Verantwortlichkeiten [← Inhalt](#)

**DEIN KAPITELSTART** Du setzt Bedingungen und Fallunterscheidungen nicht als isolierte Syntaxübungen ein, sondern modellierst damit fachliche Regeln verantwortlicher Objekte.

### 4.1 Projektkarussell: drei Automaten - ein Denkprinzip

| Projekt         | Eingaben          | Regel                         | mögliche Modellklasse |
|-----------------|-------------------|-------------------------------|-----------------------|
| Milchautomat    | Menge, Gefäß      | Abgabe nur bei gültiger Menge | Milchautomat          |
| Parkautomat     | Dauer, Tarifzone  | Preis und Höchstdauer prüfen  | Parkvorgang           |
| Getränkeautomat | Auswahl, Guthaben | Bestand und Zahlung prüfen    | Verkaufsautomat       |

**[OOP-K04-A01] · KERNPFAD · Station 1 ·**

| Projekt                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Eingaben | Regel | mögliche Modellklasse |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|-------|-----------------------|
| <b>Milchautomat</b> <ol style="list-style-type: none"> <li>1. <b>PROJEKTGRUNDLAGE</b> Verwende K04_Automatenkarussell_MVC aus 02_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie AutomatenAbnahme aus.</li> <li>2. Leite aus einem einfachen Mockup die Zustände bereit, ausverkauft und störung ab.</li> <li>3. Formuliere eine Entscheidungstabelle für die Abgabe.</li> <li>4. Implementiere die Regel zunächst mit if/else in einer Modellmethode.</li> <li>5. Teste mindestens einen Grenzwert.</li> </ol> <p><b>Erwartetes Lernprodukt:</b> Entscheidungstabelle, Methode und drei Testfälle.</p> <p><b>Möglicher Startprompt:</b> Prüfe meine Entscheidungstabelle auf fehlende Kombinationen. Gib mir nur Hinweise auf Lücken.</p> |          |       |                       |

### [OOP-K04-A02] · BKWI2-AUFBAU · Station 2 · Parkautomat

1. **PROJEKTGRUNDLAGE** Verwende K04\_Automatenkarussell\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie AutomatenAbnahme aus.
  2. Modelliere verschiedene Tarifzonen mit switch.
  3. Trenne Preisberechnung und Ticketdarstellung.
  4. Begründe, welches Objekt die Höchstdauer prüfen sollte.
  5. Ergänze einen Test für eine unzulässige Parkdauer.
- Erwartetes Lernprodukt:** Eine erweiterbare Preislogik ohne duplizierte Tarifformeln.

### [OOP-K04-A03] · VERTIEFUNG · Station 3 · Getränkeautomat

1. **PROJEKTGRUNDLAGE** Verwende K04\_Automatenkarussell\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie AutomatenAbnahme aus.
2. Prüfe Auswahl, Bestand und Guthaben in einer sinnvollen Reihenfolge.
3. Vermeide tief verschachtelte Bedingungen durch frühe Rückgaben oder kleine Prüfmethode.
4. Beschreibe, wann ein Zustandsobjekt oder Enum später sinnvoll werden könnte.
5. Vergleiche die Lesbarkeit beider Varianten.

**Erwartetes Lernprodukt:** Zwei funktional gleiche Implementierungen mit Qualitätsvergleich.

## 4.2 Transfer statt Beispielsammlung

Nach jeder Station werden nicht nur Ergebnisse verglichen. Entscheidend ist, welche Regel verallgemeinert werden kann und welche fachlich zum jeweiligen Automaten gehört. Dadurch entsteht keine künstliche Universal-Automatenklasse.

**REFLEXIONSFRAGE** Welche Redundanz ist wirklich problematisch – und welche Ähnlichkeit sollte bewusst getrennt bleiben, weil die fachliche Bedeutung verschieden ist?

## 4.3 Transfer zum Lernagenten

### [OOP-K04-A04] · TRANSFER LERNAGENT · Entscheidungslogik für den nächsten Lernschritt

1. **PROJEKTGRUNDLAGE** Verwende K04\_Automatenkarussell\_MVC aus

02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie AutomatenAbnahme aus.

2. Formuliere Regeln für sicher, teilweise sicher und noch unsicher.
3. Ordne jedem Zustand eine passende nächste Aktion zu.
4. Implementiere die Auswahl zunächst mit einer übersichtlichen Kontrollstruktur.
5. Prüfe, ob die Regel in die Klasse Lernstand, Aufgabe oder Lernpfad gehört, und begründe deine Entscheidung.
6. Teste Grenzfälle, etwa fehlende Selbsteinschätzung oder keine weitere Aufgabe.

**Erwartetes Lernprodukt:** Eine getestete Methode zur Auswahl eines nächsten Lernschritts plus Verantwortungsbegründung.

## 4.4 Vom Ablauf zum Struktogramm

### METHODENHILFE

Modelliere Sequenz, Verzweigung und Schleife zuerst als Struktogramm. Im Unterricht ist der eingeführte Struktogramm-Editor maßgeblich; draw.io kann ergänzend zur Dokumentation genutzt werden.

Jeder Block muss sich eindeutig einer Codezeile oder einem Codeblock zuordnen lassen. Prüfe Grenzfälle, bevor du implementierst.

### [OOP-K04-A05] · MODELLIEREN UND IMPLEMENTIEREN · Parkautomat

**PROJEKTGRUNDLAGE** Verwende K04\_Automatenkarussell\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie AutomatenAbnahme aus.

1. Erstelle ein Struktogramm: Eingabe Minuten, Ablehnung negativer Werte, Preis je angefangene 30 Minuten,

Tageshöchstpreis.

2. Übertrage das Modell in eine Java-Methode `berechnePreis(int minuten)`.

3. Prüfe mindestens die Werte  $-1$ ,  $0$ ,  $29$ ,  $30$ ,  $31$  und einen Wert oberhalb des Tagesmaximums.

Prüfkriterien: vollständige drei Kontrollstrukturtypen ·

Struktogramm und Code sind deckungsgleich · erwartete und tatsächliche Ergebnisse sind dokumentiert.

## 4.5 Kapitel-Sicherung

HALTE FEST · Bedingungen als Fachregeln. Leite zuerst vollständige Fälle und Grenzwerte ab.

MINI-CHECK

1. Erkläre den Kernbegriff in höchstens zwei Sätzen.

2. Zeige ihn an einem konkreten Code-, Modell- oder Testbeleg.

3. Nenne eine typische Fehlentscheidung und ihre erkennbare Folge.

GESTUFTE HILFE

Denkimpuls: Welche fachliche Verantwortung steht im Mittelpunkt?

Konkreter Hinweis: Leite zuerst vollständige Fälle und Grenzwerte ab.

Teilstruktur: Ausgangslage → Entwurfsentscheidung →

Implementierung/Modell → Test → Änderungsfolge.

[Zurück zum Inhaltsverzeichnis](#)

## 5 · Methoden, Verträge und automatisierte Tests [↩ Inhalt](#)

**DEIN KAPITELSTART** Du formulierst Methoden als überprüfbare Verträge. Gültige Zustände, erlaubte Eingaben und zugesicherte Ergebnisse werden sichtbar, bevor eine Oberfläche oder Datenbank hinzukommt.

## 5.1 Vom Button zur fachlichen Operation

Ein Button beschreibt nur eine Bedienhandlung. Im Modell zählt die fachliche Absicht: Ein Mitglied wird nicht beliebig gesetzt, sondern aufgenommen, einem Team zugeordnet oder deaktiviert. Eine Methode bündelt dazu Eingabeprüfung, Zustandsänderung und zugesichertes Ergebnis.

| Vertragsteil  | Leitfrage                   | BMI-Beispiel                 | Prüfung           |
|---------------|-----------------------------|------------------------------|-------------------|
| Vorbedingung  | Was muss vorher gelten?     | Größe > 0;<br>Gewicht > 0    | Negativtest       |
| Nachbedingung | Was ist danach zugesichert? | BMI ist positiv<br>berechnet | Ergebnistest      |
| Invariante    | Was bleibt immer gültig?    | Messwerte sind<br>plausibel  | Zustandstest      |
| Seiteneffekt  | Was verändert sich?         | reine Berechnung:<br>nichts  | Wiederholungstest |

### [OOP-K05-A01] · KERNPFAD · Einen Methodenvertrag formulieren

1. **PROJEKTGRUNDLAGE** Verwende K05\_BMI\_und\_Zustaende\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie VertragsAbnahme aus.
2. Leite aus dem BMI-Mockup die fachliche Operation berechneBmi ab.
3. Formuliere Vorbedingung, Nachbedingung und mindestens eine Invariante in eigenen Worten.
4. Entscheide, ob die Methode Zustand verändern oder nur einen Wert liefern soll.
5. Notiere zunächst Testfälle und implementiere die Methode erst danach.

**Erwartetes Lernprodukt:** Eine Vertragskarte und eine Tabelle mit Normal-, Grenz- und Fehlerfällen.

**Möglicher Startprompt:** Stelle mir Fragen zu gültigen Eingaben und zugesicherten Ergebnissen. Formuliere den Vertrag erst, nachdem ich alle Grenzfälle selbst benannt habe.

**Gestufte Tutorhilfe**

| Vertragsteil | Leitfrage | BMI-Beispiel | Prüfung |
|--------------|-----------|--------------|---------|
|--------------|-----------|--------------|---------|

Hilfe 1: Teste Werte direkt an der Grenze.  
Hilfe 2: Ein Fehlerfall braucht eine erwartete Reaktion, nicht nur die Aussage 'ungültig'.

## 5.2 Tests beschreiben beobachtbares Verhalten

### Java · Eine kleine Klasse mit klarer Systemgrenze

```
public final class BmiRechner {
    public double berechne(double gewichtKg, double
    groesseM) {
        if (gewichtKg <= 0 || groesseM <= 0) {
            throw new IllegalArgumentException("Messwerte
            müssen positiv sein");
        }
        return gewichtKg / (groesseM * groesseM);
    }
}
```

### JUnit · Positiv- und Negativtest als ausführbare Beispiele

```
@Test
void berechnetBmiFuerGueltigeMesswerte() {
    double bmi = new BmiRechner().berechne(72.0, 1.80);
    assertEquals(22.22, bmi, 0.01);
}

@Test
void weistNichtPositiveGroesseZurueck() {
    assertThrows(IllegalArgumentException.class,
        () -> new BmiRechner().berechne(72.0, 0.0));
}
```

### [OOP-K05-A02] · BKWI2-AUFBAU · Änderung sicher durchführen

1. **PROJEKTGRUNDLAGE** Verwende K05\_BMI\_und\_Zustaende\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README

und führe AppMain sowie VertragsAbnahme aus.

2. Ergänze eine fachliche Bewertungskategorie, ohne die Berechnungsformel zu duplizieren.
3. Markiere, welche vorhandenen Tests unverändert gültig bleiben müssen.
4. Schreibe einen Test, der die neue Grenze exakt prüft.
5. Führe die Tests vor und nach der Änderung aus und dokumentiere das Ergebnis.

**Erwartetes Lernprodukt:** Eine getestete Änderung mit kurzem Änderungsprotokoll.

**Möglicher Startprompt:** Prüfe meine Tests auf Aussagekraft und fehlende Grenzfälle. Gib keine Implementierung vor.

## 5.3 Sprachtransfer: Vertrag bleibt, Testwerkzeug wechselt

| Aspekt               | Java                         | Python                   | PHP                          |
|----------------------|------------------------------|--------------------------|------------------------------|
| Fehler signalisieren | IllegalArgumentExce<br>ption | ValueError               | InvalidArgumentExce<br>ption |
| Testwerkzeug         | JUnit                        | pytest / unittest        | PHPUnit                      |
| Gleitkommazahl       | assertEquals mit<br>Delta    | approx / almost<br>equal | assertEqualsWithDelt<br>a    |
| Transferkern         | Vertrag und Fälle            | Vertrag und Fälle        | Vertrag und Fälle            |

### [OOP-K05-A03] · SPRACHTRANSFER · Vertrag statt Syntax übertragen

1. Übertrage die beiden Testfälle in Python oder PHP.
2. Behalte Eingabebereich, erwartetes Ergebnis und Fehlerszenario fachlich unverändert.
3. Kennzeichne reine Syntaxunterschiede getrennt von Änderungen am Vertrag.

**Erwartetes Lernprodukt:** Eine kurze Testübersetzung mit begründetem Gleichheitsnachweis.

## 5.4 Transfer zum Lernagenten

### [OOP-K05-A04] · TRANSFER LERNAGENT · Gültige Lernzustände absichern

1. **PROJEKTGRUNDLAGE** Verwende K05\_BMI\_und\_Zustaende\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie VertragsAbnahme aus.
2. Formuliere einen Vertrag für markiereErreicht und einen für starteBearbeitung.
3. Lege fest, welche Zustandswechsel erlaubt und welche unzulässig sind.
4. Schreibe Tests für einen normalen Verlauf und mindestens zwei ungültige Übergänge.
5. Prüfe, ob die Oberfläche einen ungültigen Zustand überhaupt erzeugen kann.

**Erwartetes Lernprodukt:** Eine getestete Zustandslogik für Lernziele.

**Möglicher Startprompt:** Frage mich nach den erlaubten Zustandsübergängen. Hilf mir, daraus Tests abzuleiten, ohne die Methoden fertig zu schreiben.

## 5.5 Kapitel-Sicherung

HALTE FEST · Methodenvertrag und Tests. Ordne Vorbedingung, Nachbedingung und Invariante einem Test zu.

MINI-CHECK

1. Erkläre den Kernbegriff in höchstens zwei Sätzen.
2. Zeige ihn an einem konkreten Code-, Modell- oder Testbeleg.
3. Nenne eine typische Fehlentscheidung und ihre erkennbare Folge.

GESTUFTE HILFE

Denkimpuls: Welche fachliche Verantwortung steht im Mittelpunkt?  
Konkreter Hinweis: Ordne Vorbedingung, Nachbedingung und Invariante einem Test zu.

Teilstruktur: Ausgangslage → Entwurfsentscheidung → Implementierung/Modell → Test → Änderungsfolge.

[Zurück zum Inhaltsverzeichnis](#)

## 6 · Listen, Schleifen und Objektmengen

← Inhalt

**DEIN KAPITELSTART** Du verarbeitest nicht lose Parallel-Arrays, sondern Sammlungen fachlicher Objekte. Schleifen, Filter und Sortierungen beantworten konkrete Fragen des Modells.

### AKTIVIERUNGS-MOCKUP · Volleyball-Teammanager

Abb.: Grafischer Oberflächenentwurf als Ausgangspunkt für Anforderungsanalyse und Objektmodellierung.

### 6.1 Von Einzelobjekten zur Sammlung

Ein Team verwaltet Spielerobjekte. Name, Position und Einsatzstatus bleiben dadurch zusammen. Mehrere getrennte Listen für Namen, Positionen und Statuswerte würden leicht auseinanderlaufen und erschweren spätere Erweiterungen.

**[OOP-K06-A01] · KERNPFAD · Objektliste statt Parallelstruktur**

1. **PROJEKTGRUNDLAGE** Verwende K06\_K07\_Team\_Shop\_Beziehungen\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie BeziehungsAbnahme aus.
2. Leite aus dem Mockup die Klassen Team und Spieler ab.
3. Begründe, warum eine Liste von Spielerobjekten geeigneter ist als drei parallele Listen.
4. Definiere eine Methode aufnehmen, die doppelte Spielernummern verhindert.
5. Teste das Einfügen eines neuen und eines bereits vorhandenen Spielers.

**Erwartetes Lernprodukt:** UML-Ausschnitt, Klassenentwurf und Tests für die Aufnahme.

**Möglicher Startprompt:** Lass mich die Risiken paralleler Listen selbst entdecken. Frage nach Änderungen, Löschungen und Sortierungen.

## 6.2 Schleifen beantworten fachliche Fragen

### Java · Verständliche Schleife mit geschützter Rückgabe

```
public List<Spieler> einsatzberechtigteSpieler() {
    List<Spieler> auswahl = new ArrayList<>();
    for (Spieler kandidat : spieler) {
        if (kandidat.istEinsatzbereit()) {
            auswahl.add(kandidat);
        }
    }
    return List.copyOf(auswahl);
}
```

**ENTWURFSENTSCHEIDUNG** Die Methode gibt keine veränderbare interne Liste heraus. Das Team behält die Kontrolle über seine Sammlung; aufrufender Code erhält

eine unabhängige Sicht auf das Ergebnis.

### [OOP-K06-A02] · BKWI2-AUFBAU · Suchen, filtern und sortieren

1. **PROJEKTGRUNDLAGE** Verwende K06\_K07\_Team\_Shop\_Beziehungen\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie BeziehungsAbnahme aus.
2. Implementiere eine lineare Suche nach Spielernummer.
3. Filtere alle einsatzberechtigten Spieler zunächst mit einer Schleife.
4. Sortiere das Ergebnis nach Position und Name, ohne die interne Teamliste unbeabsichtigt umzubauen.
5. Bestimme für Suche, Filter und Sortierung jeweils passende Testfälle.

**Erwartetes Lernprodukt:** Eine getestete Teamabfrage mit begründeter Wahl der Datenstruktur.

**Möglicher Startprompt:** Prüfe meine Wahl von Liste, Suche und Sortierung anhand der benötigten Operationen. Frage zuerst nach Datenmenge und Änderungsmuster.

## 6.3 Sprachtransfer: verschiedene Sammlungen, gleiche Modellfrage

| Operation   | Java                              | Python                               | PHP                                  |
|-------------|-----------------------------------|--------------------------------------|--------------------------------------|
| Sammlung    | List<Spieler>                     | list[Spieler]                        | array von Spieler                    |
| Iteration   | for-each                          | for                                  | foreach                              |
| Filtern     | Schleife / Stream                 | Schleife /<br>Comprehension          | foreach /<br>array_filter            |
| Schutzfrage | interne Liste nicht<br>preisgeben | Kopie /<br>kontrollierter<br>Zugriff | Kopie /<br>kontrollierter<br>Zugriff |

### [OOP-K06-A03] · SPRACHTRANSFER · Eine Abfrage in drei Sprachen lesen

1. Markiere in drei Kurzvarianten Sammlung, Iteration,

| Operation | Java                                                                                                                                                                                                                                                                                                                                                                   | Python | PHP |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|-----|
|           | <p>Bedingung und Ergebnis.</p> <ol style="list-style-type: none"> <li>2. Erkläre, warum das Team in allen Sprachen Eigentümer seiner Spiellersammlung bleibt.</li> <li>3. Übertrage nur die Methode einsatzberechtigteSpieler in eine Transfersprache.</li> </ol> <p><b>Erwartetes Lernprodukt:</b> Eine kommentierte Gegenüberstellung derselben Objektoperation.</p> |        |     |

## 6.4 Transfer zum Lernagenten

### [OOP-K06-A04] · TRANSFER LERNAGENT · Aufgabenpool modellieren

1. **PROJEKTGRUNDLAGE** Verwende K06\_K07\_Team\_Shop\_Beziehungen\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie BeziehungsAbnahme aus.
2. Modelliere einen Aufgabenpool als Sammlung von Aufgabenobjekten.
3. Filtere Aufgaben nach Lernziel, Schwierigkeitsgrad und Bearbeitungsstatus.
4. Verhindere, dass dieselbe Aufgabe zweimal in denselben Lernpfad aufgenommen wird.
5. Teste leere Treffermengen und mehrere gleich geeignete Aufgaben.

**Erwartetes Lernprodukt:** Ein getesteter Aufgabenpool mit fachlichen Filtermethoden.

## 6.5 Kapitel-Sicherung

HALTE FEST · Objektlisten und Schleifen. Begründe, warum eine Liste von Objekten Parallelstrukturen ersetzt.

MINI-CHECK

1. Erkläre den Kernbegriff in höchstens zwei Sätzen.

2. Zeige ihn an einem konkreten Code-, Modell- oder Testbeleg.
3. Nenne eine typische Fehlentscheidung und ihre erkennbare Folge.

### GESTUFTE HILFE

Denkimpuls: Welche fachliche Verantwortung steht im Mittelpunkt?

Konkreter Hinweis: Begründe, warum eine Liste von Objekten Parallelstrukturen ersetzt.

Teilstruktur: Ausgangslage → Entwurfsentscheidung →

Implementierung/Modell → Test → Änderungsfolge.

[Zurück zum Inhaltsverzeichnis](#)

## 7 · Objektbeziehungen bewusst modellieren [← Inhalt](#)

**DEIN KAPITELSTART** Du entscheidest nicht nur, welche Objekte existieren, sondern auch, wie sie zusammenarbeiten, wem Teilobjekte gehören und welche Lebensdauer fachlich sinnvoll ist.

### 7.1 Beziehungen enthalten Fachbedeutung

| Beziehung   | Beispiel                 | Leitfrage                            | typische Umsetzung    |
|-------------|--------------------------|--------------------------------------|-----------------------|
| 1:1         | Mitglied – Ausweis       | Ist genau ein Partner erlaubt?       | ein Attribut          |
| 1:n         | Team – Spieler           | Wer verwaltet die Menge?             | Liste am Eigentümer   |
| n:m         | Mitglied – Veranstaltung | Braucht die Verbindung eigene Daten? | Verbindungsobjekt     |
| Komposition | Bestellung – Position    | Kann der Teil allein sinnvoll leben? | Erzeugung über Ganzes |

### [OOP-K07-A01] · KERNPFAD · Vereinsbeziehungen aus Fachregeln ableiten

1. Modelliere Mitglieder, Teams und Teamzugehörigkeiten zunächst ohne Datenbanktabellen.
2. Lege Multiplizitäten und Navigationsrichtungen fest.
3. Entscheide, ob eine Teamzugehörigkeit Beginn, Rolle oder Rückennummer speichern soll.

| Beziehung                                                                                                                                                                                                                                                                                                                         | Beispiel | Leitfrage | typische Umsetzung |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|-----------|--------------------|
| <p>4. Begründe, wann daraus eine eigene Klasse wird.</p> <p><b>Erwartetes Lernprodukt:</b> Ein UML-Beziehungsmodell mit begründeten Multiplizitäten.</p> <p><b>Möglicher Startprompt:</b> Frage mich zu jeder Beziehung nach Anzahl, Lebensdauer und eigenen Informationen. Nenne den Beziehungstyp erst nach meiner Antwort.</p> |          |           |                    |

## 7.2 Implementierungswerkstatt: Assoziation und Aggregation

### Syntaxhilfe Java · Ein Objekt verwaltet eine Beziehung zu vorhandenen Objekten

```
public final class Team {
    private final List<Spieler> spieler = new
    ArrayList<>();

    public void nehmeAuf(Spieler kandidat) {
        if (kandidat == null) {
            throw new IllegalArgumentException("Spieler
fehlt");
        }
        boolean nummerVergeben = spieler.stream()
            .anyMatch(s -> s.nummer() ==
kandidat.nummer());
        if (nummerVergeben) throw new
IllegalArgumentException("Nummer vergeben");
        spieler.add(kandidat);
    }

    public List<Spieler> spieler() {
        return List.copyOf(spieler);
    }
}
```

**[OOP-K07-A02] · IMPLEMENTIEREN · Eine 1:n-Beziehung implementieren**

1. **PROJEKTGRUNDLAGE** Verwende K06\_K07\_Team\_Shop\_Beziehungen\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie BeziehungsAbnahme aus.
2. Implementiere Team und Spieler als zwei eigenständige Klassen; ein Team verwaltet mehrere vorhandene Spielerobjekte.
3. Sichere beim Aufnehmen, dass weder null noch eine doppelte Spielernummer zugelassen wird.
4. Gib keine veränderbare interne Liste heraus und belege den Schutz mit einem Test.
5. Ergänze die Anforderung, einen Spieler anhand seiner Nummer zu finden, ohne Beziehungsdaten zu duplizieren.
6. Bewerte eine KI-Lösung danach, ob Eigentümerschaft, Multiplizität und Änderungsrichtung zum Fachmodell passen.

**Erwartetes Lernprodukt:** Zwei verbundene Java-Klassen, mindestens vier Beziehungstests und eine begründete Navigationsrichtung.

**Möglicher Startprompt:** Lass mich zuerst festlegen, welches Objekt die Beziehung verwaltet. Zeige anschließend nur die Signaturen der benötigten Methoden.

## 7.3 Implementierungswerkstatt: Komposition schützt den Zusammenhang

### Java · Die Bestellung kontrolliert ihre Positionen

```
public final class Bestellung {
    private final List<Bestellposition> positionen = new
    ArrayList<>();

    public void fuegeHinzu(Produkt produkt, int menge) {
        if (menge <= 0) throw new
        IllegalArgumentException("Menge");
        positionen.add(new Bestellposition(produkt,
```

```

menge));
    }

    public List<Bestellposition> positionen() {
        return List.copyOf(positionen);
    }
}

```

### [OOP-K07-A03] · IMPLEMENTIEREN · Komposition im Mini-Shop implementieren

1. **PROJEKTGRUNDLAGE** Verwende K06\_K07\_Team\_Shop\_Beziehungen\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie BeziehungsAbnahme aus.
2. Implementiere Bestellposition mit Produkt, Menge und Einzelpreis; der Konstruktor weist ungültige Werte zurück.
3. Ergänze entfernePosition(...) und berechneGesamtsumme(), ohne die interne Liste herauszugeben.
4. Erzeuge Positionen nur über Bestellung und begründe die gemeinsame Lebensdauer.
5. Teste ungültige Mengen, Summe, Entfernen und den Schutz der Positionsliste.
6. Ergänze eine Rabattregel und begründe, welche Klasse dafür verantwortlich ist.

**Erwartetes Lernprodukt:** Bestellung und Bestellposition, vier Tests und eine Änderungs begründung.

**Möglicher Startprompt:** Prüfe meine Verantwortungsverteilung mit Fragen; gib keine fertige Berechnung aus.

## 7.4 Typische Beziehungsfehler erkennen

### [OOP-K07-A04] · VERTIEFUNG · Modellreview

1. Suche im Modell nach beidseitig veränderbaren Listen.
2. Markiere Beziehungen, deren Multiplizität nur geraten wurde.
3. Prüfe, ob IDs statt echter Objektbeziehungen das Fachmodell unnötig technisch machen.
4. Formuliere für jeden Fund eine lokale Verbesserung und einen Regressionstest.

**Erwartetes Lernprodukt:** Ein Reviewprotokoll mit priorisierten Modellverbesserungen.

**Möglicher Startprompt:** Gib mir zu jedem Modellfehler nur eine Diagnosefrage. Warte auf meine Verbesserungsidee, bevor du bewertest.

## 7.5 Transfer zum Lernagenten

### [OOP-K07-A05] · TRANSFER LERNAGENT · Lernverlauf als Beziehung modellieren

1. **PROJEKTGRUNDLAGE** Verwende K06\_K07\_Team\_Shop\_Beziehungen\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie BeziehungsAbnahme aus.
2. Verbinde Lernende, Lernziele und Bearbeitungen mit begründeten Multiplizitäten.
3. Entscheide, welche Informationen zur Bearbeitung und nicht zur Aufgabe gehören.
4. Modelliere Versuchsnummer, Zeitpunkt, Ergebnis und Rückmeldung in einem geeigneten Objekt.
5. Teste, dass eine Bearbeitung nicht ohne Lernenden und Aufgabe entstehen kann.

**Erwartetes Lernprodukt:** Ein Lernverlaufsmodell mit getesteten Konsistenzregeln.

## 7.6 Kapitel-Sicherung

HALTE FEST · Assoziation, Aggregation und Komposition. Prüfe Besitz, Lebensdauer und Multiplizität getrennt.

MINI-CHECK

1. Erkläre den Kernbegriff in höchstens zwei Sätzen.
2. Zeige ihn an einem konkreten Code-, Modell- oder Testbeleg.
3. Nenne eine typische Fehlentscheidung und ihre erkennbare Folge.

GESTUFTE HILFE

Denkimpuls: Welche fachliche Verantwortung steht im Mittelpunkt?

Konkreter Hinweis: Prüfe Besitz, Lebensdauer und Multiplizität getrennt.

Teilstruktur: Ausgangslage → Entwurfsentscheidung →

Implementierung/Modell → Test → Änderungsfolge.

[Zurück zum Inhaltsverzeichnis](#)

## 8 · Abstraktion, Interfaces und Polymorphie [↔ Inhalt](#)

**DEIN KAPITELSTART** Du programmierst gegen gemeinsame Verträge. Konkrete Varianten werden austauschbar, ohne dass aufrufender Code ihre internen Details kennen muss.

### 8.1 Implementierungswerkstatt: Abstraktion durch ein Interface

Wenn ein Lernagent verschiedene Rückmeldungen geben soll, führt eine wachsende if- oder switch-Kette schnell zu zentraler Änderungsabhängigkeit. Ein Interface beschreibt stattdessen, was jede Feedbackstrategie leisten muss. Die konkrete Auswahl wird von der Nutzung getrennt.

## Syntaxhilfe Java · Gemeinsamer Vertrag und eine konkrete Implementierung

```
public interface FeedbackStrategie {
    String erzeugeFeedback(Bearbeitung bearbeitung);
}

public final class MotivierendesFeedback implements
FeedbackStrategie {
    @Override
    public String erzeugeFeedback(Bearbeitung bearbeitung)
    {
        return bearbeitung.istErfolgreich()
            ? "Stark – das Lernziel sitzt."
            : "Guter Anfang – prüfe deinen Lösungsweg noch
einmal.";
    }
}
```

### [OOP-K08-A01] · IMPLEMENTIEREN · Ein Interface und zwei Implementierungen entwickeln

1. **PROJEKTGRUNDLAGE** Verwende K08\_Feedback\_und\_Vereinsrollen\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie PolymorphieAbnahme aus.
2. Formuliere den Vertrag einer FeedbackStrategie in eigenen Worten.
3. Implementiere eine motivierende und eine prüfungsnahe Strategie.
4. Lege für beide Klassen dieselben Methodensignaturen fest und kennzeichne @Override.
5. Teste jede Implementierung isoliert mit einer erfolgreichen und einer nicht erfolgreichen Bearbeitung.
6. Prüfe eine KI-Lösung darauf, ob sie wirklich gegen den Vertrag implementiert oder nur gleichnamige Methoden zufällig wiederholt.

**Erwartetes Lernprodukt:** Interface, zwei konkrete Java-

Klassen und vier Verhaltenstests.

**Möglicher Startprompt:** Zeige mir zuerst nur ein Interface-Gerüst. Lass mich Rückgabtyp, Parameter und Vertragsaussage selbst festlegen.

## 8.2 Implementierungswerkstatt: Vererbung bewusst einsetzen

### Syntaxhilfe Java · Abstrakte Basisklasse, extends, super und Override

```
public abstract class Vereinsrolle {
    private final String name;

    protected Vereinsrolle(String name) {
        if (name == null || name.isBlank()) {
            throw new IllegalArgumentException("Name");
        }
        this.name = name;
    }

    public String name() { return name; }
    public abstract String aufgabeImVerein();
}

public final class Spielerrolle extends Vereinsrolle {
    public Spielerrolle(String name) { super(name); }

    @Override
    public String aufgabeImVerein() { return "spielt im Team"; }
}
```

### [OOP-K08-A02] · IMPLEMENTIEREN · Eine kleine, begründete Vererbungshierarchie implementieren

1. **PROJEKTGRUNDLAGE** Verwende K08\_Feedback\_und\_Vereinsrollen\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die

- freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie PolymorphieAbnahme aus.
2. Implementiere die abstrakte Basisklasse Vereinsrolle und die Unterklassen Spielerrolle und Trainerrolle.
  3. Verschiebe nur wirklich gemeinsamen Zustand und gemeinsames Verhalten in die Basisklasse.
  4. Erzeuge beide Unterklassen und teste Konstruktion sowie überschriebenes Verhalten.
  5. Ergänze eine dritte Rolle und prüfe, ob dafür die Basisklasse verändert werden musste.
  6. Formuliere die Ist-ein-Aussage für jede Unterklasse und lehne mindestens eine unpassende vorgeschlagene Unterklasse begründet ab.

**Erwartetes Lernprodukt:** Abstrakte Basisklasse, drei Unterklassen, Tests und eine kurze Prüfung des Ersetzungsprinzips.

**Möglicher Startprompt:** Frage mich vor jedem extends nach der Ist-ein-Aussage. Weise mich auf Kopplungsrisiken hin, ohne die Hierarchie fertig zu entwerfen.

## 8.3 Vererbung oder Komposition?

| Prüffrage               | Vererbung kann passen                | Komposition ist oft besser                 |
|-------------------------|--------------------------------------|--------------------------------------------|
| Ist-es-Beziehung?       | Untertyp erfüllt denselben Vertrag   | Rolle oder Verhalten wird nur genutzt      |
| Austausch zur Laufzeit? | selten erforderlich                  | Strategie kann gewechselt werden           |
| Änderungsgrund?         | gemeinsame stabile Abstraktion       | Varianten ändern unabhängig                |
| Kopplungsrisiko?        | Basisklasse beeinflusst Unterklassen | kleine Schnittstelle begrenzt Abhängigkeit |

### [OOP-K08-A03] · BKWI2-AUFBAU · Hierarchie kritisch prüfen

1. Untersuche eine vorgeschlagene Vererbungshierarchie für Feedbackarten.
2. Formuliere die echte Ist-es-Aussage und prüfe das Ersetzungsprinzip.

| Prüffrage                                                                                 | Vererbung kann passen | Komposition ist oft besser |
|-------------------------------------------------------------------------------------------|-----------------------|----------------------------|
| 3. Entwirf alternativ eine Komposition mit Strategie-Schnittstelle.                       |                       |                            |
| 4. Vergleiche, welche Variante sich bei einer neuen Feedbackart lokal erweitern lässt.    |                       |                            |
| <b>Erwartetes Lernprodukt:</b> Ein begründeter Entwurfsvergleich mit Änderungssimulation. |                       |                            |

## 8.4 Implementierungswerkstatt: Polymorphie im aufrufenden Code

### Syntaxhilfe Java · Der Client kennt nur den gemeinsamen Vertrag

```
public final class FeedbackDienst {
    private final FeedbackStrategie strategie;

    public FeedbackDienst(FeedbackStrategie strategie) {
        this.strategie = strategie;
    }

    public String gibFeedback(Bearbeitung bearbeitung) {
        return strategie.erzeugeFeedback(bearbeitung);
    }
}
```

### [OOP-K08-A04] · IMPLEMENTIEREN · Austauschbares Verhalten ohne Typabfrage implementieren

1. **PROJEKTGRUNDLAGE** Verwende K08\_Feedback\_und\_Vereinsrollen\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie PolymorphieAbnahme aus.
2. Implementiere FeedbackDienst so, dass er ausschließlich von FeedbackStrategie abhängt.
3. Übergib nacheinander die motivierende und die

prüfungsnahe Strategie über den Konstruktor.

4. Schreibe denselben Client-Test für beide Strategien und prüfe das jeweils erwartete Verhalten.
5. Ergänze eine dritte Strategie, ohne FeedbackDienst zu ändern.
6. Entferne vorhandene instanceof-, switch- oder Typcode-Abfragen und erkläre, wodurch sie ersetzt wurden.

Erwartetes Lernprodukt: Polymorpher Client, drei Strategien, gleichartig aufgebaute Abnahmeszenarien und Änderungsnachweis.

**Möglicher Startprompt:** Prüfe nur meine Abhängigkeitsrichtung und suche nach Typabfragen. Gib mir keine weitere Strategieklassse vor.

## 8.5 Ein Prinzip – drei Sprachformen

### Python · Polymorphie über gleiches Verhalten

```
# Python: entscheidend ist der erfüllte Vertrag
class MotivierendesFeedback:
    def erzeuge_feedback(self, bearbeitung):
        return "Stark!" if bearbeitung.erfolgreich else
        "Prüfe deinen Weg."
```

### PHP · Explizites Interface wie im Java-Modell

```
<?php
interface FeedbackStrategie {
    public function erzeugeFeedback(Bearbeitung
    $bearbeitung): string;
}
```

### [OOP-K08-A05] · SPRACHTRANSFER · Vertrag in Python und PHP wiedererkennen

1. **PROJEKTGRUNDLAGE** Verwende K08\_Feedback\_und\_Vereinsrollen\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die

- freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie PolymorphieAbnahme aus.
2. Markiere den gemeinsamen fachlichen Vertrag in allen drei Sprachen.
  3. Trenne Unterschiede der Typprüfung von Unterschieden des Objektmodells.
  4. Ergänze jeweils eine zweite Strategie und prüfe denselben Clientgedanken.
  5. Begründe, warum Polymorphie nicht an das Schlüsselwort interface gebunden ist.

**Erwartetes Lernprodukt:** Eine kommentierte Dreisprachen-Gegenüberstellung mit gleichem Entwurfskern.

## 8.6 Transfer zum Lernagenten

### [OOP-K08-A06] · TRANSFER LERNAGENT · Austauschbare Lernstrategien

1. **PROJEKTGRUNDLAGE** Verwende K08\_Feedback\_und\_Vereinsrollen\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie PolymorphieAbnahme aus.
2. Definiere einen Vertrag für die Auswahl des nächsten Lernschritts.
3. Implementiere eine wiederholende und eine herausfordernde Auswahlstrategie.
4. Übergib die Strategie von außen an den Lernpfad, statt sie dort fest zu erzeugen.
5. Teste beide Varianten mit demselben Aufgabenpool.
6. Dokumentiere, welche Erweiterung nun ohne Änderung am Lernpfad möglich ist.

**Erwartetes Lernprodukt:** Ein modularer Lernpfad mit austauschbarer Auswahlstrategie.

**Möglicher Startprompt:** Prüfe mit Rückfragen, ob meine

Abhängigkeiten in die richtige Richtung zeigen. Nenne das passende Entwurfsmuster erst am Ende.

## 8.7 Kapitel-Sicherung

HALTE FEST · Abstraktion, Vererbung und Polymorphie. Zeige den Austausch einer Implementierung ohne Typabfrage.

MINI-CHECK

1. Erkläre den Kernbegriff in höchstens zwei Sätzen.
2. Zeige ihn an einem konkreten Code-, Modell- oder Testbeleg.
3. Nenne eine typische Fehlentscheidung und ihre erkennbare Folge.

GESTUFTE HILFE

Denkimpuls: Welche fachliche Verantwortung steht im Mittelpunkt?

Konkreter Hinweis: Zeige den Austausch einer Implementierung ohne Typabfrage.

Teilstruktur: Ausgangslage → Entwurfsentscheidung → Implementierung/Modell → Test → Änderungsfolge.

[Zurück zum Inhaltsverzeichnis](#)

## 9 · MVC: Oberfläche und Fachmodell trennen ↵ Inhalt

**DEIN KAPITELSTART** Du leitest aus einem grafischen Mockup Ereignisse ab und verteilst Anzeige, Ablaufsteuerung und Fachlogik so, dass jede Schicht unabhängig getestet und ausgetauscht werden kann.

AKTIVIERUNGS-MOCKUP · BMI-Coach

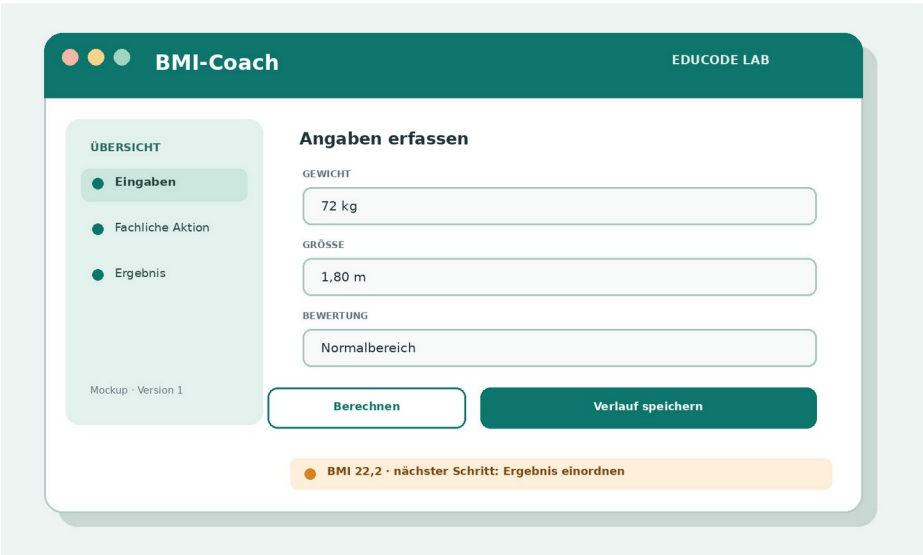


Abb.: Grafischer Oberflächenentwurf als Ausgangspunkt für Anforderungsanalyse und Objektmodellierung.

9.1 Vom Button-Ereignis zur Verantwortungsverteilung

Das Mockup zeigt, was Menschen bedienen. Es legt aber nicht fest, wo Berechnung, Eingabeprüfung oder Darstellung implementiert werden. Im Model-View-Controller-Prinzip bleibt das Modell unabhängig von Fenstern, Webseiten und Konsolenausgaben.

| Baustein   | Verantwortung          | kennt                  | kennt nicht         |
|------------|------------------------|------------------------|---------------------|
| Model      | Fachzustand und Regeln | Fachobjekte            | Buttons, Textfelder |
| View       | Ein- und Ausgabe       | darstellbare Werte     | Berechnungsdetails  |
| Controller | Ablauf koordinieren    | Model und View-Vertrag | SQL, Layoutdetails  |

[OOP-K09-A01] · KERNPFAD · MVC im Mockup markieren

- 1. Markiere Eingaben, Ereignisse und Ausgaben im BMI-Mockup.
- 2. Ordne jede Fachregel Model, View oder Controller zu.

| Baustein                                                                                                                                                                                                                        | Verantwortung                                                  | kennt | kennt nicht |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------|-------|-------------|
| 3.                                                                                                                                                                                                                              | Begründe zwei Zuordnungen mit dem erwarteten Änderungsgrund.   |       |             |
| 4.                                                                                                                                                                                                                              | Entwirf ein UML-Komponentendiagramm ohne GUI-Frameworkklassen. |       |             |
| <p><b>Erwartetes Lernprodukt:</b> Annotiertes Mockup und begründete MVC-Komponentenkarte.</p> <p><b>Möglicher Startprompt:</b> Frage mich bei jeder Zuordnung: Ändert sich das wegen Fachlichkeit, Darstellung oder Ablauf?</p> |                                                                |       |             |

## 9.2 Implementierungswerkstatt: MVC verdrahten

### Syntaxhilfe Java · Controller arbeitet nur gegen Modell und View-Vertrag

```

public interface BmiView {
    double liesGewichtKg();
    double liesGroesseM();
    void zeigeErgebnis(double bmi);
    void zeigeFehler(String meldung);
}

public final class BmiController {
    private final BmiRechner model;
    private final BmiView view;

    public BmiController(BmiRechner model, BmiView view) {
        this.model = model;
        this.view = view;
    }

    public void berechnen() {
        try {
            view.zeigeErgebnis(model.berechne(
                view.liesGewichtKg(),
                view.liesGroesseM()));
        } catch (IllegalArgumentException ex) {
            view.zeigeFehler("Bitte Eingaben prüfen.");
        }
    }
}

```

}

### [OOP-K09-A02] · IMPLEMENTIEREN · Einen testbaren Controller implementieren

1. **PROJEKTGRUNDLAGE** Verwende K09\_BMI\_Controller\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie ControllerAbnahme aus.
2. Implementiere BmiView, BmiController und eine Konsolen- oder Swing-View.
3. Teste den Controller mit einer Fake-View für gültige und ungültige Eingaben.
4. Ergänze zurücksetzen(), ohne BmiRechner zu verändern.
5. Prüfe KI-Code auf GUI-Zugriffe im Modell und konkrete View-Erzeugung im Controller.

**Erwartetes Lernprodukt:** MVC-Prototyp, Fake-View und mindestens vier Controller-Tests.

**Möglicher Startprompt:** Gib mir nur die benötigten View-Methoden als Gerüst. Lass mich die Ereignisfolge selbst formulieren.

## 9.3 Implementierungswerkstatt: Beobachtbare Zustände

### Syntaxhilfe Java · Eine Oberfläche meldet ein Ereignis über einen Vertrag

```
public interface ErgebnisBeobachter {
    void wurdeBerechnet(double bmi);
}

public final class BmiVerlauf implements ErgebnisBeobachter
{
    private final List<Double> werte = new ArrayList<>();
}
```

```
@Override
public void wurdeBerechnet(double bmi) {
    werte.add(bmi);
}
}
```

### BKW12-AUSBAU · Eine zweite Reaktion ergänzen

1. Registriere neben der Ergebnisanzeige einen Verlauf als Beobachter.
2. Teste beide Reaktionen unabhängig von einer grafischen Oberfläche.
3. Ergänze einen Protokoll-Beobachter, ohne vorhandene Beobachter zu ändern.
4. Begründe, wann ein direkter Methodenaufruf verständlicher wäre.

**Erwartetes Lernprodukt:** Zwei austauschbare Reaktionen, Tests und eine Entwurfsbegründung.

**Möglicher Startprompt:** Prüfe mit mir zuerst, ob wirklich mehrere unabhängige Reaktionen benötigt werden.

## 9.4 Transfer zum Lernagenten

### [OOP-K09-A03] · TRANSFER LERNAGENT · Lernagenten-Oberfläche entkoppeln

1. **PROJEKTGRUNDLAGE** Verwende K09\_BMI\_Controller\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie ControllerAbnahme aus.
2. Entwirf ein View-Interface für Aufgabe, Hinweis und Rückmeldung.
3. Implementiere einen Controller für AufgabeAnfordern() und antwortPruefen().
4. Teste den Ablauf mit einer Fake-View und einer Fake-Strategie.

5. Tausche die Oberfläche aus, ohne das Lernmodell zu ändern.

**Erwartetes Lernprodukt:** Bedienbarer Lernagenten-Prototyp mit Controller-Tests.

**Möglicher Startprompt:** Lass mich zuerst den Ablauf als Ereignisfolge beschreiben; nenne danach fehlende Verträge.

## 9.5 GUI-Werkstatt: Designer, Ereignisse und Typumwandlung

### EINORDNUNG

Ein GUI-Designer wie WindowBuilder kann Oberflächencode erzeugen. Der generierte Code ersetzt weder das Fachmodell noch die bewusste Ereignisverarbeitung.

Maus- und Tastaturereignisse werden über Listener behandelt. Texteingaben müssen vor der Übergabe an das Modell konvertiert und validiert werden.

BG: Modell und Präsentation bilden mindestens zwei klar getrennte Schichten. BKWI2: MVC differenziert die Präsentationsschicht zusätzlich in View und Controller.

```
button.addActionListener(e -> {
    try {
        double groesse = Double.parseDouble(tfGroesse.getText());
        controller.berechnen(groesse);
    } catch (NumberFormatException ex) {
        view.zeigeFehler("Bitte eine Zahl eingeben.");
    }
});
```

### [OOP-K09-A04] · IMPLEMENTIEREN · Ereignis sicher verarbeiten

**PROJEKTGRUNDLAGE** Verwende K09\_BMI\_Controller\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie ControllerAbnahme aus.

1. Erzeuge ein Fenster mit Label, Textfeld, Button und Ausgabefeld.

2. Verarbeite den Buttonklick über einen Listener; ergänze zusätzlich ein Tastaturereignis für Enter.
  3. Konvertiere die Eingabe mit `parseDouble` und behandle `NumberFormatException`.
  4. Halte die Fachberechnung außerhalb der View.
- Prüfkriterien: gültige Eingabe funktioniert · ungültige Eingabe beendet das Programm nicht · View enthält keine Fachformel.

## 9.6 Kapitel-Sicherung

HALTE FEST · MVC und Ereignisse. Verfolge einen Buttonklick von der View bis zum getesteten Modell.

MINI-CHECK

1. Erkläre den Kernbegriff in höchstens zwei Sätzen.
2. Zeige ihn an einem konkreten Code-, Modell- oder Testbeleg.
3. Nenne eine typische Fehlentscheidung und ihre erkennbare Folge.

GESTUFTE HILFE

Denkimpuls: Welche fachliche Verantwortung steht im Mittelpunkt?  
Konkreter Hinweis: Verfolge einen Buttonklick von der View bis zum getesteten Modell.

Teilstruktur: Ausgangslage → Entwurfsentscheidung → Implementierung/Modell → Test → Änderungsfolge.

[Zurück zum Inhaltsverzeichnis](#)

# 10 · Daten verwalten und persistieren ↵

Inhalt

**DEIN KAPITELSTART** Du unterscheidest Fachobjekt, Transferdarstellung und Speichertechnik. Das Modell bleibt verständlich, auch wenn Daten künftig als Datei, JSON oder in einer Datenbank gespeichert werden.

## 10.1 Persistenz ist eine eigene Verantwortung

Ein Objektmodell beantwortet fachliche Fragen. Ein Speicherformat beantwortet technische Fragen. Werden Dateipfade, JSON-

Feldnamen und Fachregeln in derselben Klasse vermischt, wird jede Änderung unnötig groß.

| Ebene         | Beispiel            | Änderungsgrund                        |
|---------------|---------------------|---------------------------------------|
| Domänenobjekt | Spieler             | Vereinsregel ändert sich              |
| DTO/Datensatz | SpielerDaten        | Speicherformat ändert sich            |
| Adapter       | JsonSpielerSpeicher | Bibliothek oder Dateiziel ändert sich |

### [OOP-K10-A01] · KERNPFAD · Speichergrenzen modellieren

1. Markiere im Teammanager fachliche und technische Daten.
2. Entwirf SpielerDaten als unveränderlichen Transferdatensatz.
3. Lege die Abbildung zwischen Spieler und SpielerDaten fest.
4. Begründe, welche Validierung beim Einlesen erneut ausgeführt werden muss.

**Erwartetes Lernprodukt:** Mapping-Skizze mit Fachregeln und Speichergrenze.

**Möglicher Startprompt:** Frage mich für jedes Feld, ob es fachliche Bedeutung oder nur technische Darstellung besitzt.

## 10.2 Implementierungswerkstatt: Speichervertrag

### Syntaxhilfe Java · Speicherzugriff hinter einem Interface

```
public interface SpielerSpeicher {
    void speichere(Spieler spieler);
    Optional<Spieler> finde(int nummer);
    List<Spieler> findeAlle();
}

public final class InMemorySpielerSpeicher implements
SpielerSpeicher {
    private final Map<Integer, Spieler> daten = new
HashMap<>();
}
```

```

    public void speichere(Spieler spieler) {
        daten.put(spieler.nummer(), spieler);
    }

    public Optional<Spieler> finde(int nummer) {
        return Optional.ofNullable(daten.get(nummer));
    }

    public List<Spieler> findeAlle() {
        return List.copyOf(daten.values());
    }
}

```

### [OOP-K10-A02] · IMPLEMENTIEREN · Einen austauschbaren Speicher implementieren

1. **PROJEKTGRUNDLAGE** Verwende K10\_K11\_Speicher\_und\_Schnittstellen\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie SpeicherAbnahme aus.
2. Implementiere SpielerSpeicher und die In-Memory-Variante.
3. Teste Speichern, Finden, Überschreiben und eine leere Treffermenge.
4. Ergänze loesche(int nummer) samt Vertrag und Tests.
5. Bewerte KI-Code auf veränderbare Rückgaben und versteckte globale Daten.

**Erwartetes Lernprodukt:** Speicherinterface, In-Memory-Adapter und mindestens fünf Tests.

**Möglicher Startprompt:** Zeige mir nur den Vertrag. Frage danach, welche Fälle beobachtbar getestet werden müssen.

## 10.3 Implementierungswerkstatt: JSON und Mapping

### Syntaxhilfe Java · DTO und Mapper schützen das Fachmodell

```
public record SpielerDaten(int nummer, String name, String position) {}

public final class SpielerMapper {
    public SpielerDaten nachDaten(Spieler spieler) {
        return new SpielerDaten(
            spieler.nummer(), spieler.name(),
            spieler.position());
    }

    public Spieler nachModell(SpielerDaten daten) {
        return new Spieler(
            daten.nummer(), daten.name(),
            daten.position());
    }
}
```

#### [OOP-K10-A03] · IMPLEMENTIEREN · JSON-Persistenz ergänzen

1. **PROJEKTGRUNDLAGE** Verwende K10\_K11\_Speicher\_und\_Schnittstellen\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie SpeicherAbnahme aus.
2. Implementiere SpielerMapper und teste beide Abbildungsrichtungen.
3. Speichere eine Liste von SpielerDaten als JSON im Starterprojekt.
4. Lies die Datei ein und rekonstruiere gültige Spielerobjekte.
5. Behandle fehlende, beschädigte und fachlich ungültige Daten mit klaren Fehlern.

**Erwartetes Lernprodukt:** JSON-Datei, Mapper, Adapter sowie Normal- und Fehlertests.

**Möglicher Startprompt:** Hilf mir zuerst, Domänenfehler von Datei- und Formatfehlern zu unterscheiden; gib keinen fertigen Adapter aus.

## 10.4 Sprachtransfer und Lernagent

### TRANSFER · Dasselbe Speichermodell wiedererkennen

1. Übertrage nur SpielerDaten und Mapper in Python oder PHP.
2. Vergleiche Typprüfung, Fehlerbehandlung und JSON-Abbildung.
3. Implementiere anschließend LernzielSpeicher in Java.
4. Zeige durch einen Test, dass der Lernpfad den konkreten Speicher nicht kennt.

**Erwartetes Lernprodukt:** Sprachvergleich und austauschbarer Lernziel-Speicher.

**Möglicher Startprompt:** Prüfe nur, ob Fachmodell, DTO und Adapter weiterhin getrennte Rollen besitzen.

## 10.5 Dateiwerkstatt: FileWriter und FileReader

```
try (FileWriter writer = new FileWriter(pfad)) {
    writer.write(mitglied.toCsv());
}

try (FileReader reader = new FileReader(pfad)) {
    int zeichen;
    while ((zeichen = reader.read()) != -1) {
        // Daten kontrolliert verarbeiten
    }
}
```

### [OOP-K10-A04] · IMPLEMENTIEREN · Mitglieder offline speichern

**PROJEKTGRUNDLAGE** Verwende

K10\_K11\_Speicher\_und\_Schnittstellen\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie SpeicherAbnahme aus.

1. Implementiere speichern(Path, List<Mitglied>) und laden(Path).
2. Nutze try-with-resources und behandle I/O-Fehler an einer sinnvollen Systemgrenze.

3. Prüfe leere Datei, zwei Datensätze und eine fehlerhafte Zeile.

Prüfkriterien: Ressourcen werden sicher geschlossen · Reihenfolge bleibt erhalten · fehlerhafte Daten führen zu einer verständlichen Rückmeldung.

## 10.6 Kapitel-Sicherung

HALTE FEST · Persistenz und Mapping. Trenne Domänenobjekt, Speicherformat und Adapter.

### MINI-CHECK

1. Erkläre den Kernbegriff in höchstens zwei Sätzen.
2. Zeige ihn an einem konkreten Code-, Modell- oder Testbeleg.
3. Nenne eine typische Fehlentscheidung und ihre erkennbare Folge.

### GESTUFTE HILFE

Denkimpuls: Welche fachliche Verantwortung steht im Mittelpunkt?

Konkreter Hinweis: Trenne Domänenobjekt, Speicherformat und Adapter.

Teilstruktur: Ausgangslage → Entwurfsentscheidung →

Implementierung/Modell → Test → Änderungsfolge.

[Zurück zum Inhaltsverzeichnis](#)

## 11 · Datenbanken und Services anbinden ↔ Inhalt

**DEIN KAPITELSTART** Du bindest relationale Datenbanken und externe Services über schmale Schnittstellen an. SQL, HTTP und Zugangsdaten bleiben außerhalb der Fachklassen.

## 11.1 Repository und DAO als Systemgrenze

Ein Repository formuliert fachlich benötigte Speicheroperationen. Ein DAO oder Adapter setzt diese Operationen mit JDBC, einer API oder einer anderen Technik um. Entscheidend ist die Abhängigkeitsrichtung: Die Anwendung kennt den Vertrag, nicht die technische Variante.

### [OOP-K11-A01] · KERNPFAD · Abfragen vom Fachbedarf her entwerfen

1. Formuliere die benötigten Suchoperationen der Vereinsverwaltung.
2. Trenne fachliche Suchbegriffe von Tabellen- und Spaltennamen.
3. Entwirf ein MitgliedRepository mit kleinen Rückgabetypen.
4. Begründe, wann Optional und wann eine Liste passt.

**Erwartetes Lernprodukt:** Repository-Vertrag mit begründeten Signaturen.

**Möglicher Startprompt:** Frage mich zuerst nach den Anwendungsfällen und erst danach nach SQL.

## 11.2 Implementierungswerkstatt: JDBC sicher kapseln

### Syntaxhilfe Java · PreparedStatement, try-with-resources und Fehlerübersetzung

```
public Optional<Mitglied> findeNachnummer(String nummer) {
    String sql = "SELECT nummer, name FROM mitglied WHERE
nummer = ?";
    try (Connection con = datenquelle.getConnection();
        PreparedStatement ps = con.prepareStatement(sql))
    {
        ps.setString(1, nummer);
        try (ResultSet rs = ps.executeQuery()) {
            if (!rs.next()) return Optional.empty();
            return Optional.of(new Mitglied(
                rs.getString("nummer"),
```

```

rs.getString("name")));
    }
} catch (SQLException ex) {
    throw new SpeicherzugriffException(
        "Mitglied konnte nicht geladen werden", ex);
}
}

```

## [OOP-K11-A02] · IMPLEMENTIEREN · Ein JDBC-Repository implementieren

1. **PROJEKTGRUNDLAGE** Verwende K10\_K11\_Speicher\_und\_Schnittstellen\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie SpeicherAbnahme aus.
2. Implementiere findeNachNummer(...) mit PreparedStatement und Ressourcenfreigabe.
3. Ergänze speichere(...) mit Behandlung doppelter Nummern.
4. Teste mit einer Testdatenbank Normal-, Leer- und Fehlerfall.
5. Prüfe KI-Code auf Stringverkettung, offene Ressourcen und durchgereichte SQL-Fehler.

**Erwartetes Lernprodukt:** JDBC-Adapter, Datenbankschema und mindestens vier Integrationstests.

**Möglicher Startprompt:** Zeige mir zunächst nur SQL mit Platzhaltern und frage nach dem Verhalten bei keinem oder mehreren Treffern.

## 11.3 Implementierungswerkstatt: Service-Adapter

### Syntaxhilfe Java · Externer Service hinter einem Anwendungsvertrag

```

public interface AufgabenQuelle {
    List<Aufgabe> ladeFuer(Lernziel ziel);
}

```

```

public final class Aufgabenservice implements
AufgabenQuelle {
    private final HttpClient client;
    private final URI basisUri;

    public Aufgabenservice(HttpClient client, URI basisUri)
    {
        this.client = client;
        this.basisUri = basisUri;
    }

    @Override
    public List<Aufgabe> ladeFuer(Lernziel ziel) {
        // Request senden, Status prüfen, DTO lesen und
        abbilden
        throw new UnsupportedOperationException("TODO");
    }
}

```

### [OOP-K11-A03] · IMPLEMENTIEREN · Eine externe Aufgabenquelle anbinden

1. **PROJEKTGRUNDLAGE** Verwende K10\_K11\_Speicher\_und\_Schnittstellen\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie SpeicherAbnahme aus.
2. Vervollständige Anfrage, Statusprüfung, DTO-Mapping und Timeout.
3. Teste mit einem lokalen Fake-Server oder einer Testantwort.
4. Ergänze eine Offline-Quelle, ohne den Lernpfad zu ändern.
5. Bewerte KI-Code auf Netzwerkzugriffe in Fachobjekten und ungeprüfte Antworten.

**Erwartetes Lernprodukt:** Service- und Offline-Adapter mit Vertrags- und Fehlertests.

**Möglicher Startprompt:** Frage mich zuerst, welche Fehler der Anwendung fachlich sichtbar sein sollen.

## 11.4 Transfer zum Lernagenten

### [OOP-K11-A04] · TRANSFER LERNAGENT · Speicher- und Servicevarianten austauschen

1. Verdrahte LernzielRepository und AufgabenQuelle über Konstruktoren.
2. Starte den Agenten einmal offline und einmal mit Service-Adapter.
3. Simuliere Datenbank- und Netzwerkausfall mit Fakes.
4. Dokumentiere, welche Klassen in beiden Betriebsarten unverändert bleiben.

**Erwartetes Lernprodukt:** Zwei lauffähige Konfigurationen und ein Architekturtest.

**Möglicher Startprompt:** Prüfe meine Abhängigkeitsrichtung; nenne keine konkrete Frameworklösung.

## 11.5 JDBC-Werkstatt: Verbindung, CRUD und ResultSet

```
String sql = "SELECT id, name FROM mitglied WHERE aktiv = ?";
try (Connection con = dataSource.getConnection();
    PreparedStatement ps = con.prepareStatement(sql)) {
    ps.setBoolean(1, true);
    try (ResultSet rs = ps.executeQuery()) {
        while (rs.next()) {
            ergebnis.add(new Mitglied(rs.getInt("id"),
rs.getString("name")));
        }
    }
}
```

### [OOP-K11-A05] · IMPLEMENTIEREN · Vollständiges CRUD-Repository

**PROJEKTGRUNDLAGE** Verwende

K10\_K11\_Speicher\_und\_Schnittstellen\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie SpeicherAbnahme aus.

1. Implementiere findAll, findById, insert, update und delete mit PreparedStatement.
  2. Kapsle Treiber/Verbindungsaufbau und mappe ResultSet-Zeilen in Domänenobjekte.
  3. Prüfe jeden CRUD-Fall sowie eine nicht vorhandene ID.
- Prüfkriterien: SQL und Parameter sind getrennt · Ressourcen werden geschlossen · ResultSet wird in einen Container übertragen · UI kennt kein SQL.

## 11.6 Kapitel-Sicherung

HALTE FEST · Repository, JDBC und Service. Zeige, dass die Fachlogik weder SQL noch HTTP kennt.

### MINI-CHECK

1. Erkläre den Kernbegriff in höchstens zwei Sätzen.
2. Zeige ihn an einem konkreten Code-, Modell- oder Testbeleg.
3. Nenne eine typische Fehlentscheidung und ihre erkennbare Folge.

### GESTUFTE HILFE

Denkimpuls: Welche fachliche Verantwortung steht im Mittelpunkt?  
Konkreter Hinweis: Zeige, dass die Fachlogik weder SQL noch HTTP kennt.

Teilstruktur: Ausgangslage → Entwurfsentscheidung → Implementierung/Modell → Test → Änderungsfolge.

[Zurück zum Inhaltsverzeichnis](#)

## 12 · Sichere Software und Datensicherheit ↵ Inhalt

**DEIN KAPITELSTART** Du behandelst Sicherheit nicht als Zusatzfunktion, sondern als überprüfbare Systemeigenschaft: Eingaben, Berechtigungen, Daten, Fehler und technische Grenzen werden bewusst geschützt.

## 12.1 Bedrohungen aus dem Anwendungsfall ableiten

| Schutzgut     | möglicher Missbrauch    | Schutzprinzip           | Nachweis        |
|---------------|-------------------------|-------------------------|-----------------|
| Lernverlauf   | fremde Ergebnisse lesen | Autorisierung           | Negativtest     |
| Datenbank     | manipulierte Eingabe    | parametrisierte Abfrage | Injection-Test  |
| Zugangsdaten  | Secret im Repository    | externe Konfiguration   | Repository-Scan |
| Fehlerdetails | Stacktrace in UI        | sichere Fehlermeldung   | Oberflächentest |

### [OOP-K12-A01] · KERNPFAD · Ein kleines Bedrohungsmodell erstellen

1. Markiere Schutzgüter und Systemgrenzen im Lernagenten-Mockup.
2. Formuliere vier Missbrauchsszenarien aus Sicht verschiedener Rollen.
3. Ordne Prävention, Erkennung und Wiederherstellung zu.
4. Priorisiere nach möglichem Schaden und Eintrittswahrscheinlichkeit.

**Erwartetes Lernprodukt:** Bedrohungsmodell und priorisierte Sicherheitsziele.

**Möglicher Startprompt:** Stelle mir nacheinander Fragen zu Schutzgut, Angreifer, Zugang, Folge und Nachweis.

## 12.2 Implementierungswerkstatt: Autorisierung

### Syntaxhilfe Java · Berechtigung wird serverseitig vor dem Zugriff geprüft

```
public enum Rolle { LERNENDE, LEHRKRAFT }

public final class Berechtigungsdienst {
    public void pruefeEinsicht(
        Benutzer anfragender, Lernverlauf verlauf) {
        boolean eigenerVerlauf =
            verlauf.gehoertZu(anfragender.id());
        boolean lehrkraft = anfragender.rolle() ==
```

```

Rolle.LEHRKRAFT;
    if (!eigenerVerlauf && !lehrkraft) {
        throw new ZugriffVerweigertException();
    }
}
}

```

## [OOP-K12-A02] · IMPLEMENTIEREN · Rollen und Objektzugriff absichern

1. **PROJEKTGRUNDLAGE** Verwende K12\_Sicherheitslabor\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie SicherheitsAbnahme aus.
2. Implementiere die Einsichtsregel und passende Domänenfehler.
3. Teste eigenen, fremden und Lehrkraftzugriff sowie fehlende Identitäten.
4. Ergänze eine Löschregel nach dem Prinzip minimaler Berechtigung.
5. Prüfe KI-Code darauf, ob nur Buttons versteckt oder Zugriffe wirklich geschützt werden.

**Erwartetes Lernprodukt:** Berechtigungsdienst, Zugriffsmatrix und mindestens fünf Negativtests.

**Möglicher Startprompt:** Gib mir keine if-Bedingung vor. Lass mich zuerst die Zugriffsmatrix formulieren.

## 12.3 Implementierungswerkstatt: Secrets und Fehler

### Syntaxhilfe Java · Geheimnisse extern halten und Fehler ohne Datenabfluss behandeln

```

String dbUrl = System.getenv("APP_DB_URL");
if (dbUrl == null || dbUrl.isBlank()) {
    throw new IllegalStateException("Datenbank ist nicht konfiguriert");
}

```

```

}

try {
    repository.speichere(verlauf);
} catch (SpeicherzugriffException ex) {
    logger.error("Speichern fehlgeschlagen, vorgang={}",
        vorgangsId);
    view.zeigeFehler("Speichern derzeit nicht möglich.");
}

```

### [OOP-K12-A03] · IMPLEMENTIEREN · Informationsabfluss verhindern

1. **PROJEKTGRUNDLAGE** Verwende K12\_Sicherheitslabor\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie SicherheitsAbnahme aus.
2. Lade Konfiguration aus der Umgebung und entferne Secrets aus dem Quellcode.
3. Entwirf sichere Meldungen für Oberfläche und Protokoll.
4. Teste, dass Passwort, SQL und personenbezogene Daten nicht ausgegeben werden.
5. Untersuche einen KI-Vorschlag auf übermäßiges Logging und verschluckte Fehler.

**Erwartetes Lernprodukt:** Konfigurationsklasse, sichere Fehlerbehandlung und Log-Prüftests.

**Möglicher Startprompt:** Frage mich bei jeder Information, wer sie für welchen Zweck wirklich sehen muss.

## 12.4 Transfer zum Lernagenten

### [OOP-K12-A04] · TRANSFER LERNAGENT · Security-Review des bisherigen Systems

1. **PROJEKTGRUNDLAGE** Verwende K12\_Sicherheitslabor\_MVC aus

- 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie SicherheitsAbnahme aus.
2. Prüfe Eingaben, Speicher, Service und Oberfläche an allen Systemgrenzen.
  3. Ergänze je einen Negativtest für Manipulation, Ausfall und unzulässigen Zugriff.
  4. Dokumentiere verbleibende Risiken und bewusste Grenzen.
  5. Leite zwei Sicherheitsanforderungen für das Abschlussprojekt ab.

**Erwartetes Lernprodukt:** Security-Checkliste, Negativtests und Rest-Risiko-Protokoll.

**Möglicher Startprompt:** Führe mich systematisch durch Datenfluss, Vertrauensgrenze und beobachtbares Fehlverhalten.

## 12.5 Schutzwerkstatt: Hashing ist keine Verschlüsselung

### SICHERHEITSWISSEN

Verschlüsselung ist mit einem Schlüssel umkehrbar und schützt vertrauliche Daten bei Speicherung oder Übertragung. Hashing ist eine Einwegabbildung und eignet sich mit individuellem Salt und geeignetem Passwort-Hashverfahren zur Passwortprüfung.

Allgemeine Hashfunktionen wie SHA-256 allein sind für Passwörter nicht ausreichend. Im Projekt wird ein etabliertes Passwort-Hashverfahren beziehungsweise eine geprüfte Bibliothek verwendet – kein selbst erfundener Algorithmus.

### [OOP-K12-A05] · ANALYSIEREN UND IMPLEMENTIEREN · Sicherer Zugang

**PROJEKTGRUNDLAGE** Verwende K12\_Sicherheitslabor\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und

führe AppMain sowie SicherheitsAbnahme aus.

1. Ordne vier Fälle zu: Transportverschlüsselung, verschlüsselte Datei, Passwortprüfung, Integritätsprüfung.
  2. Ersetze in einem vorgegebenen Login-Modul Klartextpasswörter durch eine Passwort-Hashbibliothek.
  3. Dokumentiere Salt, Verifikation und Fehlerbehandlung; protokolliere niemals Passwörter oder Hashwerte.
- Prüfkriterien: Verfahren fachlich korrekt zugeordnet · kein Klartextpasswort gespeichert · falsches Passwort wird ohne Informationsleck abgewiesen.

## 12.6 Kapitel-Sicherung

HALTE FEST · Sichere Systemgrenzen. Teste Normalfall, Missbrauch und Informationsabfluss.

MINI-CHECK

1. Erkläre den Kernbegriff in höchstens zwei Sätzen.
2. Zeige ihn an einem konkreten Code-, Modell- oder Testbeleg.
3. Nenne eine typische Fehlentscheidung und ihre erkennbare Folge.

GESTUFTE HILFE

Denkimpuls: Welche fachliche Verantwortung steht im Mittelpunkt?

Konkreter Hinweis: Teste Normalfall, Missbrauch und Informationsabfluss.

Teilstruktur: Ausgangslage → Entwurfsentscheidung → Implementierung/Modell → Test → Änderungsfolge.

[Zurück zum Inhaltsverzeichnis](#)

## 13 · Softwarequalität, Refactoring und Erweiterung ↩ Inhalt

**DEIN KAPITELSTART** Du verbesserst vorhandenen Code in kleinen, abgesicherten Schritten. Tests bewahren Verhalten; klare Verantwortlichkeiten und Schnittstellen begrenzen künftige Änderungen.

## 13.1 Code-Smells als Änderungshindernisse

| Beobachtung       | Risiko                             | mögliche Richtung      |
|-------------------|------------------------------------|------------------------|
| lange Methode     | Regeln schwer isolierbar           | Methoden extrahieren   |
| Typabfragen       | jede Variante ändert eine Zentrale | Polymorphie            |
| globale Daten     | Tests beeinflussen sich            | Abhängigkeit übergeben |
| duplizierte Regel | Varianten laufen auseinander       | Verantwortung bündeln  |

### [OOP-K13-A01] · KERNPFAD · Änderungsrisiken diagnostizieren

1. Markiere in einem vorgegebenen Fremdcode vier konkrete Code-Smells.
2. Ordne jedem Fund ein mögliches Änderungsszenario zu.
3. Priorisiere nach Risiko statt nach persönlichem Stil.
4. Formuliere vor dem Umbau passende Regressionstests.

**Erwartetes Lernprodukt:** Priorisierter Refactoring-Plan mit Testabsicherung.

**Möglicher Startprompt:** Nenne mir zu jeder Stelle nur eine Frage nach Verantwortung, Abhängigkeit oder Duplikation.

## 13.2 Werkstatt: Typabfrage durch Polymorphie ersetzen

### Syntaxhilfe Java · Refactoringziel aus einem Code-Smell ableiten

```
// vorher: jede neue Art ändert diese Methode
String feedback(AlteAufgabe aufgabe) {
    if (aufgabe.art() == Art.MULTIPLE_CHOICE) return
    "Optionen prüfen";
    if (aufgabe.art() == Art.CODE) return "Test ausführen";
    throw new IllegalArgumentException("Unbekannte Art");
}

// Zielvertrag
public interface Aufgabe {
    String hinweis();
}
```

## [OOP-K13-A02] · IMPLEMENTIEREN · Ein verhaltensbewahrendes Refactoring durchführen

1. **PROJEKTGRUNDLAGE** Verwende K13\_Refactoring\_DI\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie RefactoringAbnahme aus.
2. Sichere das bestehende Verhalten zunächst mit Tests ab.
3. Implementiere zwei Aufgabenarten hinter dem gemeinsamen Vertrag.
4. Entferne die Typabfrage und führe alle Tests erneut aus.
5. Ergänze eine dritte Aufgabenart, ohne bestehenden Client-Code zu ändern.

**Erwartetes Lernprodukt:** Refactoring-Commitfolge, grüne Tests und Änderungsvergleich.

**Möglicher Startprompt:** Lass mich zuerst den Charakterisierungstest schreiben; schlage noch keine Zielklassen vor.

## 13.3 Implementierungswerkstatt: Abhängigkeiten umkehren

### Syntaxhilfe Java · Fachservice erhält technische Abhängigkeiten von außen

```
public final class LernpfadService {
    private final LernzielRepository ziele;
    private final AufgabenQuelle aufgaben;

    public LernpfadService(
        LernzielRepository ziele, AufgabenQuelle
aufgaben) {
        this.ziele = ziele;
        this.aufgaben = aufgaben;
    }
}
```

```
}
```

### [OOP-K13-A03] · IMPLEMENTIEREN · Testbarkeit durch Dependency Injection erhöhen

1. **PROJEKTGRUNDLAGE** Verwende K13\_Refactoring\_DI\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie RefactoringAbnahme aus.
2. Entferne new-Aufrufe technischer Adapter aus einem Fachservice.
3. Übergib beide Verträge über den Konstruktor.
4. Teste den Service mit Fakes für Erfolg und Ausfall.
5. Prüfe KI-Code auf Service-Locator, globale Zustände und unnötige Frameworkbindung.

**Erwartetes Lernprodukt:** Entkoppelter Service, Fakes und mindestens vier Verhaltenstests.

**Möglicher Startprompt:** Frage mich, welche Abhängigkeiten echte Varianten oder Systemgrenzen darstellen.

## 13.4 Qualitätsnachweis und Lernagent

### [OOP-K13-A04] · TRANSFER LERNAGENT · Eine Erweiterung messbar begrenzen

1. **PROJEKTGRUNDLAGE** Verwende K13\_Refactoring\_DI\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie RefactoringAbnahme aus.
2. Wähle eine neue Feedback-, Speicher- oder Aufgabenvariante.
3. Prognostiziere vorab alle notwendigen Änderungen.
4. Implementiere die Variante mit Tests und vergleiche

Prognose und Ergebnis.

5. Dokumentiere Kopplung, Duplikation, Verständlichkeit und verbleibende Schulden.

**Erwartetes Lernprodukt:** Qualitätsbericht mit Änderungsnachweis und Testprotokoll.

**Möglicher Startprompt:** Bewerte meinen Entwurf nicht pauschal. Frage nach konkreten Änderungsstellen und Testbelegen.

## 13.5 Kapitel-Sicherung

HALTE FEST · Refactoring und Änderbarkeit. Belege unverändertes Verhalten vor und nach dem Umbau.

MINI-CHECK

1. Erkläre den Kernbegriff in höchstens zwei Sätzen.
2. Zeige ihn an einem konkreten Code-, Modell- oder Testbeleg.
3. Nenne eine typische Fehlentscheidung und ihre erkennbare Folge.

GESTUFTE HILFE

Denkimpuls: Welche fachliche Verantwortung steht im Mittelpunkt?  
Konkreter Hinweis: Belege unverändertes Verhalten vor und nach dem Umbau.

Teilstruktur: Ausgangslage → Entwurfsentscheidung → Implementierung/Modell → Test → Änderungsfolge.

[Zurück zum Inhaltsverzeichnis](#)

## 14 · Abschlussprojekt: ein langlebiges Softwaresystem [← Inhalt](#)

**DEIN KAPITELSTART** Du führst Mockup, Objektmodell, Implementierung, Tests, Persistenz, Anbindung und Sicherheit in einem nachvollziehbaren Systeminkrement zusammen. Bewertet wird nicht nur, ob es läuft, sondern wie sicher es sich ändern lässt.

## 14.1 Projektwahl und verbindlicher Systemkern

| Projekt           | fachlicher Kern       | Pflichterweiterung             | möglicher Transfer |
|-------------------|-----------------------|--------------------------------|--------------------|
| Lernagent         | Lernpfad und Feedback | neue Strategie plus Persistenz | Service-Quelle     |
| Volleyball        | Team und Aufstellung  | Auswahlstrategie plus Speicher | Vereinsdaten       |
| Vereinsverwaltung | Mitglied und Rollen   | Berechtigung plus Repository   | Beitragssystem     |
| Mini-Shop         | Bestellung und Preis  | Rabattstrategie plus Datenbank | Web-Frontend       |

### [OOP-K14-A01] · KERNPFAD · Projektauftrag präzisieren

1. Wähle einen Kontext und formuliere drei prüfbare Anwendungsfälle.
2. Erstelle ein grafisches Mockup und leite Objektkandidaten ab.
3. Definiere Qualitäts- und Sicherheitsziele vor der Implementierung.
4. Plane Kernpfad und zwei optionale Erweiterungsinkremente.

**Erwartetes Lernprodukt:** Projektsteckbrief, Mockup, Anwendungsfälle und Abnahmekriterien.

**Möglicher Startprompt:** Frage mich nach Nutzenden, sichtbarem Nutzen, späterer Änderung und schützenswerten Daten.

## 14.2 Implementierungswerkstatt: Architekturkeim

### Syntaxhilfe Java · Der Anwendungskern hängt von fachlichen Verträgen ab

```
public final class LernagentAnwendung {
    private final LernzielRepository ziele;
    private final AufgabenQuelle aufgaben;
    private final FeedbackStrategie feedback;

    public LernagentAnwendung(
        LernzielRepository ziele,
```

```

        AufgabenQuelle aufgaben,
        FeedbackStrategie feedback) {
    this.ziele = ziele;
    this.aufgaben = aufgaben;
    this.feedback = feedback;
}
}

```

### [OOP-K14-A02] · IMPLEMENTIEREN · Ein vertikales Inkrement implementieren

1. **PROJEKTGRUNDLAGE** Verwende K14\_Abschlussprojekt\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie AbschlussAbnahme aus.
2. Implementiere einen vollständigen Anwendungsfall vom Mockup bis zum Speicher.
3. Verwende mindestens zwei fachliche Interfaces und eine gekapselte Beziehung.
4. Sichere Normal-, Grenz-, Fehler- und Berechtigungsfall automatisiert ab.
5. Ergänze eine Variante, ohne den Anwendungskern unnötig zu ändern.

**Erwartetes Lernprodukt:** Lauffähiges Inkrement, Quellcode, Tests und Architekturübersicht.

**Möglicher Startprompt:** Gib mir keine Komplettlösung. Prüfe nach jedem Schritt nur Vertrag, Verantwortung und beobachtbares Verhalten.

## 14.3 Werkstatt: KI-Code verantwortungsvoll integrieren

### [OOP-K14-A03] · IMPLEMENTIEREN · Einen KI-Vorschlag prüfen und verbessern

1. **PROJEKTGRUNDLAGE** Verwende

K14\_Abschlussprojekt\_MVC aus

02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie AbschlussAbnahme aus.

2. Lass für eine klar abgegrenzte Methode einen KI-Vorschlag erzeugen.
3. Prüfe Vertrag, Nullfälle, Fehler, Sicherheit und Abhängigkeitsrichtung.
4. Übernimm nur verstandene Teile und ergänze eigene Tests.
5. Dokumentiere Prompt, Änderungen, verworfene Teile und fachliche Begründung.

**Erwartetes Lernprodukt:** Nachvollziehbares KI-Review mit verbessertem Code und Tests.

**Möglicher Startprompt:** Antworte zuerst nur mit Rückfragen und Prüfkriterien; erzeuge Code erst nach meinem eigenen Entwurf.

## 14.4 Abnahme, Präsentation und Transfer

### [OOP-K14-A04] · ABSCHLUSS · Die Änderungsprobe bestehen

1. **PROJEKTGRUNDLAGE** Verwende K14\_Abschlussprojekt\_MVC aus 02\_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie AbschlussAbnahme aus.
2. Ziehe eine unbekannte Zusatzanforderung und prognostiziere betroffene Klassen.
3. Implementiere sie in einem kleinen, getesteten Änderungsinkrement.
4. Führe eine Sicherheits- und Qualitätsprüfung durch.
5. Belege, wie Kapselung, Beziehungen, Abstraktion und Polymorphie die Änderung begrenzen.

**Erwartetes Lernprodukt:** Lauffähiges System,

Testbericht, Änderungsprotokoll und Kurzpräsentation.

**Möglicher Startprompt:** Stelle mir Prüferfragen. Fordere zu jeder Behauptung einen konkreten Code- oder Testbeleg.

## 14.5 Der Lernagent wächst mit

| OOP-Baustein | Beitrag zum Lernagenten                          | sichtbarer Nachweis                |
|--------------|--------------------------------------------------|------------------------------------|
| Kapselung    | Lernzustände bleiben gültig                      | Invarianten und Tests              |
| Beziehungen  | Ziele, Aufgaben und Bearbeitungen sind verbunden | UML und Objektgraph                |
| Polymorphie  | Strategien bleiben austauschbar                  | neue Variante ohne Client-Änderung |
| MVC          | Oberfläche bleibt von Lernlogik getrennt         | Fake-View-Test                     |
| Persistenz   | Speichertechnik bleibt ersetzbar                 | zwei Adapter                       |
| Sicherheit   | Daten und Zugriffe sind begrenzt                 | Negativtests                       |
| Refactoring  | Änderungen bleiben lokal                         | Änderungsprotokoll                 |

**ABSCHLUSSGEDANKE** Die Sprache liefert die Ausdrucksmittel. Langlebig wird Software erst durch bewusst verteilte Verantwortung, überprüfbare Verträge, geschützte Zustände und kleine, austauschbare Bausteine.

## 14.6 Kapitel-Sicherung

HALTE FEST · Gesamtsystem und Änderungsprobe. Begründe jede Architekturentscheidung mit einem Test- oder Änderungsnachweis.

MINI-CHECK

1. Erkläre den Kernbegriff in höchstens zwei Sätzen.
2. Zeige ihn an einem konkreten Code-, Modell- oder Testbeleg.
3. Nenne eine typische Fehlentscheidung und ihre erkennbare Folge.

**GESTUFTE HILFE**

Denkimpuls: Welche fachliche Verantwortung steht im Mittelpunkt?  
 Konkreter Hinweis: Begründe jede Architekturentscheidung mit einem Test- oder Änderungsnachweis.

Teilstruktur: Ausgangslage → Entwurfsentscheidung →  
 Implementierung/Modell → Test → Änderungsfolge.

[Zurück zum Inhaltsverzeichnis](#)

**Anhang: Projekt- und Aufgabenpool**

**EINORDNUNG** Die folgenden Impulse sind offene Wahl- und Transferaufgaben. Sie ersetzen keine ausgewiesenen Prüfungsanker und erhalten deshalb bewusst keine reguläre Aufgaben-ID.

Die folgenden Kontexte sind als wechselnde Transferprojekte vorgesehen. Sie werden nicht alle zu gleich großen Leitprojekten ausgebaut.

| Kontext                | Geeignete OOP-Themen                       | Lernprodukt                 | Rolle                    |
|------------------------|--------------------------------------------|-----------------------------|--------------------------|
| BMI-App                | MVC, Validierung, Tests, Persistenz        | wachsendes Referenzsystem   | durchgängiger Anker      |
| Volleyball-Teammanager | Listen, Schleifen, Suche, Assoziationen    | Team- und Spielerverwaltung | Datenstrukturen          |
| Vereinsverwaltung      | 1:N/M:N, Rollen, Vererbung/Komposition     | Mitglieder, Teams, Beiträge | größerer Transfer        |
| Mini-Shop              | Bestellung, Position, Produkt, Preisregeln | Warenkorb und Bestellung    | Wirtschaftsbezug dosiert |
| Lernagent              | Schnittstellen, Strategien, Persistenz     | modularer Lernpfad          | Zukunftstransfer         |

**Kapitelübersicht 9-14**

- Kapitel 9 · MVC: vom Mockup zur austauschbaren Benutzeroberfläche
- Kapitel 10 · Daten verwalten und persistieren: Dateien, JSON und Objektabbildung
- Kapitel 11 · Datenbanken und Services anbinden: Repository, DAO, JDBC und API
- Kapitel 12 · Sichere Software und Datensicherheit

- Kapitel 13 · Softwarequalität, Refactoring und nachhaltige Erweiterung
- Kapitel 14 · Abschlussprojekt: modularer Lernagent oder eigenes System

## Vertiefungsschwerpunkte

### KAPITEL 10 UND 11 · DATENVERWALTUNG UND

**ANBINDUNG** Die Lernenden unterscheiden fachliche Objekte, gespeicherte Daten und technische Zugriffsschichten. Zuerst werden Objekte lokal in Datei- und JSON-Strukturen überführt. Danach wird der Zugriff über Repository beziehungsweise DAO gekapselt und auf relationale Datenbanken oder einen Service übertragen.

**KAPITEL 12 · SICHERE SOFTWARE** Sicherheit wird als Qualitätsanforderung des gesamten Entwicklungsprozesses behandelt: schützenswerte Daten erkennen, Eingaben validieren, Berechtigungen begrenzen, Geheimnisse schützen, Datenbankzugriffe absichern und Fehler so behandeln, dass weder Daten noch interne Systeminformationen ungewollt offengelegt werden.

## Daten-Karussell: gleiche Architekturfrage, drei Speicherziele

| Station | Kontext           | technischer Fokus          | OOP-Leitfrage                                   |
|---------|-------------------|----------------------------|-------------------------------------------------|
| A       | Volleyball-Team   | Datei/JSON, Serialisierung | Wer übersetzt Objekt und Speicherformat?        |
| B       | Vereinsverwaltung | Repository/DAO und JDBC    | Wie bleibt das Modell von SQL unabhängig?       |
| C       | Lernagent         | Service/API und DTO        | Wie wird eine externe Datenquelle austauschbar? |

### [OOP-ANH-A01] · DATENVERWALTUNG ·

| Station                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Kontext | technischer Fokus | OOP-Leitfrage |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|-------------------|---------------|
| <b>Speicher austauschen, Modell erhalten</b> <ol style="list-style-type: none"> <li><b>PROJEKTGRUNDLAGE</b> Verwende K14_Abschlussprojekt_MVC aus 02_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie AbschlussAbnahme aus.</li> <li>Entwirf zunächst eine Speicher-Schnittstelle für ein fachliches Objekt.</li> <li>Implementiere eine lokale Datei- oder JSON-Variante.</li> <li>Ersetze sie anschließend durch eine Datenbank- oder Service-Variante.</li> <li>Dokumentiere, welche Modellklassen unverändert bleiben und wodurch dies erreicht wird.</li> </ol> <p><b>Erwartetes Lernprodukt:</b> Zwei austauschbare Speicheradapter, Integrationstests und eine Architekturbegründung.</p> <p><b>Möglicher Startprompt:</b> Frage mich zuerst, welche Verantwortung zu Modell, Abbildung und Speicherung gehört. Schlage erst danach eine Schnittstelle vor.</p> |         |                   |               |

## Sicherheits-Karussell: Risiken erkennen und systematisch begrenzen

| Station | Angriff/Fehler        | Schutzprinzip                            | Lernprodukt              |
|---------|-----------------------|------------------------------------------|--------------------------|
| A       | manipulierte Eingaben | Validierung und sichere Systemgrenzen    | Negativtests             |
| B       | SQL-Injection         | Prepared Statements, minimale Rechte     | abgesicherter DB-Zugriff |
| C       | unzulässiger Zugriff  | Rollen, Berechtigungen, Datenminimierung | Berechtigungsmatrix      |
| D       | Informationsabfluss   | sichere Fehler, Logs und Secrets         | Sicherheitsreview        |

## [OOP-ANH-A02] · SICHERE SOFTWARE · Bedrohungsprobe am Lernagenten

| Station                                                                                                                                                        | Angriff/Fehler                                                                                                                                                                                                     | Schutzprinzip | Lernprodukt |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|-------------|
| 1.                                                                                                                                                             | <b>PROJEKTGRUNDLAGE</b> Verwende K12_Sicherheitslabor_MVC aus 02_Schuelermaterial/Starterprojekte. Beachte die freigegebenen Bearbeitungsstellen im Projekt-README und führe AppMain sowie SicherheitsAbnahme aus. |               |             |
| 2.                                                                                                                                                             | Markiere schützenswerte Daten und mögliche Systemgrenzen im Mockup.                                                                                                                                                |               |             |
| 3.                                                                                                                                                             | Formuliere mindestens vier Missbrauchs- oder Fehlerszenarien.                                                                                                                                                      |               |             |
| 4.                                                                                                                                                             | Ordne präventive, erkennende und wiederherstellende Maßnahmen zu.                                                                                                                                                  |               |             |
| 5.                                                                                                                                                             | Implementiere zwei Schutzmaßnahmen und sichere sie mit Negativtests ab.                                                                                                                                            |               |             |
| 6.                                                                                                                                                             | Prüfe, ob Fehlermeldungen oder Logs vertrauliche Informationen verraten.                                                                                                                                           |               |             |
| <b>Erwartetes Lernprodukt:</b> Ein einfaches Bedrohungsmodell, getestete Schutzmaßnahmen und eine Security-Checkliste.                                         |                                                                                                                                                                                                                    |               |             |
| <b>Möglicher Startprompt:</b> Hilf mir mit Rückfragen, Vermögenswerte, Angriffsflächen und Folgen zu unterscheiden. Nenne noch keine fertigen Schutzmaßnahmen. |                                                                                                                                                                                                                    |               |             |

## Konkrete Unterrichtsmatrix

Die Doppelstunde ist die kleinste planbare Einheit. Im zweistündigen Bildungsgang wird der Kernpfad bearbeitet. Im BKWI2 schließen sich in derselben Woche zwei weitere Doppelstunden mit Implementierung, Tests, Sprachtransfer und Projektvertiefung an.

| Kap. | Kern-Doppelstunde                        | BKWI2-Ausbau                      | Kontext     | Lernagent-Baustein |
|------|------------------------------------------|-----------------------------------|-------------|--------------------|
| 1    | Qualitätsziele und Änderungsprobe        | Architekturvergleich, Code-Smells | BMI-App     | Komponentenkar-te  |
| 2    | Mockup, Anwendungsfall, Objektkandidat-e | OOA/OOD und UML-Entwurf           | Parkautomat | Objektkandidat-e   |

| Kap. | Kern-Doppelstunde           | BKW12-Ausbau                                      | Kontext               | Lernagent-Baustein   |
|------|-----------------------------|---------------------------------------------------|-----------------------|----------------------|
| 3    | Klasse, Objekt, Kapselung   | JUnit, Invarianten, 3-Sprachen-Fenster            | Verein                | Klasse Lernziel      |
| 4    | Bedingungen als Fachregeln  | Entscheidungsbaumen, Refactoring                  | Automaten             | Auswahlregel         |
| 5    | Methoden und Verträge       | Testfälle, Exceptions, Secure Coding              | BMI-App               | gültige Zustände     |
| 6    | Listen und Schleifen        | Suche, Filter, Sortierung, Generics               | Volleyball            | Aufgabenpool         |
| 7    | Assoziationen               | 1:n, n:m, Aggregation/Komposition                 | Verein/Shop           | Lernverlauf          |
| 8    | Abstraktion und Polymorphie | Interfaces, Strategien, Komposition               | Zahlung/Feedback      | FeedbackStrategie    |
| 9    | MVC                         | GUI-Ereignisse und austauschbare Views            | BMI-App               | bedienbarer Prototyp |
| 10   | Daten verwalten             | Datei, JSON, Mapping, Serialisierung              | Volleyball            | Dateispeicher        |
| 11   | Systeme anbinden            | Repository/DAO, JDBC, Service/API                 | BMI/Verein            | Speicheradapter      |
| 12   | Sichere Software            | Validierung, Rechte, Prepared Statements, Secrets | Lernagent             | Sicherheitskonzept   |
| 13   | Refactoring und Qualität    | Testsuite, Dokumentation, Review                  | Fremdcode             | Qualitätsbericht     |
| 14   | Abschluss und Transfer      | eigenständige Erweiterung und Präsentation        | Lernagent/Wahlprojekt | Minimum Viable Agent |

## Auswahl an konkreten Aufgaben

### KERN + TEST · Änderungsauftrag BMI-App

1. Ergänze eine neue Bewertungskategorie, ohne View-Code in der Modellklasse abzulegen.
2. Notiere vor der Änderung alle betroffenen Klassen.

3. Sichere die Regel mit Tests ab und vergleiche Prognose und tatsächliche Änderungen.

**Erwartetes Lernprodukt:** Ein prüfbares Lernprodukt mit Entwurfsbegründung und Änderungsnachweis.

### DATENSTRUKTUREN · Volleyball-Teammanager

1. Modelliere Team und Spieler als verbundene Objekte.
2. Filtere einsatzberechtigte Spieler und ermittle eine Aufstellung.
3. Begründe, welche Logik beim Team und welche bei einer Auswahlstrategie liegt.

**Erwartetes Lernprodukt:** Ein prüfbares Lernprodukt mit Entwurfsbegründung und Änderungsnachweis.

### ASSOZIATIONEN · Verein oder Mini-Shop

1. Vergleiche Mitglied-Team mit Bestellung-Position.
2. Entscheide jeweils zwischen loser Assoziation, Aggregation und Komposition.
3. Prüfe die Lebensdauer der beteiligten Objekte anhand eines Löschszenarios.

**Erwartetes Lernprodukt:** Ein prüfbares Lernprodukt mit Entwurfsbegründung und Änderungsnachweis.

### POLYMORPHIE · Austauschbares Feedback

1. Entwirf einen gemeinsamen Vertrag für kurzes, motivierendes und prüfungsnahes Feedback.
2. Implementiere zwei Strategien in Java.
3. Übertrage nur den Vertrag und eine Strategie nach Python oder PHP.

**Erwartetes Lernprodukt:** Ein prüfbares Lernprodukt mit Entwurfsbegründung und Änderungsnachweis.

**TRANSFER · Lernagent-Abschlussinkrement**

1. Wähle ein bestehendes Modellinkrement und ergänze eine neue Anforderung.
2. Dokumentiere die Änderung vom Mockup über UML bis zum Test.
3. Zeige, wodurch die Erweiterung lokal begrenzt bleibt.

**Erwartetes Lernprodukt:** Ein prüfbares Lernprodukt mit Entwurfsbegründung und Änderungsnachweis.

**Der Lernagent wächst mit**

| OOP-Baustein           | Beitrag zum Lernagenten                              | sichtbares Zwischenprodukt          |
|------------------------|------------------------------------------------------|-------------------------------------|
| Klassen und Kapselung  | Lernziele und Lernstände bleiben gültig              | Modellklasse Lernziel               |
| Kontrollstrukturen     | nächster Lernschritt wird regelbasiert gewählt       | Auswahlmethode                      |
| Listen und Schleifen   | Aufgaben und Ergebnisse werden verarbeitet           | Aufgabenpool                        |
| Assoziationen          | Lernende, Ziele und Bearbeitungen werden verbunden   | Lernverlaufsmodell                  |
| Interfaces/Polymorphie | Aufgaben- und Feedbackstrategien werden austauschbar | Strategie-Schnittstelle             |
| MVC                    | Oberfläche bleibt von Lernlogik getrennt             | bedienbarer Prototyp                |
| Datenverwaltung        | Lernfortschritt wird strukturiert abgebildet         | JSON-Dateispeicher                  |
| Systemanbindung        | Speicher und Services werden austauschbar            | Repository/Service-Adapter          |
| Sichere Software       | Lerndaten und Systemgrenzen werden geschützt         | Sicherheitskonzept und Negativtests |

PILOTPHASE Zeitbedarf, Verständlichkeit, Hilfebedarf, Aufgabenlösbarkeit und Korrekturaufwand werden im ersten Unterrichtsdurchlauf mit dem beigefügten Beobachtungsbogen erfasst. Allgemeingültige Rückmeldungen fließen anschließend in die Masteroutine der gesamten Skriptserie ein.

## Quellen, Lizenz und Prüfnachweis ↩ Inhalt

### Verwendete und weiterführende Quellen

- Ministerium für Kultus, Jugend und Sport Baden-Württemberg: Bildungsplan Berufskolleg Wirtschaftsinformatik, Bildungsplaneinheit 8 „Objektorientierte Systementwicklung“, Fassung vom 13.03.2025.
- Landesbildungsserver Baden-Württemberg: Bildungsplan Wirtschaftsinformatik – Objektorientierte Softwareentwicklung, BPE 5, [https://www.bildungsplaene-bw.de/Wifo\\_OS](https://www.bildungsplaene-bw.de/Wifo_OS) (Abruf: 08.09.2026).
- Landesbildungsserver Baden-Württemberg: OOA/OOD/OOP – Unterrichtsmaterialien, <https://www.schule-bw.de/faecher-und-schularten/mathematisch-naturwissenschaftliche-faecher/informatik/material/softwareentwicklung/ooa-ood-oop> (Abruf: 08.09.2026).
- Christine Janischek: E-Learning Objektorientierte Programmierung, <https://www.emotionalspirit.de/eLearning/OOP/> (inhaltliche Orientierung; Aktualisierungsbedarf berücksichtigt).
- Christine Janischek: java-bmiapp-docker-mv-template, <https://github.com/ChristineJanischek/java-bmiapp-docker-mv-template> (Unterrichtsstruktur, Versionspfade, Aufgaben- und Bewertungsansätze; Abruf: 08.09.2026).

Oracle: Java SE Support Roadmap – LTS-Versionen und Supportzeiträume, Stand August 2026:

<https://www.oracle.com/java/technologies/java-se-support-roadmap.html>

OpenJDK: JDK 25 – offizielle Projekt- und Releaseinformationen:

<https://openjdk.org/projects/jdk/25/>

Spring: Spring Boot – offizielle Projektbeschreibung und

Einsatzfelder: <https://spring.io/projects/spring-boot/>

### Lizenz und Erstellungshinweis

**Lizenz:** Custom License – Christine Janischek NC School-Only 1.0. Nicht-kommerzielle Nutzung ausschließlich im staatlichen Schulwesen. Sichtbare Namensnennung: „Christine Janischek –

<https://emotionalspirit.de>“. Alle anderen Nutzungen nur nach vorheriger schriftlicher Genehmigung.

**KI-Hinweis:** Das Skript wurde mit Unterstützung generativer KI redaktionell erstellt und anschließend nach einer fünfstufigen Lernskript-Prüfroutine überarbeitet. Es kann dennoch Fehler enthalten. Fachliche und technische Hinweise bitte an die Autorin Christine Janischek melden.

## Pilot-Prüfnachweis

| Check       | Status              | Nachweis in v1.2                                                                                                              |
|-------------|---------------------|-------------------------------------------------------------------------------------------------------------------------------|
| Didaktik    | zur Fachprüfung     | Kapitelstandard, Java-Praxisfenster, Sicherungen, Hilfen und Lernagent-Transfer vorhanden                                     |
| Prüfbarkeit | technisch aufgebaut | 65 Aufgaben-IDs, 18 Prüfungsanker, Erwartungshorizonte und Punkte; fachlich und didaktisch durch die Autorin geprüft          |
| Materialien | technisch geprüft   | 10 Starterprojekte und 10 Referenzlösungen kompilieren; 20 Anwendungs- und Abnahmeläufe bestanden; Offline-Pfadtest bestanden |
| Master      | technisch geprüft   | A4/A5, Navigation, semantischer Blätterbuch-Lesemodus und seitenweise Sichtprüfung                                            |

| Check            | Status      | Nachweis in v1.2                                                                      |
|------------------|-------------|---------------------------------------------------------------------------------------|
|                  |             | durchgeführt; durch die Autorin zur Finalisierung freigegeben                         |
| Barrierefreiheit | barrierearm | Struktur vorhanden; manuelle Tastatur-, Screenreader-, Zoom- und PDF/UA-Prüfung offen |