PROFESSOR ERKLÄRBÄR

Serie Informatik · Folge 10 · Advanced

INFORMATIK

LEHRERINFO ZUM VIDEO

Objektorientierte Programmierung

FACH	STUFE	DAUER	VIDEO
Informatik	Gymnasium · Sek II	12 min	youtube.com/@Erklaerbaer

01 Lernziele

Nach dem Video können die Schülerinnen und Schüler …

- Klassen mit Konstruktor, Attributen und Methoden entwerfen und Objekt- von Klassenattributen unterscheiden,
- Kapselung begründen und umsetzen (Invariante, `_balance` und `@property` in Python, `private` mit Getter/Setter in Java),
- Vererbung einsetzen (Ober- und Unterklasse, `super()`) und von Komposition abgrenzen,
- Polymorphie erklären: überschriebene Methoden, gleicher Aufruf – anderes Verhalten, Duck Typing,
- abstrakte Klassen und Interfaces einordnen,
- UML-Klassendiagramme lesen und zeichnen (Sichtbarkeit, Vererbung, Assoziation, Aggregation, Komposition, Multiplizitäten).

02 Voraussetzungen

Klasse, Objekt, Attribut, Methode, `__init__`, `self` (Folge 10 · Grundlagen); Funktionen und Listen (Folge 9); Rekursion und O-Notation sind hilfreich, aber nicht nötig.

03 Ablauf des Videos

ZEIT	KAPITEL	INHALT UND BEISPIEL
0:00	Einstieg	Ritter, Bogenschützin, Magier · Ident · Fahrplan vier Säulen
0:34	Was du brauchst	Klasse, Objekt, `__init__`, `self` · Hinweis auf 10G
0:53	Warum OOP?	Spaghetti-Code gegen Module · Daten + Verhalten · vier Säulen
1:35	Klassen entwerfen	`BankAccount`: owner, balance, deposit, withdraw · Konstruktor · Klassenattribut `rate`

2:09	Kapselung	Invariante balance ≥ 0 · withdraw(150) abgelehnt, withdraw(30) → 70 € · `@property` · Java private · Merksatz
3:17	Vererbung	`Character` → Knight, Archer, Mage · `super().__init__()` · «ist ein» PAUSE-FRAGE Auto erbt von Motor?
4:22	Polymorphie	attack() überschrieben · Rüstung halbiert Schaden · Schleife über party PAUSE-FRAGE Mage ohne attack() · Duck Typing · Merksatz
5:44	Abstrakte Klassen und Interfaces	`ABC`, `@abstractmethod`, TypeError · Java abstract und interface
6:26	UML-Klassendiagramm	drei Fächer, − und + · Vererbungsdreieck · Assoziation, Aggregation, Komposition · Mediathek mit Multiplizitäten · Merksatz
7:47	Projekt-Baustein	Token, Lexer, Parser, Calculator, Operatoren mit apply() (15A) · Klasse ↔ Tabelle (16A)
8:12	Prüfungsfallen	fünf Fallen (siehe 04)
9:11	Zusammenfassung	Kernaussagen · eine Oberklasse, drei Unterklassen
9:46	Quiz	4 Fragen · Endkarte Folge 11

04 Prüfungsfallen

- Vererbung ist immer gut, je tiefer desto besser – nur bei echter «ist ein»-Beziehung; sonst Komposition («hat ein»).
- Privat heisst in Python unsichtbar – `_name` ist Konvention; erzwungen wird Kapselung erst in Java vom Compiler.
- Überladen = Überschreiben – Überschreiben ersetzt eine geerbte Methode; Überladen heisst gleicher Name, andere Parameter (Java, nicht Python).
- Aggregation = Komposition – bei Komposition lebt das Teil nicht ohne das Ganze (Raum – Gebäude), bei Aggregation schon (Spielerin – Team).
- Der Vererbungspfeil zeigt zur Unterklasse – er zeigt zur Oberklasse.

05 Einsatz im Unterricht

Modellieren zuerst: Aus einem kurzen Auftragstext (z. B. Velo-Verleih) Substantive und Verben markieren und daraus ein Klassendiagramm ableiten (Aufgabe 5).

Kapselung testen: In Python versuchen, `acc.balance = -1000` zu setzen (AttributeError dank `@property`), dann über `_balance` «tricksen» – und diskutieren, warum das trotzdem keine gute Idee ist.

Python und Java: Die Java-Fassungen auf der nächsten Seite neben die Python-Beispiele legen und Unterschiede benennen (Sichtbarkeit, Typen, Klammern).

Prüfungsvorbereitung: Aufgaben 4 und 5 entsprechen typischen Abituraufgaben zu UML.

Java-Fassung der Beispiele: `class BankAccount { private double balance; public BankAccount(String owner, double balance) { … } public double getBalance() { return balance; } public void withdraw(double amount) { if (amount > balance) throw new IllegalArgumentException("Nicht genug Geld"); balance -= amount; } }·abstract class Character { protected String name; abstract String attack(); }·class Knight extends Character { String attack() { return name + " schlägt mit dem Schwert zu"; } }·interface Fighter { String attack(); }`

Hinweise: Python kennt kein echtes «privat»; zwei Unterstriche (`__balance`) lösen nur eine Namensänderung aus. Die TypeError-Meldung lautet ab Python 3.12 «Can't instantiate abstract class Character without an implementation for abstract method 'attack'». In der Konsole des Videos hat Mage eine eigene attack()-Methode (Feuerball); die Pause-Frage fragt bewusst nach dem Fall ohne.

06 Quellen

- Python Software Foundation: The Python Tutorial, Kapitel 9 «Classes» · Built-in Functions: property · Modul abc
- Oracle: The Java Tutorials – Object-Oriented Programming Concepts, Interfaces and Inheritance
- OMG: Unified Modeling Language (UML), Version 2.5.1 – Klassendiagramme
- EDK: Rahmenlehrplan für das obligatorische Fach Informatik an Gymnasien (2017)

NAME _____ KLASSE _____ DATUM _____

Objektorientierte Programmierung

1 Die vier Säulen ●○○

Ordne zu: a) Ein Konto lässt sich nur über withdraw() verringern. b) Knight übernimmt name und hp von Character. c) Eine Schleife ruft attack() auf, jede Figur reagiert anders. d) Für den Shop zählen Preis und Lager, nicht die Farbe des Kartons.

2 Kapselung umsetzen ●●○

Ergänze `BankAccount` um `deposit(amount)`, das nur positive Beträge annimmt (sonst ValueError). **Teste**: Start 100 €, deposit(50), withdraw(30), deposit(-5) – welcher Stand, welche Meldung?

3 Vererbung oder Komposition? ●●○

Entscheide und begründe: a) Hund – Tier b) Auto – Rad c) Bibliothek – Buch d) Quadrat – Rechteck e) Handy – Akku

4 Polymorphie ●●○

Schreibe eine Oberklasse `Shape` mit abstrakter Methode `area()` und die Unterklassen `Rectangle` und `Circle`. **Berechne** in einer Schleife die Flächen von Rectangle(3, 4) und Circle(1).

5 Klassendiagramm

«Ein Velo-Verleih hat mehrere Stationen. An jeder Station stehen Velos. Kundinnen und Kunden mieten ein Velo für eine bestimmte Zeit.» **Zeichne** ein UML-Klassendiagramm mit mindestens drei Klassen, Attributen, einer Methode, Beziehungen und Multiplizitäten.

Lösungen

1 Die vier Säulen

a) Kapselung · b) Vererbung · c) Polymorphie · d) Abstraktion

2 Kapselung umsetzen

`def deposit(self, amount): · if amount <= 0: raise ValueError("Betrag muss positiv sein") · self._balance = self._balance + amount` → **100 → 150 → 120;** deposit(-5) löst ValueError aus, der Stand bleibt 120 €.

3 Vererbung oder Komposition?

a) Vererbung (ein Hund ist ein Tier) · b) Komposition (ein Auto hat Räder) · c) Aggregation (Bücher gibt es auch ohne die Bibliothek) · d) Vererbung möglich, aber heikel: Ändert man beim Quadrat nur die Breite, ist es kein Quadrat mehr (Liskov-Problem) · e) Komposition (das Handy hat einen Akku)

4 Polymorphie

```
class Shape(ABC): · @abstractmethod · def area(self): ... · class Rectangle(Shape): · def __init__(self, w, h): self.w, self.h = w, h · def area(self): return self.w * self.h · class Circle(Shape): · def __init__(self, r): self.r = r · def area(self): return 3.14159 * self.r ** 2 · for s in [Rectangle(3, 4), Circle(1)]: print(s.area())
```

→ **12 und 3.14159**

5 Klassendiagramm

Zum Beispiel **Station** (– name) 1 ◆— 0..* **Velo** (– nummer, – frei) · **Kunde** (– name, + mieten(velo)) 1 — 0..* **Miete** (– start, – ende) 0..* — 1 **Velo**. Station–Velo auch als Aggregation vertretbar (ein Velo kann die Station wechseln) – entscheidend ist die Begründung.

Q Quiz im Video

1 B (Abstraktion, Kapselung, Vererbung, Polymorphie) · 2 A (Konstruktor der Oberklasse) · 3 C (gefüllte Raute) · 4 B (privat)