Codeklasse

Der Rennwagen-Stammbaum

Thema: Vererbung

Name: _____

Klasse: _____ Datum: _____

Grundlagen & Merkkasten

WISSEN

In der objektorientierten Programmierung (OOP) müssen oft mehrere Klassen ähnliche Eigenschaften und Fähigkeiten besitzen. Anstatt gemeinsamen Code mehrfach zu schreiben, nutzt man das Konzept der Vererbung: Eine allgemeine Basisklasse (Elternklasse) definiert grundlegende Attribute und Methoden. Spezialisierte Unterklassen (Kindklassen) erben diese automatisch und können um eigene, exklusive Merkmale erweitert werden.

`class Fahrzeug:`
`  def __init__(self, marke):`
`  self.marke = marke`
	Definition der Basisklasse (Elternklasse) mit gemeinsamen Attributen.

`class Rennwagen(Fahrzeug):` `  pass`	Vererbung: Durch Angabe der Elternklasse in Klammern erbt die Unterklasse alle Attribute und Methoden.
`auto = Rennwagen('Apex')` `auto.marke`	Instanzen der Unterklasse besitzen Zugriff auf die Attribute der Basisklasse.

1. Code-Puzzle (Reihenfolge & Einrückung)

AUFGABE 1

Bringe die Code-Bausteine in die korrekte Reihenfolge, um die Basisklasse `Fahrzeug` sowie die abgeleitete Klasse `Rennwagen` zu definieren und anschließend die geerbte Methode `hupen()` aufzurufen. Achte genau auf die korrekte Einrückung!

[ ]	`class Fahrzeug:`
[ ]	`def __init__(self, marke):`
[ ]	`self.marke = marke`
[ ]	`def hupen(self):`
[ ]	`print(self.marke + ' hupt: Hup Hup!')`
[ ]	`class Rennwagen(Fahrzeug):`
[ ]	`pass`
[ ]	`flitzer = Rennwagen('TurboGT')`
[ ]	`flitzer.hupen()`
[ ]	`class Rennwagen:`

Hinweis: Trage in die eckigen Klammern die korrekte Reihenfolge (1, 2, 3...) ein. Streiche überflüssige oder fehlerhafte Zeilen (Distraktoren) durch!

2. Fehlersuche & Code-Reparatur

AUFGABE 2

Im Fuhrpark-System existiert ein Fehler in der Klassenvererbung.
Repariere den Klassenkopf von `Traktor`, sodass er von `Fahrzeug` erbt, und behebe den Aufruf der geerbten Methode `info()`.

Fehlerhafter Ausgangs-Code:

```
1  class Fahrzeug:
2      def __init__(self, marke):
3          self.marke = marke
4
5      def info(self):
6          print("Fahrzeug-Marke:", self.marke)
7
8  # BUG: Traktor erbt aktuell von keiner Klasse!
9  class Traktor:
10     def pfluegen(self):
11         print(self.marke, "pfluegt das Feld.")
12
13 trecker = Traktor("AgroTitan")
14 # BUG: Die Methode heisst info(), nicht zeige_info()
15 trecker.zeige_info()
16 trecker.pfluegen()
```

Deine Korrektur / Notizen:

3. Programmier-Challenge

AUFGABE 3

Modelliere ein Fahrzeug-Stammbaum-System nach folgenden Vorgaben:

1. Erstelle eine Basisklasse `Fahrzeug` mit:
- Konstruktor `__init__(self, marke)`, der das Attribut `self.marke` setzt.
- Methode `hupe(self)`, die `"<marke> macht: Tut tut!"` auf der Konsole ausgibt.
2. Erstelle eine abgeleitete Klasse `Rennwagen`, die von `Fahrzeug` erbt und eine eigene Methode besitzt:
- `turbo(self)`: Gibt `"<marke> rast mit 300 km/h!"` aus.
3. Erstelle eine Instanz `mein_renner` der Klasse `Rennwagen` mit der Marke `'TurboGT'`.
4. Rufe an `mein_renner` zuerst die geerbte Methode `hupe()` und danach `turbo()` auf!

Dein Python-Code:

Reflexion: Das DRY-Prinzip und Klassenhierarchien

AUFGABE 4

Erkläre anhand des Konzepts der Vererbung das „DRY-Prinzip“ (Don't Repeat Yourself). Welche konkreten Nachteile entstehen in einem großen Softwaresystem, wenn spezialisierte Klassen wie Rennwagen, LKW und Traktor alle gemeinsamen Attribute (wie Marke, Geschwindigkeit, Hupe) unabhängig voneinander definieren, statt von einer gemeinsamen Basisklasse zu erben?