Codeklasse

Das OOP-Battle-Arena-Game

Thema: Synthese-Projekt: Objektorientierte Programmierung

Name: _____

Klasse: _____ Datum: _____

Grundlagen & Merkkasten

WISSEN

In komplexen Softwareprojekten interagieren viele eigenständige Einheiten miteinander.
Durch das Zusammenspiel von Klassen, Vererbung und Polymorphie können gemeinsame Basisstrukturen definiert und spezialisierte Unterklassen mit individuellem Verhalten ausgestattet werden.
In diesem Synthese-Projekt verbinden wir alle Kernkonzepte der objektorientierten Programmierung zu einer interaktiven Kampf-Simulation.

`class Krieger(Charakter):`	Leitet eine spezialisierte Unterklasse von einer übergeordneten Basisklasse ab.
`super().__init__(name, hp, atk)`	Ruft den Konstruktor der Elternklasse auf, um Basiseigenschaften zu initialisieren.
`def angreifen(self, ziel):`	Methoden können andere Objekte als Argumente entgegennehmen und deren Zustand verändern.
`self.hp = max(0, self.hp - schaden)`	Kapselt Zustandsänderungen sicher innerhalb des jeweiligen Objekts.

1. Code-Puzzle (Reihenfolge & Einrückung)

AUFGABE 1

Die Basisklasse `Charakter` ist im Hintergrund bereits vordefiniert:
Sie speichert `name`, `hp` und `atk` und besitzt eine Methode `nimm_schaden(schaden)`.

Setze die spezialisierte Klasse `Krieger` zusammen, die im Konstruktor einen zusätzlichen `wut`-Wert initialisiert, die Methode `angreifen(ziel)` überschreibt (Schaden = `atk + wut`) und anschließend ein Duell ausführt.

```
[ ]  class Krieger(Charakter):
[ ]  def __init__(self, name, hp, atk, wut):
[ ]  super().__init__(name, hp, atk)
[ ]  self.wut = wut
[ ]  def angreifen(self, ziel):
[ ]  gesamtschaden = self.atk + self.wut
[ ]  print(f'{self.name} schlaegt mit Wut zu!')
[ ]  ziel.nimm_schaden(gesamtschaden)
[ ]  hero = Krieger('Valerius', 100, 20, 10)
[ ]  gegner = Charakter('Dummy', 100, 0)
[ ]  hero.angreifen(gegner)
[ ]  print(f'Gegner HP: {gegner.hp}')
[ ]  super().__init__(self, name, hp, atk)
[ ]  ziel.hp -= self.atk
```

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

In dieser Arena-Logik haben sich zwei Fehler eingeschlichen:
1. Der Konstruktor der Unterklasse `Magier` ruft die Elternklasse `Charakter` nicht korrekt auf.
2. Die Methode `feuerball` zieht zwar Mana ab, ruft aber nicht `ziel.nimm_schaden()` auf.

Korrigiere den Code, sodass der Magier erfolgreich zaubert und dem Ziel Schaden zufügt!

Fehlerhafter Ausgangs-Code:

```
class Charakter:
    def __init__(self, name, hp):
        self.name = name
        self.hp = hp

    def nimm_schaden(self, schaden):
        self.hp -= schaden

class Magier(Charakter):
    def __init__(self, name, hp, mana):
        # BUG: super() Aufruf fehlerhaft / unvollstaendig
        self.name = name
        self.mana = mana

    def feuerball(self, ziel, schaden):
        self.mana -= 15
        # BUG: Ziel erleidet noch keinen Schaden!
        print(f"{self.name} wirkt Feuerball!")

magier = Magier("Ignis", 100, 50)
gegner = Charakter("Boss", 100)
magier.feuerball(gegner, 20)

print("Gegner HP:", gegner.hp)
print("Magier Mana:", magier.mana)
```

Deine Korrektur / Notizen:

3. Programmier-Challenge

AUFGABE 3

Erstelle die Kern-Architektur für das Arena-Kampfspiel:

1. **Basisklasse `Charakter`**:
- Konstruktor `__init__(self, name, hp, atk)` speichert die Attribute.
- Methode `nimm_schaden(self, punkte)`: Zieht `punkte` von `self.hp` ab (Minimum 0: nutze z. B. `max(0, ...)`).
- Methode `ist_am_leben(self)`: Gibt `True` zurück, wenn `hp > 0`, sonst `False`.
2. **Unterklasse `Schurke(Charakter)`**:
- Erbt von `Charakter`.
- Konstruktor `__init__(self, name, hp, atk, krit_bonus)`: Ruft `super().__init__(name, hp, atk)` auf und speichert `self.krit_bonus = krit_bonus`.
- Methode `hinterhalt(self, ziel)`: Berechnet Schaden als `self.atk + self.krit_bonus`, fügt diesen `ziel` über `ziel.nimm_schaden(...)` zu und gibt `"Hinterhalt ausgefuehrt!"` auf der Konsole aus.
3. **Instanziierung & Duell:**
- Erstelle `held` als `Schurke` mit Name `'Valerius'`, HP `100`, ATK `20` und Krit-Bonus `10`.
- Erstelle `boss` als Basis-`Charakter` mit Name `'Gegner'`, HP `100` und ATK `10`.
- Lasse `held` die Methode `hinterhalt(boss)` ausführen.
- Gib `f"Boss HP: {boss.hp}"` und `f"Boss Lebt: {boss.ist_am_leben()}"` aus.

Dein Python-Code:

Transfer & Architektur-Reflexion

AUFGABE 4

Erkläre das Prinzip der Polymorphie anhand einer Schleife:
Angenommen, eine Liste `team` enthält sowohl Instanzen der Klasse `Krieger` als auch der Klasse `Magier`.
Warum kann man mit `for mitglied in team: mitglied.angreifen(ziel)` arbeiten, ohne vorher mit `if/else` den genauen Typ der Spielfigur abzufragen? Welcher Software-Architektur-Vorteil entsteht dadurch?