Kapselung umsetzen bei Python

Kapselung

- Anlass ist die bereits behandelte Aufgabe zum direkten Setzen des Attributwertes für die Farbe

```
stuhl.f="brown"
```

- Sie zeigt, dass der Wert verändert werden kann, ohne dass sich die Darstellung des Objekts Stuhl verändert.

Kapselung

- Dasselbe Problem stellt sich grundsätzlich aber auch für die xPosition, yPosition, die Breite und Tiefe und für die Sichtbarkeit.
- So wird beispielsweise die Zuweisung

```
stuhl.b=100
```

- zwar ausgeführt und der Wert gespeichert, aber das Bild nicht neu dargestellt.

Kapselung

- Ziel ist, dass Zugriffe auf Attribute niemals direkt ausgeführt werden (können), sondern nur
  - lesend über die Get-Methoden (sondierende Methoden)
  - schreibend über die Set-Methoden (verändernde Methoden)
- Direkte Zugriffe auf Attribute sind dann nur innerhalb der Klassendefinition möglich. (Sie können durch den Zugriff auf Get- und Set-Methoden auch dort vermieden werden.)

Kapselung

Umsetzung bei Java:

- Die Attribute bekommen das Sichtbarkeitsprädikat private

Umsetzung bei Python:

- Die Attributnamen werden am Beginn mit zwei Unterstreichungsstrichen versehen.

```
class Stuhl():
    def __init__(self,
                 xPos=20,
                 yPos=20,
                 breite=40,
                 tiefe=40,
                 winkel=0,
                 farbe="black",
                 sichtbar=False):
        self.__x=xPos
        self.__y=yPos
        self.__b=breite
        self.__t=tiefe
        self.__w=winkel
        self.__f=farbe
        self.__s=sichtbar
        if sichtbar: self.Zeige()
```

Kapselung

Umsetzung bei Python:

- Die Attributnamen werden am Beginn mit zwei Unterstreichungsstrichen versehen.
- Aufruf von

  ```
  stuhl.__b=100
  ```

  führt zu einem Fehler.

Problem bei Python:

- Die Kapselung lässt sich umgehen.