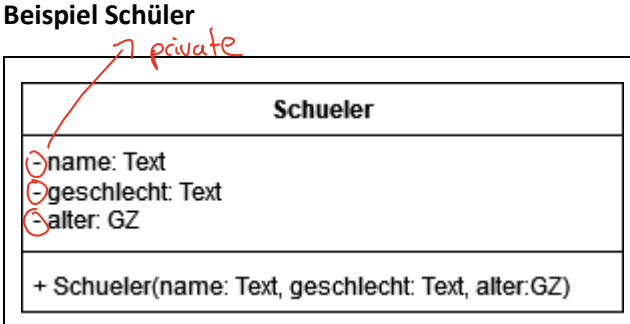
Datenkapselung (Geheimnisprinzip)

Wir haben inzwischen mit Klassen und Objekten gearbeitet und dabei Methoden erstellt. Nun beschäftigen wir uns mit einem Thema, das professionelle Softwareentwickler betrifft – **Datenkapselung**.

Die Idee der Kapselung besteht darin, die Implementierung der Attribute vor dem Nutzer des Objektes zu verbergen. Sie sind dann privat. Der Nutzer kann die Attributwerte eines Objektes durch **set**- und **get**-Methoden festlegen und auslesen.

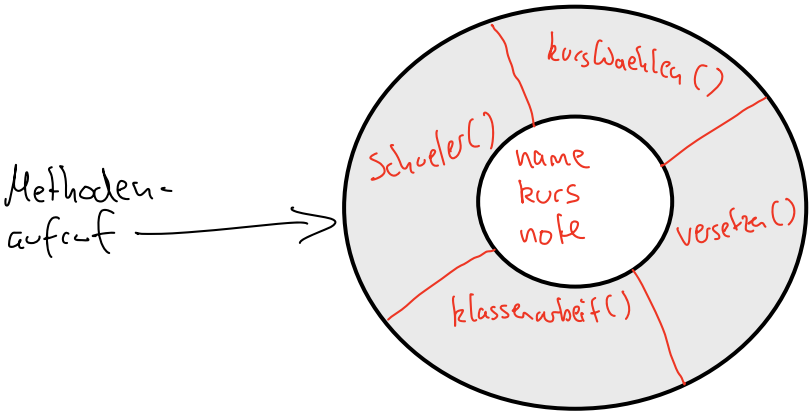
Beispiel Schüler

	Die Attribute sind statt wie bisher mit einem Pluszeichen (+) nun mit einem **Minuszeichen (-)** gekennzeichnet. Das bedeutet, dass der Nutzer im Hauptprogramm **keinen** Zugriff darauf hat. Er muss eine entsprechende **get**-Methode zum Auslesen und eine **set**-Methode zum Festlegen eines Attributwertes anwenden.

Geheimnisprinzip

Lesen oder Schreiben der Attributwerte eines Objektes ist nun nur noch mit Methoden möglich.

➔ Zustandsänderungen des Objektes erfolgen mittels Methodenaufrufen.



Kapselung: Attribute sind durch Operationen vor der „Außenwelt" verborgen.

Schnittstelle: Menge aller öffentlichen Operationen einer Klasse.

Sichtbarkeitsebene

Jede Operation / Methode kann auf die Attribute des Objekts zugreifen.

Für Methoden und Attribute muss die Sichtbarkeitsebene angegeben werden:

- **private**(**-**) : nur innerhalb der Klasse direkt verwendbar
- **public** (**+**): außerhalb der Klasse sichtbar
- **protected** (**#**): nur innerhalb Unterklassen sichtbar

In der Regel werden Attribute als **private** angegeben, damit keine unkontrollierte Manipulation der Daten erfolgen kann (**Datenkapselung**). Dies hat außerdem den Vorteil, dass Details der internen Implementierung verborgen bleiben (**Geheimnisprinzip**).

Private Attribute deklarieren

```
class Schueler():
    def __init__(self, name, geschlecht, klasse, alter):
        self.__name = name
        self.__geschlecht = geschlecht
        self.__alter = alter

    def __str__(self):
        return f"Name: {self.__name}\n
                 Geschlecht: {self.__geschlecht}\n
                 Alter: {self.__alter}"
```

Set- und get-Methoden definieren

Um den Zugriff auf die Attribute dennoch sicherzustellen, erfolgt dieser über **get**- und **set**-Methode, die zunächst definiert werden müssen.

```
class Schueler():
    def __init__(self, name, geschlecht, klasse, alter):
        self.__name = name
        self.__geschlecht = geschlecht
        self.__alter = alter

    @property
    def get_geschlecht(self):
        return self.__geschlecht

    def set_geschlecht(self, geschlecht):
        self.__geschlecht = geschlecht


    def __str__(self):
        ...
```

Set- und get-Methoden anwenden

Im Hauptprogramm werden set- und get-Methoden wie folgt aufgerufen:

```
schueler1 = Schueler("Sarah", "weiblich", 17)
print(schueler1.get_geschlecht())
schueler1.set_geschlecht("männlich")
print(schueler1)
```