

Hinweis: Diese Lernnotiz enthält einen sehr sinnvollen Vorschlag um den Lehrstoff der 6. Woche der Veranstaltung zu erlernen. Er ist gegliedert in die generellen Ziele und die Arbeitsschritte. Es ist notwendig, dass Sie die in dieser Lernnotiz genannten Videos bis zum Ende der offiziellen Vorlesungszeit (Mi 4.11, 16:00, am Mo 9.11, 16:00) durchgearbeitet haben. Zu den Vorlesungszeiten besteht die Möglichkeit in Zoom Fragen zu stellen und weitergehende Themen zu Aufgaben oder auch dem Studium zu diskutieren.

<https://hs-osnabrueck.zoom.us/my/kleuker>

Denken Sie daran, dass ich auch über E-Mail erreichbar bin und Fragen beantworte.

Ziele VL 11

- Fähigkeit zur Unterscheidung von dem Selben (Identität) und dem Gleichen (Gleichheit)
- Fähigkeit zur systematischen Programmierung einer equals()-Methode
- Sinn und Zweck von toString() verstehen und selber umsetzen können

Arbeitsschritte VL 11

- *Laden Sie sich die folgenden Videos zuerst herunter, wenn Sie die HS-Plattform nutzen und schauen Sie sich diese an. Es ist sinnvoll die Folien danach nochmals durchzugehen.*

Entwicklung eines vorläufigen equals() in BlueJ

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Equals1.mp4> (26:09), auch <https://youtu.be/z5Ck63wikXk>

Folien 268 – 281: Systematischer Aufbau von equals()

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Equals2.mp4> (25:28), auch <https://youtu.be/smHEOHgJAqQ>

Als Randnotiz sei angemerkt, außerhalb des ersten Semesters schreiben Sie die letzten 4 Zeilen mit der gleichen Bedeutung als:

```
return "Andrea".equals(this.name);    oder  
return this.name.equals("Andrea");
```

Folien 282 – 289: Systematische Umsetzung der Objektausgabe mit toString()

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1ToString.mp4> (23:44), auch <https://youtu.be/1AB-jTssJUk>

Beispiele zur Nutzung von ArrayList in BlueJ

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1ArrayList1.mp4> (14:33), auch <https://youtu.be/UZpkl5wE4xU>

- Bearbeiten Sie das Quiz unter http://kleuker.iui.hs-osnabrueck.de/quiz/prog11_11239.html und merken Sie sich die oben angegebenen Lösungsbuchstaben für das Praktikum.
- Lesen Sie das zur Vorlesung gehörende Fragen-Und-Antworten-Dokument, das meist kurz nach der Vorlesung auf der Veranstaltungsseite in der Nähe dieser Lernnotiz steht.
- Prüfen Sie, ob Sie die angegebenen Lernziele erreicht haben.

Ziele VL 12

- Akzeptanz und Fähigkeit zur Nutzung von benötigten Wrapper-Klassen für elementare Typen
- Fähigkeit ArrayList für Sammlungen von Objekten gleichen Typs nutzen zu können
- Syntax und Semantik einfacher while-Schleifen mit ihrem typischen Aufbau kennen und in einfachen Beispielen nutzen können

Arbeitsschritte VL 12

- *Laden Sie sich die folgenden Videos zuerst herunter, wenn Sie die HS-Plattform nutzen und schauen Sie sich diese an. Es ist sinnvoll die Folien danach nochmals durchzugehen.*

Folien 290 – 302: Einführung von ArrayList

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1ArrayList2.mp4> (27:23), auch <https://youtu.be/Q9myhqwHCmo>

Einführung von while in BlueJ

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Schleife1.mp4> (21:27), auch <https://youtu.be/vOwvhl8vTU>

Folien 303 – 326: Nutzung einer einfachen while-Schleife

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Schleife2.mp4> (31:57), auch <https://youtu.be/jrmYPnnBBBE>

- Bearbeiten Sie das Quiz unter http://kleuker.iui.hs-osnabrueck.de/quiz/prog12_11240.html und merken Sie sich die oben angegebenen Lösungsbuchstaben für das Praktikum.
- Lesen Sie das zur Vorlesung gehörende Fragen-Und-Antworten-Dokument, das meist kurz nach der Vorlesung auf der Veranstaltungsseite in der Nähe dieser Lernnotiz steht.
- Prüfen Sie, ob Sie die angegebenen Lernziele erreicht haben.
- Bearbeiten Sie Aufgabenblatt 6. Denken Sie daran, dass ich für Fragen meist kurzfristig erreichbar bin.

Übertragung nach Python (nicht relevant für das 1. Semester)

Folien 269 – 279:

Unabhängig von einer Programmiersprache, gibt es immer die verwandten Konzepte der Identität und der Gleichheit. Identität bedeutet, dass ein identisches Objekt (das selbe) gemeint ist. Dies wird in Java mit == und in Python mit is überprüft. Bei der Gleichheit entscheidet die entwickelnde Person, wann zwei Objekt inhaltlich gleich sind. Dies meint oft eine Übereinstimmung in den Werten aller Objektvariablen, kann sich aber auch auf andere Werte beziehen. Da Gleichheit ein sehr wichtiges Konzept ist, wird dies in vielen Programmiersprachen besonders unterstützt. In Java ist es die Methode equals(), von der später deutlich wird, dass damit eine spezielle Methode der Klasse Object überschrieben wird. Nun kann so eine Methode auch in Python einfach equals() genannt werden. Vereinfachend wird hier aber gleich der Standard genutzt, es wird eine Methode __eq__() geschrieben. Diese sogenannten Dunder (double underscore) -Methoden haben eine besondere Bedeutung in Python. Die Grundidee dabei ist, dass __eq__() automatisch dann

aufgerufen wird, wenn in Python der Vergleich mit == stattfindet. Dies zeigt das folgende kleine Beispiel.

```
class X:
    def __init__(self, val:int) -> None:
        self.val = val

    def __eq__(self, other:"X") -> bool:
        print("in __eq__")
        if other is None:
            return False
        if self.val is None:
            return other.val is None
        return self.val == other.val

x1:X = X(3)
x2:X = X(3)
x3:X = x1
x4:X = None      # Warning: incompatible types
x5:X = X(None)   # Warning: incompatible types

print(f" 1: {x1 is x2}")
print(f" 2: {x1 == x2}")
print(f" 3: {x1 is x3}")
print(f" 4: {x1 == x3}")
print(f" 5: {x1 is x4}")
print(f" 6: {x1 == x4}")
print(f" 7: {x1 is x5}")
print(f" 8: {x1 == x5}")
print(f" 9: {x4 is x1}")
print(f"10: {x4 == x1}")
print(f"11: {x5 is x1}")
print(f"12: {x5 == x1}")
print(f"13: {x4 == None}")
print(f"14: {x4 is None}")
print(x4)
print(type(x4))
```

Die Ausgaben lauten:

```
1: False
in __eq__
2: True
3: True
in __eq__
4: True
5: False
in __eq__
6: False
7: False
in __eq__
8: False
9: False
in __eq__
10: False
```

```
11: False
in __eq__
12: False
13: True
14: True
None
```

```
<class 'NoneType'>
```

Das Beispiel zeigt auch deutlich, dass `None` in Python und `null` in Java sich nicht identisch verhalten. In der Ausgabezeile 9 steht in Python kein Fehler, während in Java jedweder Methodenaufruf auf `null` immer zu einem Fehler, einer `NullPointerException` führt. Die Ausgabezeilen 9 und 10 machen deutlich, dass es sich bei `None` um ein „normales“ Objekt der Klasse `None`, genauer des Typs `NoneType`, handelt. `None` ist genauer das einzige und immer existierende Objekt dieser Klasse. Damit existiert die Methode `__eq__()` für den `None`-Type. Wird nebenbei keine Implementierung für `__eq__()` angegeben, wird in Python, wie in Java, eine Prüfung auf Identität durchgeführt.

Festzuhalten bleibt, dass zwar die konkrete Umsetzung von Identität und Gleichheit in Programmiersprachen unterschiedlich ist, die generellen Konzepte aber identisch sind. Dadurch, dass `None` nicht `null` entspricht, ist die typische Umsetzung von `__eq__()` im Detail anders als in Java, da innerhalb der Methoden keine Methodenaufrufe auf `None` stattfinden. Deshalb funktionieren beide folgende Implementierungen, wovon die erste natürlich einfacher ist.

```
def equalsPython(self, other: "Studierend") -> bool:
    if other is None:
        return False
    if (self.__vorname == other.get_vorname()
        and self.__nachname == other.get_nachname()
        and self.__studiengang == other.get_studiengang()
        and self.__geburtsjahr == other.get_geburtsjahr()
        and self.__matrikelnummer == other.get_matrikelnummer()):
        return True
    return False

def equals(self, other: "Studierend") -> bool:
    if other is None:
        return False

    if self.__nachname is None:
        if other.get_nachname() is not None:
            return False
    else:
        if self.__nachname != other.get_nachname():
            return False

    if self.__studiengang is None:
        if other.get_studiengang() is not None:
            return False
    else:
        if self.__studiengang != other.get_studiengang():
            return False

    if self.__vorname is None:
        if other.get_vorname() is not None:
```

```
        return False
    else:
        if self.__vorname != other.get_vorname():
            return False

    return (other.get_geburtsjahr() == self.__geburtsjahr
            and other.get_matrikelnummer() == self.__matrikelnummer)
```

Die Ausgaben zeigen, dass beide Methoden funktionieren.

```
s1.equalsPython(s2): True
s1.equals(s2): True
s1.equalsPython(s3): False
s1.equals(s2): False
s1.equalsPython(s1): False
s1.equals(s1): False
s3.equalsPython(s4): False
s3.equals(s4): False
```

Würde das erste if mit der None-Prüfung gelöscht, würden auch in Python Fehler auftreten, da dann wieder Methoden aufgerufen werden würden, die es für None nicht gibt.

Bei den Parametern in den letzten beiden Beispielen bei `__eq__()` fällt auf, dass dort Typen in Anführungsstrichen ("X", "Studierend") stehen. Dies ist in Python eine Möglichkeit auf Typen zuzugreifen, die noch nicht vollständig definiert sind. Die Methoden befinden sich ja jeweils in den Klassen für die die Gleichheit definiert wird.

Folie 280f:

Python hat keine Markierung für mehrzeilige Kommentare. In Python muss vor jeder auskommentierten Zeile ein # stehen. Praktisch jede Entwicklungsumgebung bietet dazu die Möglichkeit mehrere Zeilen auszuwählen und dann über eine Auswahl auszukommentieren oder bei jeder Zeile das führende Hash-Symbol wieder zu löschen.

```
class Leitend:
    """
    Klasse zur Verwaltung des aktuell leitenden Person.
    """

    def __init__(self, name: str) -> None:
        """
        Konstruktor fuer neues Leitend-Objekt
        Parameter name: neuer Name der leitenden Person
        """
        self.__name: str = name          # Name Leitend
        self.__level: int = 23           # erreichte Stufe

    def isAndrea(self) -> bool:
        """
        Methode zur Pruefung, ob Leitend Andrea heisst
        Ergebnis: heisst Leitend Andrea
        """
        if "Andrea" == self.__name:
            return True
        return False
```

```
l1: Leitend = Leitend("Andrea")
l2: Leitend = Leitend("Bert")
print("l1.isessesAndrea():", l1.isessesAndrea())
print("l2.isessesAndrea():", l2.isessesAndrea())
```

Die zugehörige Ausgabe lautet:

```
l1.isessesAndrea(): True
l2.isessesAndrea(): False
```

Eine besondere Bedeutung haben die Kommentare die sich innerhalb von drei Anführungsstrichen am Start und am Ende befinden. Diese Kommentare, die wie in Java mit JavaDoc zur detaillierten Dokumentation genutzt werden, sind z. B. in Entwicklungsumgebungen leicht abrufbar bzw. bei Mauskontakt sichtbar.

```
37     l1: Leitend = Leitend("Andrea")
38     l2: Leitend = Leitend("Bert")
39
40     print("l1.isessesAndrea():", l1.isessesAndrea())
41     print("l2.isessesAndrea():", l2.isessesAndrea())
```

(method) def isessesAndrea() -> bool
Methode zur Pruefung, ob Leitend Andrea heisst Ergebnis: heisst Leitend Andrea

Als Randnotiz sei angemerkt, außerhalb des ersten Semesters schreiben Sie die letzten 3 Zeilen als:

```
return "Andrea" == self.__name
```

Folie 284:

Der +-Operator kann in Python nur Strings verketteten, eine automatische Umwandlung anderer Objekte in Strings findet nicht statt. Um diese Umwandlung zu nutzen, wird die Funktion `str()` genutzt.

```
ergebnis1: str = "Hallo" + "dele"
print(ergebnis1)                                # Hallodele
ergebnis2: str = "Zeich ma Zahl " + str(161)
print(ergebnis2)                                # Zeich ma Zahl 161
```

In jeder objektorientierten Programmiersprache gibt es eine einfache Möglichkeit sich ein Objekt genauer anzusehen. Der Methode `toString()` in Java entspricht die Methode `__str__()` in Python. In Python gibt es zusätzlich noch die Methode `__repr__()` (representation) die ebenfalls einen String liefert. Die Methode wird typischerweise vom Debugger genutzt. Ist die Methode nicht ausimplementiert, wird `__str__` aufgerufen, was im folgenden Beispiel explizit ausprogrammiert wurde.

```
def __str__(self) -> str:
    return f"{self.__vorname} {self.__nachname} (" \
           +f"{self.__matrikelnummer}):(self.__studiengang}"

def __repr__(self) -> str:
    return self.__str__()

s5:Studierend = Studierend("Eva", "Mustermann", 1995, "AS", 232323)
s6:Studierend = Studierend(None, None, None, None, None)
print(s5)
print(s6)
string:str = str(s5)
print(string)
```

Die Ausgabe lautet:

```
Eva Mustermann (232323):AS
```

None None (None):None
Eva Mustermann (232323):AS

Folie 288:

```
def sonderzeichen(self) -> None:
    io: EinUndAusgabe = EinUndAusgabe()
    io.ausgeben("0123456789012345678\n")
    io.ausgeben("01\t123\t45\n")
    io.ausgeben("\"ein\\Pfeil\"\n")
    io.ausgeben("\u00c4 \u00e4 \u00d6 \u00f6 \u00dc \u00fc "
                + " \u00df \u00E9 \u00A6 \u00A2 \u20AC \u2020 ")
```

Die zugehörige Ausgabe lautet:

```
0123456789012345678
01      123      45
"ein\ /Pfeil"
Ä ä Ö ö Ü ü  ß é | ¢ € †
```

Folie 292:

Wie bereits erwähnt, sind alle Typen in Python Klassen, deshalb ist das etwas nervige Boxing und Unboxing nicht notwendig. Die Klasse List in Python auch eine sinnvolle Ausgabe.

Folie 295:

```
from typing import List
from EinUndAusgabe import EinUndAusgabe
```

```
class ListenSpielerei:
```

```
    def spielen(self) -> None:
        zahlen: List[int] = []
        io: EinUndAusgabe = EinUndAusgabe()
        io.ausgeben("1: " + str(zahlen) + "\n")
        zahlen.append(42)          # Java: add(42)
        zahlen.append(41)
        io.ausgeben("2: " + str(zahlen) + "\n")
        zahlen.insert(1, 23)      # Java: add(1, 23);
        io.ausgeben("3: " + str(zahlen) + "\n")
        io.ausgeben("4: " + str(zahlen[2]) + "\n")    # Java: get(2)
        zahlen[0] = 67           # Java: set(0, 67);
        io.ausgeben("5: " + str(zahlen) + "\n")
        zahlen.pop(0)            # Java: remove(0);
        # alternativ: del zahlen[0]
        io.ausgeben("6: " + str(zahlen) + "\n")
        io.ausgeben("7: " + str(len(zahlen)) + "\n")  # Java: size()

        tmp: str = "Die Liste beinhaltet {zahlen}"
        tmp2: str = f"Die Liste beinhaltet {zahlen}"
        print(tmp)
        print(tmp2)
```

```
spielerei: ListenSpielerei = ListenSpielerei()
```

```
spielerei.spielen()
```

Die zugehörige Ausgabe lautet:

```
1: []
2: [42, 41]
3: [42, 23, 41]
4: 41
5: [67, 23, 41]
6: [23, 41]
7: 2
Die Liste beinhaltet {zahlen}
Die Liste beinhaltet [23, 41]
```

In Python werden Listen und die später in der Vorlesung betrachteten Arrays aus Java nicht unterschieden. Leider gibt es keine einheitlichen Namen der Methoden, allerdings sind die Java-Namen verbreiteter.

In Python gibt es einige Funktionen, die ohne Klammern auskommen. Ein solches Beispiel ist die Funktion `del` (delete) mit dem ein Objekt gelöscht werden kann. Eine ähnliche Funktion wurde mit `f` (formatted string) bereits genutzt, mit der in Strings einfach Variablenwerte eingebaut werden können. Dabei muss auf `f` in dieser Form unmittelbar ein Anführungszeichen folgen, wie die letzten Programmzeilen zeigen.

Es gibt auf Listen in Python auch eine Methode `remove()`, mit der allerdings das erste Objekt der Liste gelöscht wird, das gleich dem bei `remove()` übergebenen Objekt ist.

Folie 297ff:

Von der Logik her, gibt es keine Unterschiede zum Java-Programm.

```
from typing import List
from EinUndAusgabe import EinUndAusgabe
```

```
class DatumsListe:
```

```
    def analyse(self) -> None:
        io: EinUndAusgabe = EinUndAusgabe()
        daten: List[Datum] = []
        d1: Datum = Datum(1, 1, 2013)
        d2: Datum = Datum(1, 1, 2014)
        daten.append(d1)
        daten.append(d2)
        daten.append(d1)                                # dieselbe Referenz
        tmp: Datum = daten[0]
        io.ausgeben(str(tmp) + "\n")
        tmp2: Datum = daten[1]
        io.ausgeben(str(tmp2) + "\n")
        io.ausgeben(str(daten[2]) + "\n")
        d1.setTag(9)
        tmp.setMonat(9)
        daten[0].setJahr(99)
        io.ausgeben(str(daten[2]) + "\n")

    def gehtNicht(self) -> None:
        """public void gehtNicht()"""
        io: EinUndAusgabe = EinUndAusgabe()
```

```
d1: Datum = Datum(1, 1, 2013)
d1.startDerZeit(d1)
io.ausgeben(str(d1) + "\n")
```

d1 ist unverändert!

```
d1: DatumsListe = DatumsListe()
d1.analyse()
print("\n=== DatumsListe.gehtNicht() ===")
d1.gehtNicht()
```

Die zugehörige Ausgabe lautet.

```
1.1.2013
1.1.2014
1.1.2013
```

```
=== DatumsListe.gehtNicht() ===
9.9.99
vorher: 1.1.2013
nachher: 0.0.0
1.1.2013
```

Folie 301f:

```
from typing import List
from EinUndAusgabe import EinUndAusgabe
```

```
class IntegerListe:
```

```
    def problem(self) -> None:
        io: EinUndAusgabe = EinUndAusgabe()
        arr: List[int] = []
        arr.append(2)
        arr.append(3)
        arr.append(4)
        big: int = 42
        arr.append(big)
        io.ausgeben(str(arr) + "\n")
        del arr[2]
        io.ausgeben(str(arr) + "\n")

        # arr.remove(43) ergibt
        # ValueError: list.remove(x): x not in list
        if 43 in arr:
            arr.remove(43)
        io.ausgeben(str(arr) + "\n")           # [3, 42]
        if big in arr:
            arr.remove(big)
        io.ausgeben(str(arr) + "\n")           # [3]
        del arr[3]
```

Die zugehörige Ausgabe lautet:

```
[2, 3, 4, 42]
[2, 3, 42]
[3, 42]
```

```
[3, 42]
[3]
File "f:\Lehre\Material_Prog\Python\folie297ff.py", line 169, in problem
    del arr[3]
        ~~~^^^
IndexError: list assignment index out of range
```

Wieder muss das etwas andere Konzept beim Löschen beachtet werden. Da die Methode das erste gleiche Objekt löscht, wird verlangt, dass dieser Wert auch in der Liste vorhanden ist. Ist das nicht der Fall, gibt es einen Fehler. Mit der Bedingung kann geprüft werden, ob ein Objekt in einer Liste enthalten ist. Im gezeigten Programm wird die Bedingung nach False ausgewertet und die Anweisung nicht ausgeführt.

Folie 305ff

Die Logik hinter den beiden Programmiersprachen ist hier identisch.

```
from typing import List
from EinUndAusgabe import EinUndAusgabe
from Interaktionsbrett import Interaktionsbrett

class ErsteSchleifen:

    def zahlenBisAusgeben(self, grenze: int) -> None:
        io: EinUndAusgabe = EinUndAusgabe()
        zaehler: int = 0
        while zaehler <= grenze:
            io.ausgeben(str(zaehler) + "\n")
            zaehler = zaehler + 1

    def maleQuadrat1(self, anzahl: int) -> None:
        ib: Interaktionsbrett = Interaktionsbrett()
        zaehler: int = 0
        while zaehler < anzahl:
            ib.neuesRechteck(10 + zaehler * 15, 10, 12, 12)
            zaehler = zaehler + 1
        # Fenster anzeigen (in Java erscheint es automatisch)
        ib.zeige()

    def maleQuadrat2(self, anzahl: int) -> None:
        ib: Interaktionsbrett = Interaktionsbrett()
        zaehler: int = 0
        xPosition: int = 10
        while zaehler < anzahl:
            ib.neuesRechteck(xPosition, 10, 12, 12)
            zaehler = zaehler + 1
            xPosition = xPosition + 15
        ib.zeige()

    def zahlenBisZurueckgeben(self, grenze: int) -> List[int]:
        ergebnis: List[int] = []
        zaehler: int = 0
        while zaehler <= grenze:
            ergebnis.append(zaehler)
```

```
        zaehler = zaehler + 1
    return ergebnis
```

```
es: ErsteSchleifen = ErsteSchleifen()
print("=== zahlenBisAusgeben(5) ===")
es.zahlenBisAusgeben(5)

print("\n=== zahlenBisZurueckgeben(7) ===")
liste: List[int] = es.zahlenBisZurueckgeben(7)
print(liste)

print("\n=== maleQuadrade1(8) (öffnet Interaktionsbrett) ===")
# Hinweis: Fenster bleibt offen, bis es geschlossen wird.
es.maleQuadrade1(8)
```

Die Ausgabe sieht wie folgt aus.

```
=== zahlenBisAusgeben(5) ===
0
1
2
3
4
5

=== zahlenBisZurueckgeben(7) ===
[0, 1, 2, 3, 4, 5, 6, 7]

=== maleQuadrade1(8) (öffnet Interaktionsbrett) ===
```



Die etwas unschöne Notwendigkeit mit dem speziellen Befehl, um das Interaktionsbrett sichtbar zu halten, ist jetzt in der schöneren Methode `ib.zeige()` versteckt, die weiterhin am Ende der Programmausführung stehen muss.

Folie 315ff:

```
from typing import List, Optional, Any
from EinUndAusgabe import EinUndAusgabe

class Studierendenverwaltung:

    def __init__(self) -> None:
        self.__studierende: List[Studierend] = []

    def hinzufuegen(self, neu: Studierend) -> None:
        self.__studierende.append(neu)
```

```
def steuern(self) -> None:
    ein: int = -1
    io: EinUndAusgabe = EinUndAusgabe()
    while ein != 0:
        io.ausgeben("Nächste Aktion:\n"
                    + " (0) Programm beenden\n"
                    + " (1) Neue studierende Person einfügen\n")
        ein = io.lese_integer()
        if ein == 1:
            self.studierendEingeben()
            io.ausgeben("\n" + str(self.__studierende) + "\n")

def studierendEingeben(self) -> Studierend:
    ergebnis: Studierend = Studierend()
    ergebnis.setVorname(self.nichtLeererText("Vorname"))
    ergebnis.setNachname(self.nichtLeererText("Nachname"))
    ergebnis.setStudiengang(self.nichtLeererText("Studiengang"))
    ergebnis.setGeburtsjahr(self.zahlZwischen("Geburtsjahr"
        , 1900, 2000))
    ergebnis.setMatrikelnummer(self.zahlZwischen("Matrikelnummer"
        , 99999, 1000000))
    self.__studierende.append(ergebnis)
    return ergebnis

def nichtLeererText(self, aufforderung: str) -> str:
    ergebnis: str = ""
    io: EinUndAusgabe = EinUndAusgabe()
    while ergebnis == "":
        io.ausgeben(aufforderung + ": ")
        ergebnis = io.lese_string()
        ergebnis = ergebnis.strip()
    return ergebnis

def zahlZwischen(
    self,
    aufforderung: str,
    untereGrenze: int,
    obereGrenze: int,
) -> int:
    ergebnis: int = untereGrenze - 1
    io: EinUndAusgabe = EinUndAusgabe()
    while ergebnis < untereGrenze or ergebnis > obereGrenze:
        io.ausgeben(aufforderung + ": ")
        ergebnis = io.lese_integer()
    return ergebnis

def getStudierende(self) -> List[Studierend]:
    return self.__studierende

def setStudierende(self, studierende: List[Studierend]) -> None:
    self.__studierende = studierende
```

```
class Studierendenspielerei:
```

```
    def eingeben(self) -> None:
        sv: Studierendenverwaltung = Studierendenverwaltung()
        ein: Studierend = sv.studierendEingeben()
        io: EinUndAusgabe = EinUndAusgabe()
        io.ausgeben("" + str(ein))
```

```
verwaltung: Studierendenverwaltung = Studierendenverwaltung()
verwaltung.steuern()
```

Die Ausgabe sieht wie folgt aus, Eingaben sind umrandet.

Nächste Aktion:

(0) Programm beenden

(1) Neue studierende Person einfügen

1

Vorname: Igor

Nachname:

Nachname: Smid

Studiengang: AS

Geburtsjahr: 2007

Geburtsjahr: 2000

Matrikelnummer: 42

Matrikelnummer: 123456

[Igor Smid (123456):AS]

Nächste Aktion:

(0) Programm beenden

(1) Neuen studierende Person einfügen

0