

Hinweis: Diese Lernnotiz enthält einen sehr sinnvollen Vorschlag um den Lehrstoff der 4. 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 14.10, 16:00 und Mo 19.10, 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>

Ziele VL 7

- Sinn und Möglichkeiten eines Debuggers verstehen, ihn als zentrales Entwicklungswerkzeug erkennen und einsetzen können
- die Abarbeitung von geschachtelten Methoden verstehen und selbstständig auf Papier ausführen können
- Verstehen, was unveränderbare Objekte sind und wie sie sich verhalten
- Bedeutung von null-Werten und das Problem von NullPointerExceptions verstehen
- Method Chaining lesen und nutzen können

Arbeitsschritte VL 7

- *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 181 – 185: Einführung in das Debugging mit BlueJ

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Debugger1.mp4> (21:32), auch <https://youtu.be/VroCao-awtg>

Folien 186 – 197: Verfolgung von Methodenausrufen im Debugger

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Debugger2.mp4> (24:51), auch <https://youtu.be/FcJSb4ywQxw>

Folien 198 – 201: Klasse String und unveränderbare Objekte (immutable objects)

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Debugger3.mp4> (11:39), auch <https://youtu.be/nitEy9swVqw>

Method Chaining

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Debugger4.mp4> (23:15), auch https://youtu.be/zy2HgA_2vHY

- Bearbeiten Sie das Quiz unter http://kleuker.iui.hs-osnabrueck.de/quiz/prog07_01236.html und merken Sie sich die oben angegebenen Lösungsbuchstaben für das Praktikum.
- Lesen Sie das Frage-und-Antworten-Dokument zur Vorlesung von der Veranstaltungs-Webseite, das spätestens am nächsten Arbeitstag online ist.
- Prüfen Sie, ob Sie die angegebenen Lernziele erreicht haben.

Ziele VL 8

- das Konzept der Nutzung von Objektreferenzen als Parameter und bei der Zuweisung verstehen und bei der selbstständigen Entwicklung nutzen
- im kleinsten Detail verstehen, wie Objektspeicherdiagramme zu lesen und zu erstellen sind

Arbeitsschritte VL 8

- 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.

Einfache Objektspeicherdiagramme erstellen

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Objektweitergabe1.mp4>
(8:50), auch <https://youtu.be/Ovmss62DLVU>

Komplexe Objektspeicherdiagramme erstellen

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Objektweitergabe2.mp4>
(13:50), auch <https://youtu.be/FwCUt3vVrEs>

Folien 202 – 219: Objektweitergabe im Detail

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Objektweitergabe3.mp4>
(32:56), auch <https://youtu.be/osQV3CPRQGs>

Folien 220 – 226: Weiteres Beispiel mit Referenzen

<http://kleuker.iui.hs-osnabrueck.de/Videos/Prog1/Prog1Objektweitergabe4.mp4>
(27:09), auch <https://youtu.be/oC1pipelBc0>

Einschub: E-Mail-Fragen eines Studis: Nach nochmaligem schauen der Videos sind mir zwei Unstimmigkeiten bei den Objektspeicherdiagrammen aufgefallen.

1. In diesem Video (<https://youtu.be/oC1pipelBc0> [1], Zeitstempel 5:13) wird der Wert des Strings direkt reingeschrieben während bei diesem Video (<https://youtu.be/osQV3CPRQGs> [2], Zeitstempel 15:31) der Wert des Strings noch einmal extra auf ein String-Objekt referenziert wird.

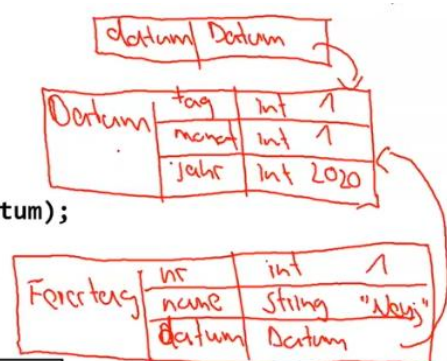
2. In diesem Video (<https://youtu.be/oC1pipelBc0> [3], Zeitstempel 7:52) werden direkt die Werte hingeschrieben, ohne den Typen int, wobei in anderen Beispielen stets auch der Typ mitgeschrieben wird.

Nun ist meine Frage wieso es unterschiedlich aufgeschrieben werden kann, bzw. ob es sich einfach nur um einen Fehler handelt. Und wenn es ein Fehler ist, welche Schreibweise ist die Richtige?

Antwort: Zunächst sehr präzise beobachtet.

```
class Analyse {
    EinUndAusgabe io = new EinUndAusgabe();

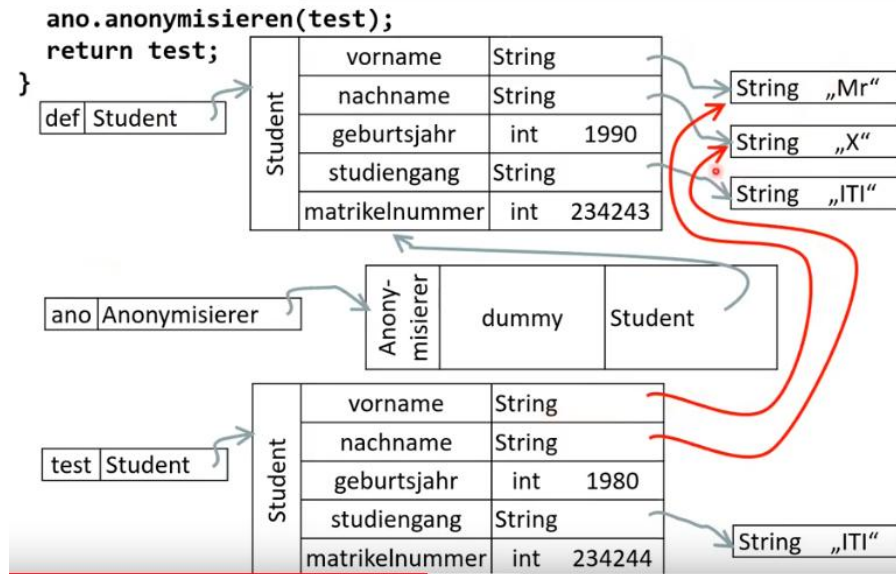
    void ersteSchritte(){
        Datum datum = new Datum(1, 1, 2020);
        this.io.ausgeben(datum.alsText() + "\n");
        Feiertag f1 = new Feiertag(1, "Neujahr", datum);
        this.io.ausgeben(f1.alsText() + "\n");
        datum.setJahr(2019);
        this.io.ausgeben(f1.alsText() + "\n");
    }
}
```



```
1.1.2020
1: 1.1.2020 Neujahr
1: 1.1.2019 Neujahr
```

Die vorherige Abbildung zeigt deutlich, dass bei dem Objekt vom Typ Feiertag der String nicht als eigenständiges Objekt eingetragen ist, es keine Objektreferenz gibt. Dies ist falsch,

sauber wäre ein eigenständig eingezeichnetes Objekt, wie es die nachfolgende Abbildung zu [2] zeigt. Nur so ist es möglich, dass mehrere Variablen den gleichen String referenzieren. Es gilt, sollte als Typ eine Klasse auftreten, muss das zugehörige Objekt immer als Referenz (Pfeil) eingezeichnet werden. Keine Pfeile stehen bei den elementaren Typen, die keine Klassen sind, also int, long, short, byte, double, float, boolean, char.

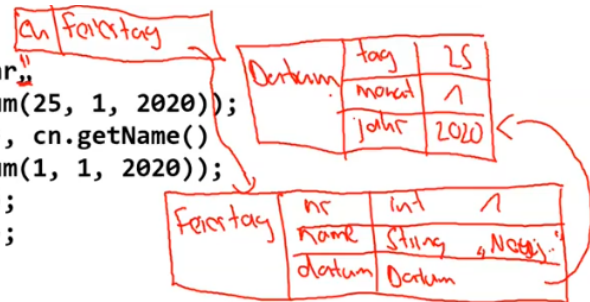


Als „Ausrede“ sei angemerkt, dass der Fehler für Strings minimal ist, da die Klasse die Eigenschaft hat, dass ihre Objekte nicht verändert werden können. Jede Methode also auch solche wie `replace()`, die das vom Namen her suggerierte Verhalten haben, liefern immer ein neues Objekt zurück. Sollten Sie in den Lösungen zur Praktikumsaufgabe die vereinfachte Darstellung bei Strings wählen, wäre das ok.

```

void verzahnteObjekte(){
    Feiertag cn = new Feiertag(1, "Neujahr",
        , new Datum(25, 1, 2020));
    Feiertag de = new Feiertag(cn.getNr(), cn.getName()
        , new Datum(1, 1, 2020));
    this.io.ausgeben(de.alsText() + "\n");
    this.io.ausgeben(cn.alsText() + "\n");
    de.setNr(0);
    cn.setName("Chinesisches Neujahr");
    this.io.ausgeben(de.alsText() + "\n");
    this.io.ausgeben(cn.alsText() + "\n");
}

```



```

1: 1.1.2020 Neujahr
1: 25.1.2020 Neujahr
0: 1.1.2020 Neujahr
1: 25.1.2020 Chinesisches Neujahr

```

Die vorherige Abbildung zeigt die Situation in [3] etwas später im Video. Die Vermutung ist korrekt, dass beim Datum schlichtweg die Typen vergessen wurden. Die Darstellung im Skript ist nach meinem Erkenntnisstand konsistent und korrekt.

- Bearbeiten Sie das Quiz unter http://kleuker.iui.hs-osnabrueck.de/quiz/prog08_01237.html und merken Sie sich die oben angegebenen Lösungsbuchstaben für das Praktikum.

- Lesen Sie das Frage-und-Antworten-Dokument zur Vorlesung von der Veranstaltungs-Webseite, das spätestens am nächsten Arbeitstag online ist.
- Bearbeiten Sie Aufgabenblatt 4. Denken Sie daran, dass ich für Fragen meist kurzfristig erreichbar bin.
- Prüfen Sie, ob Sie die angegebenen Lernziele erreicht haben.

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

Folie 133f, 140ff

in Klasse Studierend

```
def vertauscheNamen(self) -> None:    # geht so nicht
    self.vorname = self.nachname
    self.nachname = self.vorname
```

```
def vertauscheNamenOK(self) -> None:
    tmp: str = self.vorname
    self.vorname = self.nachname
    self.nachname = tmp
```

class Studierendspielerei:

```
def namenstausch(self) -> Studierend:
    xaver: Studierend = Studierend("Huber", "Xaver", 2008
                                   , "ITI", 424223)

    xaver.vertauscheNamen()
    return xaver
```

```
def namenstauschOK(self) -> Studierend:
    xaver: Studierend = Studierend("Huber", "Xaver", 2008
                                   , "ITI", 424223)

    xaver.vertauscheNamenOK()
    return xaver
```

```
ssp: Studierendspielerei = Studierendspielerei()
```

```
stud: Studierend = ssp.namenstausch()
```

```
stud.ausgeben()
```

```
stud = ssp.namenstauschOK()
```

```
stud.ausgeben()
```

Die Ergebnisse sehen wie folgt aus.

```
Vorname: Xaver
Nachname: Xaver
Studiengang: ITI
Vorname: Xaver
Nachname: Huber
Studiengang: ITI
```

Als Randinformation, es gibt eine Mehrfachzuweisung in Python `a, b = b, a`

Folie 196f

```
from EinUndAusgabe import EinUndAusgabe
```

```
class Schachtel:

    def __init__(self) -> None:
        self.io: EinUndAusgabe = EinUndAusgabe()

    def meth1(self) -> None:
        self.io.ausgeben("start meth1\n")
        self.meth2()
        self.io.ausgeben("ende meth1\n")

    def meth2(self) -> None:
        self.io.ausgeben("start meth2\n")
        self.meth3()
        self.io.ausgeben("ende meth2\n")

    def meth3(self) -> None:
        self.io.ausgeben("start meth3\n")
        self.meth4()
        self.io.ausgeben("ende meth3\n")

    def meth4(self) -> None:
        self.io.ausgeben("meth4\n")

schachtel: Schachtel = Schachtel()
schachtel.meth1()

# start meth1
# start meth2
# start meth3
# meth4
# ende meth3
# ende meth2
# ende meth1

Folie 200
from __future__ import annotations

class Datum:

    def __init__(self, tag: int, monat: int, jahr: int) -> None:
        self.tag: int = tag
        self.monat: int = monat
        self.jahr: int = jahr

    def get_tag(self) -> int:
        return self.tag

    def get_monat(self) -> int:
        return self.monat

    def get_jahr(self) -> int:
```

```
        return self.jahr

    def set_tag(self, tag: int) -> Datum:
        return Datum(tag, self.monat, self.jahr)

    def set_monat(self, monat: int) -> Datum:
        return Datum(self.tag, monat, self.jahr)

    def set_jahr(self, jahr: int) -> Datum:
        return Datum(self.tag, self.monat, jahr)

    def __str__(self) -> str:
        return f"{self.tag:02d}.{self.monat:02d}.{self.jahr}"

d1: Datum = Datum(21, 9, 2026)
print("d1:", d1)                                # 21.09.2026

d2: Datum = d1.set_tag(28)
print("d2:", d2)                                # 28.09.2026
print("d1:", d1)                                # 21.09.2026

d3: Datum = d2.set_monat(12).set_jahr(2025)
print("d3:", d3)                                # 28.12.2025
print("d2:", d2)                                # 28.09.2026
```

Die Methode `__str__()` ist ein kleiner Vorgriff, wie effizient eine Ausgabemöglichkeit für Objekte in Python aussieht (analog zu `toString()` in Java).

Anders als Java und die meisten anderen Sprachen kann Python nicht auf Artefakte (Klassen, Methoden) zugreifen, die erst später in einer Datei deklariert werden oder deren Deklaration noch nicht beendet ist. Hier wird in der Klasse `Datum` die Klasse `Datum` genutzt. Um dieses Problem zu lösen, ist der Ansatz die erste Zeile mit „`from __future__ import annotations`“.

Folie 201

Anders als in Java existiert eine Variable ohne Wert nicht. Verwandt zu `null` in Java ist `None` in Python, muss aber explizit gesetzt werden.

```
class Punkt:
    def __init__(self) -> None:
        self.x: int = None
        self.y: int = None

    def zeige(self) -> None:
        print(f"x: {self.x}")
        print(f"y: {self.y}")

p: Punkt = Punkt()
p.zeige()

class PunktBah:
    def __init__(self) -> None:
        self.x: int
```

```
        self.y: int

    def zeige(self) -> None:
        print(f"x: {self.x}")
        print(f"y: {self.y}")
        print(f"z: {self.z}")

    def set_nullpunkt(self) -> None:
        self.x = 0
        self.y = 0
        self.z = 0

p: PunktBah = PunktBah()
# p.set_nullpunkt()
p.zeige()
```

Die Klasse Punkt zeigt, dass im Konstruktor die Objektvariablen angelegt werden, da sie einen Wert erhalten. Die Ausgabe dazu ist zunächst:

```
x: None
y: None
```

Die Klasse PunktBah ist syntaktisch korrekt. Mit Java-Kenntnissen wird ein Fehler in der Zeile „print(f'z: {self.z}')“ erwartet, da z nicht deklariert ist. Die letzte Zeile führt aber zu folgendem Fehler.

```
28 p: PunktBah = PunktBah()
29 # p.set_nullpunkt()
30 p.zeige()
31
```

Kommandozeile ×

```
x: None
y: None
Traceback (most recent call last):
  File "F:\Lehre\Material Prog\Python\workspace\Interaktionsbrett\folie201.py", line 30, in <module>
    p.zeige()
  File "F:\Lehre\Material Prog\Python\workspace\Interaktionsbrett\folie201.py", line 19, in zeige
    print(f"x: {self.x}")
AttributeError: 'PunktBah' object has no attribute 'x'
```

Die Variable self.x ist nicht bekannt, da sie keinen Wert hat. Wird das Kommentarzeichen vor p.set_nullpunkt() entfernt, läuft das Programm und liefert folgende Ausgabe.

```
x: 0
y: 0
z: 0
```

Es ist also nur eine sehr sehr sinnvolle Konvention in Python Objektvariablen im Konstruktor zu deklarieren, notwendig ist es nicht. Das führt auch zu einem klassischen Python-Problem, dass Tippfehler bei Zuweisungen, wie self.falsh = 42, zu keinem Fehler führen.

Ergänzend zeigt das folgende Beispiel, dass so natürlich auch die unangenehmen NullPointerExceptions in anderer Form in Python auftreten können.

```
1 class Manager:
2     def __init__(self, nase:str) -> None:
3         self.nase = nase
4
5     def get_nase(self) -> str:
6         return self.nase
7
8 m1: Manager = Manager("ich")
9 print(m1.get_nase())
10 m2: Manager = Manager(None)
11 print(m2.get_nase())
12
13 class Firma:
14     def __init__(self, chef:Manager) -> None:
15         self.chef = chef
16
17     def get_chef(self) -> Manager:
18         return self.chef
19
20 f1: Firma = Firma(m2)
21 print(f1.get_chef().get_nase())
22 f2: Firma = Firma(None)
23 print(f2.get_chef().get_nase())
```

Kommandozeile ×

```
ich
None
None
Traceback (most recent call last):
  File "F:\Lehre\Material Prog\Python\workspace\Interaktionsbrett\NoneCheck.py", line 23, in <module>
    print(f2.get_chef().get_nase())
AttributeError: 'NoneType' object has no attribute 'get_nase'
```

AttributeError: 'NoneType' object has no attribute 'get_nase'

[NoneCheck.py, line 23](#)

Your program tries to call method get_nase of an object of type 'NoneType', but this type doesn't have such method.

- ⊕ Did you misspell the name?
- ⊕ Did you expect another type?

Warnings

May help you find the cause of the error.

[NoneCheck.py](#)

- ⊞ [Line 10](#): Argument 1 to "Manager" has incompatible type "None"; expected "str" [arg-type]
- ⊞ [Line 22](#): Argument 1 to "Firma" has incompatible type "None"; expected "Manager" [arg-type]

[Was it helpful or confusing?](#)

[General advice on dealing with errors.](#)

Interessant ist aber, dass in der Warning deutlich gemacht wird, dass None eine eigenständige Klasse ist. Dies kann man in Python durch eine sogenannte Type-Union verhindern, dabei trennt ein „|“ die alternativen Typen. Dies macht dann entwickelnden Personen deutlich, dass None-Werte auftreten können, was dann bei Implementierungen zu beachten ist. Als Vorgriff wird hier der bessere Code für das Beispiel angegeben, wobei konkrete Lösungen projektindividuell zu bestimmen sind.

```
class Manager:
    def __init__(self, nase:str | None) -> None:
        if nase is None:
            self.nase = "<unbekannt>"
        else:
            self.nase = nase

    def get_nase(self) -> str:
        return self.nase
```

```
m1: Manager = Manager("ich")
print(m1.get_nase())
m2: Manager = Manager(None)
```

```
print(m2.get_nase())

class Firma:
    def __init__(self, chef:Manager | None) -> None:
        if chef is None:
            self.chef = Manager("<unbesetzt>")
        else:
            self.chef = chef

    def get_chef(self) -> Manager:
        return self.chef

f1: Firma = Firma(Manager(None))
print(f1.get_chef().get_nase())
f2: Firma = Firma(None)
print(f2.get_chef().get_nase())
```

Die Ausgabe ist dann:

```
ich
<unbekannt>
<unbekannt>
<unbesetzt>
```

Als Ausblick sei erwähnt, dass Python und Java dafür eine generische Klasse `Optional` anbieten, die z. B. bei einem Typen `Optional<Manager>` entweder ein `Manager`-Objekt oder `None` referenzieren. Beim Entwickeln heißt dies ebenfalls, dass immer auf `null` bzw. `None` zu prüfen ist.

Folie 204

```
# in Klasse Studierendspielerei
def standardITISTudi(self) -> Studierend:
    ergebnis: Studierend = Studierend()
    ergebnis.set_studiengang("ITI")
    return ergebnis

ssp: Studierendspielerei = Studierendspielerei()
stud: Studierend = ssp.standardITISTudi()
stud.ausgeben()
```

Die jetzt etwas erweiterte Ausgabe ist:

```
Vorname: Eva
Nachname: Mustermann
Geburtsjahr: 1990
Studiengang: ITI
Matrikelnummer: 232323
```

Folie 205f

```
from Studierend import Studierend
class Anonymisierer:
    def __init__(self, dummy: Studierend) -> None:
        self.dummy = dummy

    def anonymisieren(self, zuAendern: Studierend) -> None:
```

```
zuAendern.set_vorname(self.dummy.get_vorname())
zuAendern.set_nachname(self.dummy.get_nachname())

# in Klasse Studierendspielerei
def anonymAusprobieren(self) -> Studierend:
    dum: Studierend = Studierend("Mr", "X", 1990
                                   , "ITI", 234243)
    ano:Anonymisierer = Anonymisierer(dum)
    test: Studierend = Studierend("Ms", "Y", 1980
                                   , "MID", 234244)

    ano.anonymisieren(test)
    return test

ssp: Studierendspielerei = Studierendspielerei()
stud: Studierend = ssp.anonymAusprobieren()
stud.ausgeben()

Die Ausgabe ist:
    Vorname: Mr
    Nachname: X
    Geburtsjahr: 1980
    Studiengang: MID
    Matrikelnummer: 234244

Folie 212f
from __future__ import annotations
from Studierend import Studierend

class Anonymisierer:
    def __init__(self, dummy: Studierend) -> None:
        self.dummy = dummy

    def anonymisieren(self, zuAendern: Studierend) -> None:
        zuAendern.set_vorname(self.dummy.get_vorname())
        zuAendern.set_nachname(self.dummy.get_nachname())

    def lese_dummy_aus(self, spielerei: Studierendspielerei2) -> None:
        self.dummy = spielerei.get_studi()

class Studierendspielerei2:
    def __init__(self) -> None:
        self.studi = None

    def get_studi(self) -> Studierend:
        return self.studi

    def sich_uebergeben(self) -> Studierend:
        self.studi = Studierend("Mr", "X", 1990
                                   , "ITI", 234243)
        ano:Anonymisierer = Anonymisierer(None)
        ano.lese_dummy_aus(self)
        test: Studierend = Studierend("Ms", "Y", 1980
```

```
        , "MID", 234244)

    ano.anonymisieren(test)
    return test

ssp: Studierendspielerei2 = Studierendspielerei2()
stud: Studierend = ssp.sich_uebergeben()
stud.ausgeben()
```

Die Ausgabe ist:

```
Vorname: Mr
Nachname: X
Geburtsjahr: 1980
Studiengang: MID
Matrikelnummer: 234244
```

Generell verhalten sich Python und Java hier gleich.

Ähnliche Probleme wie bei der Nutzung erst später deklarierter Artefakte, hat Python mit zyklischen Abhängigkeiten, also Klasse A nutzt Klasse B und Klasse B nutzt Klasse A. Aus diesem Grund befindet sich als einfachste Lösung der gesamte Code in einer Datei. Da zyklische Abhängigkeiten nur in Ausnahmefällen sinnvoll sind und Python auch da Lösungen hat, ist das kein generelles Problem.

Folien 220ff

```
from __future__ import annotations
from EinUndAusgabe import EinUndAusgabe
```

class Datum:

```
    def __init__(self, tag: int = 0, monat: int = 0, jahr: int = 0) -> None:
        self.tag: int = tag
        self.monat: int = monat
        self.jahr: int = jahr

    def als_text(self) -> str:
        return f"{self.tag}.{self.monat}.{self.jahr}"

    def get_tag(self) -> int:
        return self.tag

    def set_tag(self, tag: int) -> None:
        self.tag = tag

    def get_monat(self) -> int:
        return self.monat

    def set_monat(self, monat: int) -> None:
        self.monat = monat

    def get_jahr(self) -> int:
        return self.jahr

    def set_jahr(self, jahr: int) -> None:
        self.jahr = jahr
```

```
def __str__(self) -> str:
    return self.als_text()

class Feiertag:

    def __init__(
        self,
        nr: int,
        name: str,
        datum: Datum
    ) -> None:
        self.nr: int = nr
        self.name: str = name
        self.datum: Datum = datum

    def als_text(self) -> str:
        return f"{self.nr}: {self.datum.als_text()} {self.name}"

    def get_nr(self) -> int:
        return self.nr

    def set_nr(self, nr: int) -> None:
        self.nr = nr

    def get_name(self) -> str:
        return self.name

    def set_name(self, name: str) -> None:
        self.name = name

    def get_datum(self) -> Datum:
        return self.datum

    def set_datum(self, datum: Datum) -> None:
        self.datum = datum

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

class Analyse:

    def __init__(self) -> None:
        self.io: EinUndAusgabe = EinUndAusgabe()

    def erste_schritte(self) -> None:
        datum: Datum = Datum(1, 1, 2020)
        self.io.ausgeben(datum.als_text() + "\n")
        f1: Feiertag = Feiertag(1, "Neujahr", datum)
        self.io.ausgeben(f1.als_text() + "\n")
        datum.set_jahr(2019)
        self.io.ausgeben(f1.als_text() + "\n")
```

```
def verzahnte_objekte(self) -> None:
    cn: Feiertag = Feiertag(1, "Neujahr", Datum(25, 1, 2020))
    de: Feiertag = Feiertag(cn.get_nr(), cn.get_name()
                           , Datum(1, 1, 2020))
    self.io.ausgeben(de.als_text() + "\n")
    self.io.ausgeben(cn.als_text() + "\n")
    de.set_nr(0)
    cn.set_name("Chinesisches Neujahr")
    self.io.ausgeben(de.als_text() + "\n")
    self.io.ausgeben(cn.als_text() + "\n")

def verzahnte_objekte2(self) -> None:
    cn: Feiertag = Feiertag(1, "Neujahr", Datum(25, 1, 2020))
    de: Feiertag = Feiertag(1, cn.get_name(), cn.get_datum())
    self.io.ausgeben(de.als_text() + "\n")
    self.io.ausgeben(cn.als_text() + "\n")
    cn.set_name("Chinesisches Neujahr")
    de.get_datum().set_tag(1)
    self.io.ausgeben(de.als_text() + "\n")
    self.io.ausgeben(cn.als_text() + "\n")

def verzahnte_objekte3(self) -> None:
    cn: Feiertag = Feiertag(1, "Neujahr", Datum(25, 1, 2020))
    de: Feiertag = cn
    self.io.ausgeben(de.als_text() + "\n")
    self.io.ausgeben(cn.als_text() + "\n")
    cn.set_name("Chinesisches Neujahr")
    de.get_datum().set_tag(1)
    self.io.ausgeben(de.als_text() + "\n")
    self.io.ausgeben(cn.als_text() + "\n")

analyse: Analyse = Analyse()
print("=== erste_schritte() ===")
analyse.erste_schritte()
print("\n=== verzahnte_objekte() ===")
analyse.verzahnte_objekte()

print("\n=== verzahnte_objekte2() ===")
analyse.verzahnte_objekte2()

print("\n=== verzahnte_objekte3() ===")
analyse.verzahnte_objekte3()
```

Fachlich gesehen verhält sich das Python-Programm wie das Java-Programm. Der Konstruktor von Datum zeigt, wie Default-Werte in der Parameterliste gesetzt werden können. Dadurch können diese Parameter beim Aufruf weggelassen werden. Die Ausgabe ist:

```
=== erste_schritte() ===
1.1.2020
1: 1.1.2020 Neujahr
1: 1.1.2019 Neujahr
```

```
=== verzahnte_objekte() ===  
1: 1.1.2020 Neujahr  
1: 25.1.2020 Neujahr  
0: 1.1.2020 Neujahr  
1: 25.1.2020 Chinesisches Neujahr
```

```
=== verzahnte_objekte2() ===  
1: 25.1.2020 Neujahr  
1: 25.1.2020 Neujahr  
1: 1.1.2020 Neujahr  
1: 1.1.2020 Chinesisches Neujahr
```

```
=== verzahnte_objekte3() ===  
1: 25.1.2020 Neujahr  
1: 25.1.2020 Neujahr  
1: 1.1.2020 Chinesisches Neujahr  
1: 1.1.2020 Chinesisches Neujahr
```