Zaehlerprojekt entwickeln

Zaehler

UML
und
test-driven-development

Zaehlerprojekt entwickeln

Vorbereitung

- Klärung der Aufgabe

Entwickle schrittweise eine OO Realisierung

  - *eines einfachen Zählers, der beginnend bei 0 jeweils auf Anforderung um 1 weiter zählt,*
  - *eines Zählers, der sich auch zurücksetzen lässt und*
  - *eines zyklischen Zählers, der nach Erreichen der Zyklenlänge jeweils auf 0 zurückgesetzt wird.*

- Identifizierung der Klassen

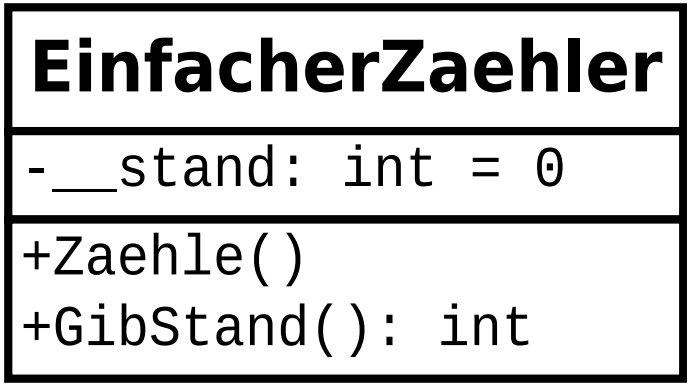
Zaehlerprojekt entwickeln

Schritt 1

- Klassenkarten erstellen und gegebenenfalls zu einem UML-Diagramm vervollständigen
- Attribute definieren
- Methoden definieren
- Beziehungen definieren
- Export des Python-Codes (?)

Zaehlerprojekt entwickeln

- Klassenkarte zu EinfacherZaehler



Zaehlerprojekt entwickeln

Schritt 2 *(bei generiertem Code)*

```
class EinfacherZaehler():

    def __init__(self):
        self.__stand=0  # int

    def Zaehle(self):
        '''''''
        pass

    def ZeigeStand(self):
        '''''''
        pass
```

Zaehlerprojekt entwickeln

Schritt 2 *(bei generiertem Code)*

- der generierte Code wird ggf umsortiert
- ggf Layout beim Klassenkommentar und bei der Einrückungstiefe (Standard 4)
- Kommentare bei den Methoden einfügen
- unnötige pass bei den Konstruktoren entfernen

Zaehlerprojekt entwickeln

Schritt 2 *(ohne generierten Code)*

- Definition des Klassenkopfes
- Konstruktor **__init__** definieren,
  dessen Parameter und die Initialisierung von Attributen klären
- Methoden definieren,
  dazu Parameter und Rückgaben klären
- Kommentare bei den Methoden einfügen
- zunächst, wenn nötig, **pass** als Methodenrumpf einfügen

Zaehlerprojekt entwickeln

Schritt 2 *ein Ergebnis*

```
class EinfacherZaehler :
    '''ein Zaehler, der um 1 weiterzaehlen und
    seinen Stand ausgeben kann'''

    def __init__(self) :
        self.__stand = 0 # int

    def Zaehle (self) :
        '''zaehlt jeweils um 1 weiter'''
        pass

    def GibStand (self) :
        '''zeigt den Stand des Zaehlers an'''
        pass
```

Zaehlerprojekt entwickeln

Schritt 3

- Einbau der Testklassen aus der Hilfe zu unittest und bearbeiten

```
import unittest
class TestEinfacherZaehler(unittest.TestCase):
    def setUp(self):
        self.zaehler = EinfacherZaehler()

    def test_Initialisierung(self):
        self.assertTrue(self.zaehler.GibStand()==0)

if __name__ == '__main__':
    unittest.main()
```

Jeweils testen und Ergebnis beurteilen!

Zaehlerprojekt entwickeln

Zulässige Assertions [Zusicherungen] (ab Python 2.7)

assertEqual(a, b)	a == b
assertNotEqual(a, b)	a != b
assertTrue(x)	Ausdruck x ist wahr
assertFalse(x)	Ausdruck x ist falsch
assertIs(a, b)	Objekte gleich
assertIsNot(a, b)	Objekte nicht gleich
assertIsNone(x)	ist None - Objekt
assertIsNotNone(x)	ist nicht ein None - Objekt
assertIn(a, b)	ist enthalten in (z.B. Liste)
assertNotIn(a, b)	ist nicht enthalten in (z.B. Liste)
assertIsInstance(a, b)	ist eine Instanz der Klasse
assertNotIsInstance(a, b)	ist keine Instanz der Klasse

Zaehlerprojekt entwickeln

Schritt 4

- Konkretisieren der Tests für EinfacherZaehler

```
def setUp(self):
    self.zaehler = EinfacherZaehler()

def test_Initialisierung(self):
    self.assertTrue(self.zaehler.GibStand()==0)

def test_Zaehlen(self):
    self.zaehler.Zaehle()
    self.assertTrue(self.zaehler.GibStand()==1)
    self.zaehler.Zaehle()
    self.assertTrue(self.zaehler.GibStand()==2)
```

Zaehlerprojekt entwickeln

Schritt 5

- Implementation der Methode GibStand()

```
def GibStand (self) :
      '''zeigt den Stand des Zaehlers an'''
      return self.__stand
```

Zaehlerprojekt entwickeln

Schritt 6

- Implementation der Methode Zaehle()

```
def Zaehle (self) :
    '''zaehlt den Zaehler jeweils um 1 weiter'''
    self.__stand = self.__stand + 1
```

In Kurzschreibweise:

```
    self.__stand += 1
```

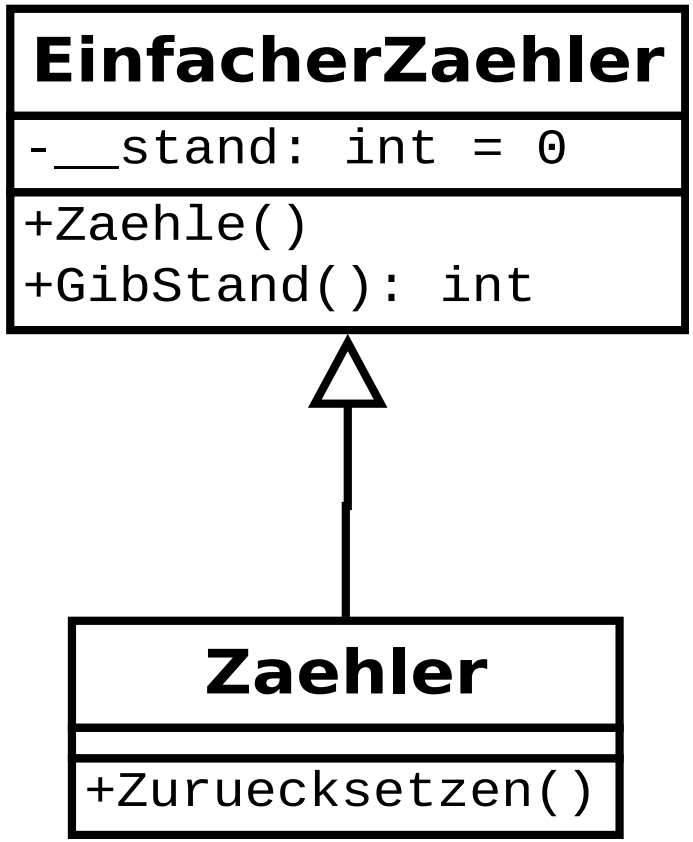
Zaehlerprojekt entwickeln

Abschluss von Schritt 6:

- Erneutes Ausführen der Testklasse
- gegebenenfalls Korrektur des Programmtextes

Zaehlerprojekt entwickeln

- Klassendiagramm mit Zaehler



Zaehlerprojekt entwickeln

Schritt 7

- Definition der Tests für Zaehler

```
def setUp(self):
    self.zaehler = Zaehler()
def test_Initialisierung(self):
    self.assertTrue(self.zaehler.GibStand()==0)

def test_Zaehle(self):
    self.zaehler.Zaehle()
    self.assertTrue(self.zaehler.GibStand()==1)
    self.zaehler.Zaehle()
    self.assertTrue(self.zaehler.GibStand()==2)
    self.zaehler.Zuruecksetzen()
    self.assertTrue(self.zaehler.GibStand()==0)
```

Zaehlerprojekt entwickeln

Schritt 8

- Bearbeiten des Konstruktors von Zaehler:

  Aufruf des Konstruktors von EinfacherZaehler

```
def __init__(self) :
    '''Konstruktor fuer ... Zaehler'''
    EinfacherZaehler.__init__(self)
```

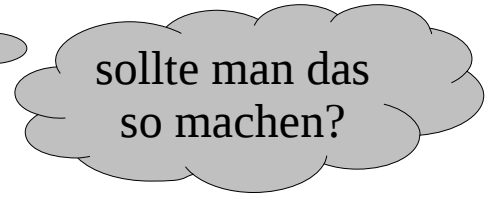
Zaehlerprojekt entwickeln

Schritt 9

- Implementation von Zuruecksetzen() in Zaehler:

```
class Zaehler(EinfacherZaehler):
    def __init__(self):
    '''Konstruktor fuer Objekte vom Typ Zaehler'''
        EinfacherZaehler.__init__(self)

    def Zuruecksetzen(self):
        '''setzt den Zaehler auf 0 zurueck'''
        self._EinfacherZaehler__stand = 0
```



self.__stand = 0 geht nicht!

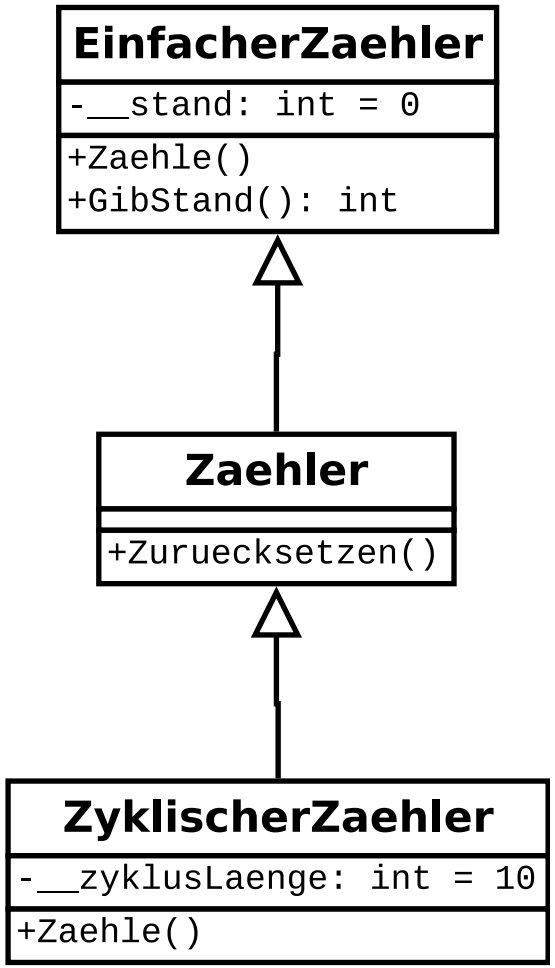
Zaehlerprojekt entwickeln

Abschluss von Schritt 9:

- Erneutes Ausführen der Testklasse
- gegebenenfalls Korrektur des Programmtextes

Zaehlerprojekt entwickeln

- Klassendiagramm zu ZyklischerZaehler



Zaehlerprojekt entwickeln

Schritt 10 (Folie 1)

- Definition der Tests für ZyklischerZaehler

```
class TestZyklischerZaehler(unittest.TestCase):

    def setUp(self):
        self.zaehler = ZyklischerZaehler(10)

    def test_Initialisierung(self):
        self.assertTrue(self.zaehler.GibStand()==0)

    def test_Zaehlen(self):
        ...
```

Zaehlerprojekt entwickeln

Schritt 10 (Folie 2)

- Definition der Tests für ZyklischerZaehler

```
...
    def test_Zaehlen(self):
        self.zaehler.Zaehle()
        self.assertTrue(self.zaehler.GibStand()==1)
        for i in range(8):
            self.zaehler.Zaehle()
        self.assertTrue(self.zaehler.GibStand()==9)
        self.zaehler.Zaehle()
        self.assertTrue(self.zaehler.GibStand()==0)
        self.zaehler.Zaehle()
        self.assertTrue(self.zaehler.GibStand()==1)
```

Zaehlerprojekt entwickeln

Schritt 11

- Korrektur des Konstruktors von ZyklischerZaehler:

  Aufruf des Konstruktors von Zaehler

```
def __init__(self, zyklusLaenge) :
     '''Konstruktor ... Zyklenlaenge'''
     Zaehler.__init__(self)
     self.__zyklusLaenge = zyklusLaenge # int
```

Zaehlerprojekt entwickeln

Schritt 12

- Implementation von Zaehle() in ZyklischerZaehler:

```
def Zaehle (self) :
    '''zaehlt den Zaehler ...
    setzt bei Erreichen ... auf 0 zurueck'''
    super().Zaehle()
    if self.GibStand()==self.__zyklusLaenge:
        self.Zuruecksetzen()
```

Zaehlerprojekt entwickeln

Hinweis nur für alte Pythonversionen

- "super() only works for new-style classes."
- "new-style class: Any class which inherits from object. "
- EinfacherZaehler muss von object erben!

Zaehlerprojekt entwickeln

Bedeutung der erneuten Implementation von Zaehle() in ZyklischerZaehler

- Die Methode Zaehle() in ZyklischerZaehler ***überschreibt*** (override) die gleichnamige Methode aus EinfacherZaehler, ersetzt sie also für Objekte vom Typ ZyklischerZaehler.
- Es ist ein Beispiel für ***Polymorphie***:
  Alle Zählertypen kennen eine Methode Zaehle(), verarbeiten sie aber in unterschiedlicher Weise

Zaehlerprojekt entwickeln

Abschluss von Schritt 12:

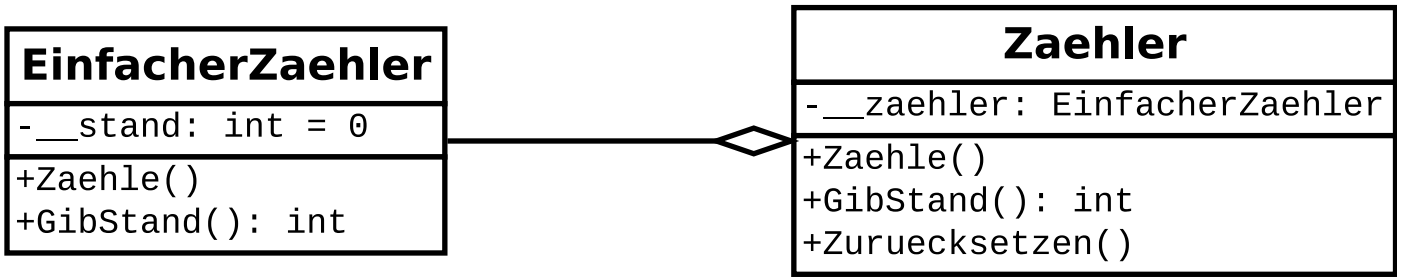
- Erneutes Ausführen der Testklasse
- gegebenenfalls Korrektur des Programmtextes

Zaehlerprojekt entwickeln

Was bleibt?

Es geht auch ohne Vererbung

- Zaehler alternativ mit Verwendungsbeziehung und mit Delegation statt mit Vererbung realisieren



Zaehlerprojekt entwickeln

- Zaehler mit Verwendungsbeziehung und mit Delegation realisieren

```
class Zaehler():
    '''Zaehler verwendet (benutzt) EinfacherZaehler'''

    def __init__(self):
        '''Konstruktor fuer Objekte vom Typ Zaehler'''
        self.__zaehler =  EinfacherZaehler()

    def Zuruecksetzen(self):
        '''erzeugt ein neues Objekt EinfacherZaehler'''
        self.__zaehler =  EinfacherZaehler()

    def Zaehle (self) :
        '''zaehlt den verwendeten Zaehler jeweils um 1 weiter'''
        self.__zaehler.Zaehle()

    def GibStand (self) :
        '''zeigt den Stand des verwendeten Zaehlers an'''
        return self.__zaehler.GibStand()
```

verwendet eine Instanz von EinfacherZaehler

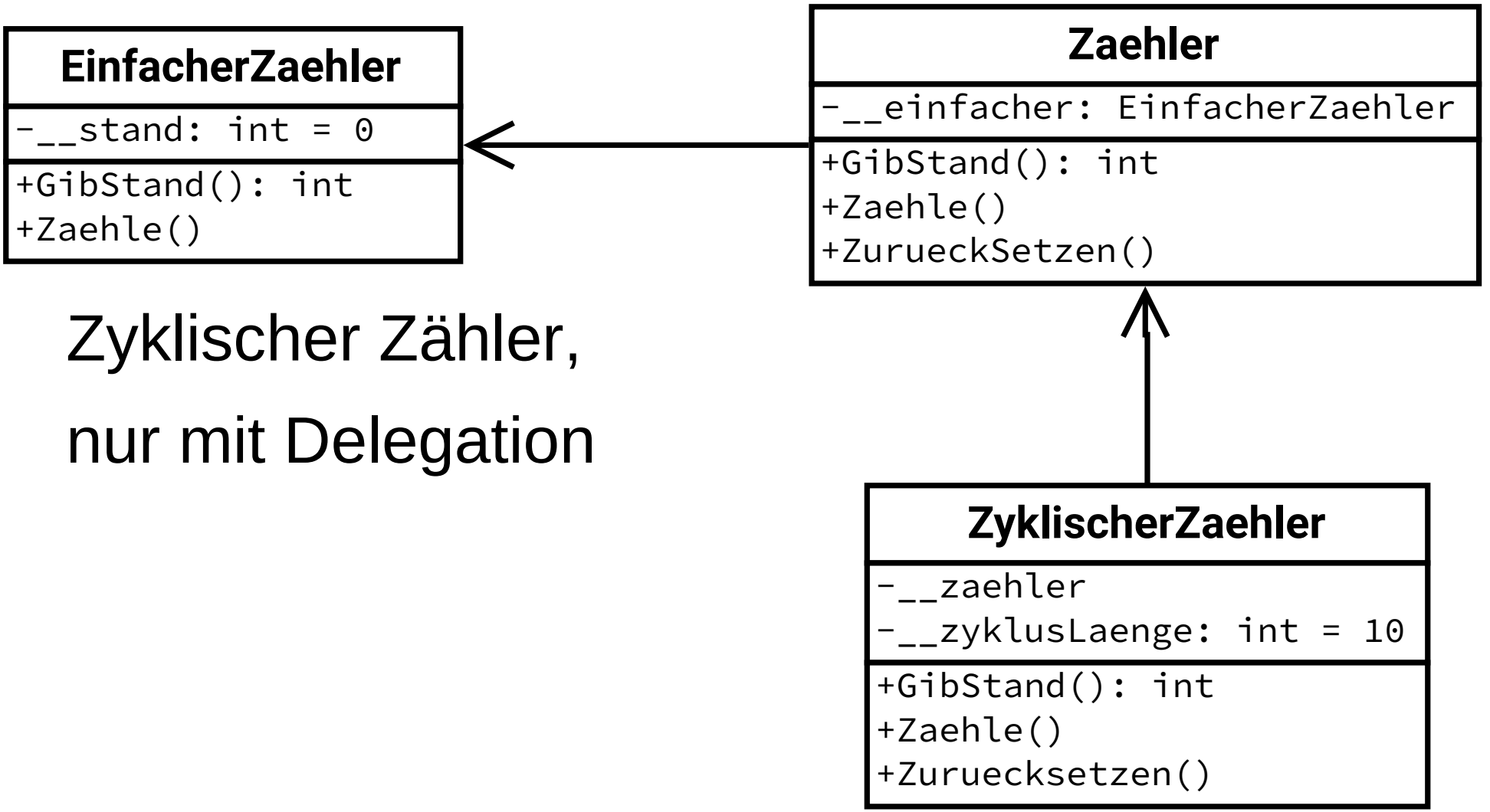
delegiert an das verwendete Objekt

Zaehlerprojekt entwickeln

Abschluss der Alternative:

- Erneutes Ausführen der Testklasse
- gegebenenfalls Korrektur des Programmtextes

Zaehlerprojekt entwickeln



Zyklischer Zähler,

nur mit Delegation