LU11.L01 - Schulverwaltung

Musterlösung zu LU11.A01 - Schulverwaltung (Repository m450-lu11-a01-school).

Bei Test Driven Development gibt es nicht **die** eine Lösung. Diese Musterlösung zeigt eine mögliche Umsetzung: zu jeder Klasse zuerst die Tests (Konstruktor zuerst), danach der Produktivcode. Abhängigkeiten zu anderen Klassen werden in den Tests durch einfache Mock-Klassen ersetzt (siehe LU11b - Mock).

Hinweise zur Umsetzung

- Im Repository heisst die Klasse `StudentReport` aus dem Klassendiagramm `ClassReport` (Datei `classreport.py`). Die Lösung behält den Namen aus dem Repository.
- Die Methoden `show_…` liefern gemäss Klassendiagramm einen **String** zurück. Die Vorlage `main.py` ruft noch `print_student_list()`, `print_student_report()` und `print_details()` auf. In der Lösung ist `main.py` an das Klassendiagramm angepasst und gibt die Strings mit `print` aus.
- Die Multiplizitäten aus dem Klassendiagramm werden geprüft: höchstens 20 Lernende pro Klasse und höchstens 4 Noten pro Fach. Wird die Grenze überschritten, löst die Methode einen `IndexError` aus.
- `Grade` ist eine `@dataclass`. Die Prüfung des Werts (1 bis 6) und die Umwandlung des Datums aus einem String erfolgen in `__setattr__`, damit sie sowohl im Konstruktor als auch beim späteren Setzen greifen.

Ergebnis

```
$ pytest -q
.............................
31 passed

$ python _run_pylint.py
Your code has been rated at 10.00/10
```

Code

grade.py

grade.py

```python
""" Eine Note mit Wert und Datum """
from dataclasses import dataclass
from datetime import datetime
```

```python
@dataclass
class Grade:
    """
    Eine Note zwischen 1.0 und 6.0 mit dem Datum der Prüfung
    :raises ValueError: wenn der Wert ausserhalb von 1 bis 6 liegt oder
das Datum ungültig ist
    """
    value: float = -1.0
    date: datetime = None

    def __setattr__(self, name, value):
        if name == 'value':
            if not 1.0 <= value <= 6.0:
                raise ValueError('Die Note muss zwischen 1 und 6
liegen')
        if name == 'date' and isinstance(value, str):
            value = datetime.strptime(value, '%d.%m.%y')
        super().__setattr__(name, value)
```

test_grade.py

test_grade.py

```python
""" Unit Tests für Grade """
from datetime import datetime

import pytest

from grade import Grade


def test_init():
    """ Konstruktor mit Datum als Text """
    grade = Grade(4.5, '1.2.25')
    assert grade.value == 4.5
    assert grade.date == datetime(2025, 2, 1)


def test_init_datetime():
    """ Konstruktor mit Datum als datetime """
    grade = Grade(6.0, datetime(2025, 3, 1))
    assert grade.date == datetime(2025, 3, 1)


@pytest.mark.parametrize('value', [1.0, 6.0])
def test_value_limits(value):
```

```
    """ Grenzwerte 1.0 und 6.0 sind gültig """
    assert Grade(value, '1.1.25').value == value


@pytest.mark.parametrize('value', [0.9, 6.1])
def test_value_invalid(value):
    """ Werte ausserhalb von 1 bis 6 werden abgelehnt """
    with pytest.raises(ValueError):
        Grade(value, '1.1.25')


def test_set_value_invalid():
    """ Auch der Setter prüft den Wert """
    grade = Grade(4.0, '1.1.25')
    with pytest.raises(ValueError):
        grade.value = 7.0
    assert grade.value == 4.0


def test_date_invalid():
    """ Ein ungültiges Datum wird abgelehnt """
    with pytest.raises(ValueError):
        Grade(4.0, '31.2.25')
```

subject.py

subject.py

```
""" Ein Schulfach mit seinen Noten """


class Subject:
    """
    Ein Fach mit höchstens 4 Noten
    """
    MAX_GRADES = 4

    def __init__(self, name):
        self._name = name
        self._grades = []

    def add_grade(self, grade):
        """
        Fügt eine Note hinzu
        :param grade: die Note
        :raises IndexError: wenn das Fach bereits 4 Noten hat
        """
        if len(self._grades) >= self.MAX_GRADES:
```

```
            raise IndexError('Ein Fach hat höchstens 4 Noten')
        self._grades.append(grade)

    def take_grade(self, index):
        """
        Liefert die Note an der Position index
        :param index: die Position
        :return: die Note
        """
        return self._grades[index]

    def count_grades(self):
        """ Liefert die Anzahl Noten """
        return len(self._grades)

    @property
    def average(self):
        """ Der Durchschnitt aller Noten, 0.0 wenn keine Note vorhanden
ist """
        if not self._grades:
            return 0.0
        total = 0.0
        for grade in self._grades:
            total += grade.value
        return total / len(self._grades)

    @property
    def name(self):
        """ Der Name des Fachs """
        return self._name
```

test_subject.py

test_subject.py

```
""" Unit Tests für Subject – Grade wird durch einen Mock ersetzt """
import pytest

from subject import Subject


class GradeMock:  # pylint: disable=too-few-public-methods
    """ Mock für Grade: nur der Wert wird gebraucht """

    def __init__(self, value):
        self.value = value
```

```python
@pytest.fixture(name='subject')
def fixture_subject():
    """ Ein Fach ohne Noten """
    return Subject('Mathe')


def test_init(subject):
    """ Konstruktor """
    assert subject.name == 'Mathe'
    assert subject.count_grades() == 0


def test_add_take_grade(subject):
    """ Noten hinzufügen und wieder lesen """
    first = GradeMock(4.0)
    second = GradeMock(5.5)
    subject.add_grade(first)
    subject.add_grade(second)
    assert subject.count_grades() == 2
    assert subject.take_grade(1) is second


def test_add_grade_too_many(subject):
    """ Ein Fach hat höchstens 4 Noten """
    for value in [4.0, 4.5, 5.0, 5.5]:
        subject.add_grade(GradeMock(value))
    with pytest.raises(IndexError):
        subject.add_grade(GradeMock(6.0))
    assert subject.count_grades() == 4


def test_take_grade_invalid(subject):
    """ Eine Position ohne Note """
    with pytest.raises(IndexError):
        subject.take_grade(0)


def test_average(subject):
    """ Durchschnitt mehrerer Noten """
    for value in [4.0, 4.5, 5.5]:
        subject.add_grade(GradeMock(value))
    assert subject.average == pytest.approx(4.6667, abs=0.0001)


def test_average_empty(subject):
    """ Ohne Noten ist der Durchschnitt 0.0 """
    assert subject.average == 0.0
```

classreport.py

classreport.py

```
""" Das Zeugnis eines Lernenden (im Klassendiagramm: StudentReport) """


class ClassReport:
    """
    Ein Zeugnis mit den Fächern eines Lernenden
    """

    def __init__(self):
        self._subjects = []
        self._student = None

    def show_overview(self):
        """
        Liefert das Zeugnis mit allen Fächern und dem Notenschnitt
        :return: das Zeugnis als Text
        """
        lines = []
        for subject in self._subjects:
            lines.append(f'{subject.name:<10}{subject.average:4.1f}')
        return '\n'.join(lines)

    def show_details(self):
        """
        Liefert alle Fächer mit den einzelnen Noten
        :return: die Noten als Text
        """
        lines = []
        for subject in self._subjects:
            grades = []
            for index in range(subject.count_grades()):
grades.append(f'{subject.take_grade(index).value:3.1f}')
            separator = ' '
            lines.append(f'{subject.name:<10}{separator.join(grades)}')
        return '\n'.join(lines)

    def add_subject(self, subject):
        """ Fügt ein Fach hinzu """
        self._subjects.append(subject)

    def take_subject(self, index):
        """ Liefert das Fach an der Position index """
        return self._subjects[index]

    def count_subjects(self):
```

```
        """ Liefert die Anzahl Fächer """
        return len(self._subjects)

    @property
    def student(self):
        """ Der Lernende, dem das Zeugnis gehört """
        return self._student

    @student.setter
    def student(self, student):
        self._student = student
```

test_classreport.py

test_classreport.py

```
""" Unit Tests für ClassReport – Subject und Student werden durch Mocks
ersetzt """
import pytest

from classreport import ClassReport


class GradeMock:  # pylint: disable=too-few-public-methods
    """ Mock für Grade """

    def __init__(self, value):
        self.value = value


class SubjectMock:
    """ Mock für Subject mit fixen Noten """

    def __init__(self, name, values):
        self.name = name
        self.grades = [GradeMock(value) for value in values]
        self.average = sum(values) / len(values)

    def count_grades(self):
        """ Anzahl Noten """
        return len(self.grades)

    def take_grade(self, index):
        """ Note an der Position index """
        return self.grades[index]


@pytest.fixture(name='report')
```

```
def fixture_report():
    """ Ein Zeugnis mit zwei Fächern """
    report = ClassReport()
    report.add_subject(SubjectMock('Mathe', [4.0, 5.0]))
    report.add_subject(SubjectMock('Deutsch', [5.5, 6.0]))
    return report


def test_init():
    """ Konstruktor: leeres Zeugnis ohne Lernenden """
    report = ClassReport()
    assert report.count_subjects() == 0
    assert report.student is None


def test_add_take_subject(report):
    """ Fächer hinzufügen und lesen """
    assert report.count_subjects() == 2
    assert report.take_subject(1).name == 'Deutsch'


def test_take_subject_invalid(report):
    """ Eine Position ohne Fach """
    with pytest.raises(IndexError):
        report.take_subject(2)


def test_student(report):
    """ Setter und Getter für den Lernenden """
    student = object()
    report.student = student
    assert report.student is student


def test_show_overview(report):
    """ Übersicht mit Fach und Schnitt """
    assert report.show_overview() == 'Mathe      4.5\nDeutsch    5.8'


def test_show_details(report):
    """ Details mit allen Noten """
    assert report.show_details() == 'Mathe     4.0 5.0\nDeutsch   5.5
6.0'
```

student.py

student.py

```
""" Ein Lernender mit seinem Zeugnis """


class Student:
    """
    Ein Lernender gehört zu einer Klasse und hat ein Zeugnis
    """

    def __init__(self, name, report):
        self._name = name
        self._school_class = None
        self._report = report
        report.student = self

    def show_report(self):
        """ Liefert die Übersicht des Zeugnisses """
        return self._report.show_overview()

    @property
    def name(self):
        """ Der Name des Lernenden """
        return self._name

    @property
    def school_class(self):
        """ Die Klasse des Lernenden """
        return self._school_class

    @school_class.setter
    def school_class(self, school_class):
        self._school_class = school_class

    @property
    def report(self):
        """ Das Zeugnis des Lernenden """
        return self._report
```

test_student.py

test_student.py

```
""" Unit Tests für Student – das Zeugnis wird durch einen Mock ersetzt
"""
import pytest

from student import Student
```

```
class ReportMock:  # pylint: disable=too-few-public-methods
    """ Mock für ClassReport """

    def __init__(self):
        self.student = None

    def show_overview(self):
        """ Liefert immer denselben Text """
        return 'Übersicht'


@pytest.fixture(name='report')
def fixture_report():
    """ Ein leeres Zeugnis """
    return ReportMock()


def test_init(report):
    """ Konstruktor: Name, Zeugnis, noch keine Klasse """
    student = Student('Pia', report)
    assert student.name == 'Pia'
    assert student.report is report
    assert student.school_class is None


def test_init_links_report(report):
    """ Das Zeugnis kennt danach seinen Lernenden """
    student = Student('Pia', report)
    assert report.student is student


def test_school_class(report):
    """ Setter und Getter für die Klasse """
    student = Student('Pia', report)
    school_class = object()
    student.school_class = school_class
    assert student.school_class is school_class


def test_show_report(report):
    """ show_report liefert die Übersicht des Zeugnisses """
    assert Student('Pia', report).show_report() == 'Übersicht'
```

school_class.py

school_class.py

```
""" Eine Schulklasse mit ihren Lernenden """
```

```
class SchoolClass:
    """
    Eine Klasse mit höchstens 20 Lernenden
    """
    MAX_STUDENTS = 20

    def __init__(self, designation):
        self._designation = designation
        self._students = []

    def add_student(self, student):
        """
        Nimmt einen Lernenden in die Klasse auf
        :param student: der Lernende
        :raises IndexError: wenn die Klasse bereits 20 Lernende hat
        """
        if len(self._students) >= self.MAX_STUDENTS:
            raise IndexError('Eine Klasse hat höchstens 20 Lernende')
        self._students.append(student)
        student.school_class = self

    def take_student(self, index):
        """ Liefert den Lernenden an der Position index """
        return self._students[index]

    def count_students(self):
        """ Liefert die Anzahl Lernende """
        return len(self._students)

    def show_student_list(self):
        """ Liefert die Klassenliste: die Bezeichnung und alle Namen
"""
        lines = [f'Klasse {self._designation}']
        for student in self._students:
            lines.append(student.name)
        return '\n'.join(lines)

    def show_student_report(self, name):
        """
        Liefert das Zeugnis eines Lernenden
        :param name: der Name des Lernenden
        :return: das Zeugnis oder eine Meldung, wenn der Name unbekannt
ist
        """
        for student in self._students:
            if student.name == name:
                return student.show_report()
        return f'{name} ist nicht in der Klasse {self._designation}'
```

```
    @property
    def designation(self):
        """ Die Bezeichnung der Klasse """
        return self._designation
```

test_school_class.py

test_school_class.py

```
""" Unit Tests für SchoolClass – Student wird durch einen Mock ersetzt
"""
import pytest

from school_class import SchoolClass


class StudentMock:  # pylint: disable=too-few-public-methods
    """ Mock für Student """

    def __init__(self, name):
        self.name = name
        self.school_class = None

    def show_report(self):
        """ Liefert einen Text mit dem Namen """
        return f'Zeugnis {self.name}'


@pytest.fixture(name='school_class')
def fixture_school_class():
    """ Eine Klasse mit zwei Lernenden """
    school_class = SchoolClass('IM25a')
    school_class.add_student(StudentMock('Max'))
    school_class.add_student(StudentMock('Pia'))
    return school_class


def test_init():
    """ Konstruktor: Bezeichnung, keine Lernenden """
    school_class = SchoolClass('IM25a')
    assert school_class.designation == 'IM25a'
    assert school_class.count_students() == 0


def test_add_take_student(school_class):
    """ Lernende hinzufügen und lesen """
    assert school_class.count_students() == 2
    assert school_class.take_student(1).name == 'Pia'
```

```
def test_add_student_sets_class(school_class):
    """ Der Lernende kennt danach seine Klasse """
    assert school_class.take_student(0).school_class is school_class


def test_add_student_full():
    """ Eine Klasse hat höchstens 20 Lernende """
    school_class = SchoolClass('IM25a')
    for number in range(20):
        school_class.add_student(StudentMock(f'S{number}'))
    with pytest.raises(IndexError):
        school_class.add_student(StudentMock('Zuviel'))
    assert school_class.count_students() == 20


def test_show_student_list(school_class):
    """ Klassenliste mit Bezeichnung und Namen """
    assert school_class.show_student_list() == 'Klasse IM25a\nMax\nPia'


def test_show_student_report(school_class):
    """ Zeugnis eines bekannten Lernenden """
    assert school_class.show_student_report('Pia') == 'Zeugnis Pia'


def test_show_student_report_unknown(school_class):
    """ Unbekannter Name """
    assert school_class.show_student_report('Theo') == 'Theo ist nicht
in der Klasse IM25a'
```

main.py

main.py

```
""" Zeigt, wie die Klassen der Schulverwaltung zusammenspielen """
from classreport import ClassReport
from student import Student
from school_class import SchoolClass
from subject import Subject
from grade import Grade


def create_subject_list(actual_report):
    """ Fügt einem Zeugnis die drei Fächer hinzu """
    actual_report.add_subject(Subject('Mathe'))
    actual_report.add_subject(Subject('Deutsch'))
```

```
    actual_report.add_subject(Subject('Turnen'))


def add_grades(report, values):
    """ Fügt die Noten pro Fach hinzu: values ist eine Liste mit einer
Liste pro Fach """
    for index, subject_values in enumerate(values):
        for value in subject_values:
            report.take_subject(index).add_grade(Grade(value,
'1.1.25'))


def main():
    """ Erstellt eine Klasse mit drei Lernenden und gibt die Zeugnisse
aus """
    the_class = SchoolClass('IM993a')
    reports = {}
    for name in ['Max', 'Pia', 'Cem']:
        reports[name] = ClassReport()
        create_subject_list(reports[name])
        the_class.add_student(Student(name, reports[name]))
    print(the_class.show_student_list())

    add_grades(reports['Max'], [[4.0, 4.5], [4.0, 6.0, 5.0], [4.5, 5.0,
5.0, 5.5]])
    add_grades(reports['Pia'], [[5.5, 5.5], [4.5, 6.0, 5.5], [4.0, 5.5,
6.0, 5.5]])
    add_grades(reports['Cem'], [[5.0, 3.5], [5.5, 6.0, 5.0], [4.5, 6.0,
6.0, 5.5]])

    for name in ['Max', 'Pia', 'Cem', 'Theo']:
        print(the_class.show_student_report(name))
    print(reports['Cem'].show_details())


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

M450-LU11, M450-C2I, M450-C1A

Marcel Suter

From:
https://wiki.bzz.ch/ - **BZZ - Modulwiki**

Permanent link:
https://wiki.bzz.ch/modul/m450/learningunits/lu11/loesungen/school

Last update: **2026/11/17 17:38**