LU09.L01 - Schulverwaltung

grade.py

```
""" Provides the Grade-class"""
from datetime import datetime
from dataclasses import dataclass


@dataclass
class Grade:
    """
    A single grade for a student in one subject.
    All attribute values are set during construction of the object.

    Attributes
    ----------
    value: float
        the value of the grade must be >= 1.0 and <= 6.0
    date: datetime
        The date/time the grade was set
    """
    value: float = -1.0
    date: datetime = None

    def __post_init__(self):
        """
        validates the initial value
        :raises: ValueError: if the value is out of bounds
        """
        if self._value > 6.0 or self._value < 1.0:
            raise ValueError


    @property
    def value(self):
        """ returns the value for this grade"""
        return self._value


    @value.setter
    def value(self, value):
        """ sets the value for this grade """
        self._value = value


    @property
    def date(self):
        """ returns the date of this grade """
        return self._date


    @date.setter
    def date(self, value):
        """
        sets the date of this grade.
        If a string "(d)d.(m)m.(yy)yy" is provided, it converts it to
DateTime
        :param: value(mixed): The date or None=now
        """

        if isinstance(value, datetime):
            self._date = value
        elif isinstance(value, str) and value != '':
            self._date = datetime.strptime(value, '%d.%m.%y')
        else:
            self._date = datetime.now()
```

subject.py

```
""" Provides the Subject-class """

class Subject:
    """
    A school subject for the grade report of one student

    Attributes
    ----------
    name: String
        The subject name
    grades: List
        A list of grades
    """

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

    def add_grade(self, grade):
        """
        adds a grade to the grades list
        :param: grade(Grade) the grade-object to add
        :raises: OverflowError  if more than 4 grades are added to the
list
        """
        if self.count_grades() < 4:
            self._grades.append(grade)
        else:
            raise OverflowError

    def take_grade(self, index):
        """
        returns the grade identified by the index
        :param: index(int):  the index of the grade
        :raises: IndexError: if the index does not exist
        """
        if index < self.count_grades():
            return self._grades[index]
        raise IndexError

    def count_grades(self):
        """
        counts the number of grades in the list
        :return: size of the list(int)
        """
        return len(self._grades)

    @property
    def average(self):
        """
        calculates the average of all grades in the list or 0 if the
grades-list ist empty
        :return: average grade(float)
        """
        if self.count_grades() == 0:
            return 0.0
        else:
            total = 0.0
            for number in range(self.count_grades()):
                total += self.take_grade(number).value
            return total / self.count_grades()

    @property
    def name(self):
        """ returns the name """
        return self._name
```

student.py

```
""" Provides the student-object """


class Student:
    """
    A student in a schoolclass with subjects and grades

    Attributes
    ----------
    name: String
        The fullname of the student
    school_class: SchoolClass
        The schoolclass this student is part of
    report: StudentReport
        The report-object with the subjects and grades for this student
    """

    def __init__(self, name, student_report):
        """
        creates the object with references to the schoolclass and
studentreport
        :param report: Referenz zum Zeugnis
        """
        self._name = name
        # create the two-way relationship between student and
studentreport
        self._report = student_report
        student_report.student = self
        self._school_class = None  # this reference will be set later

    def show_report(self):
        """ returns the report for this student """
        return self.report

    @property
    def name(self):
        """
        Liefert den Namen des Studenten
        :return: Name des Studenten
        """
        return self._name

    @property
    def school_class(self):
        """
        Liefert die Referenz der Klasse
        :return: Referenz der Klasse
        """
        return self._school_class

    @school_class.setter
    def school_class(self, school_class):
        """
        sets the reference to the schoolclass
        """
        self._school_class = school_class

    @property
    def report(self):
        """
        gets the reference to the studentreport
        """
        return self._report
```

studentreport.py

```
""" Provides the StudentReport-class """
class StudentReport:
    """
    The grade reports for a student with the subjects and grades
    """
    def __init__(self):
        """
        initializes the student report with empty attributs
        """
        self._subjects = []
        self._student = None

    def show_overview(self):
        """
        shows an overview of all subjects and average marks for the
student.

        :return: overview report
        """
        output = 'Zeugnis für: '
        # Needs a student-object to show the name.
        if self._student is not None:
            output += self._student.name

        # List all subjects with the average mark.
        for subject in self._subjects:
            output += f'\n\t  {subject.name:<10}:
{subject.average:.2f}'
        return output

    def show_details(self):
        """
        shows the details of all subjects and marks for the student.

        :return: detailed report (str)
        """
        output = ''
        for subject in self._subjects:
            output += f'Fach: {subject.name:<10} mit
{subject.count_grades()} Noten\n'
            for count in range(subject.count_grades()):
                grade = subject.take_grade(count)
                output += f' - {count + 1}: {grade.value:.2f} am
```

```
{grade.date:%d.%m.%Y}\n'
            output += f' Schnitt: {subject.average:.2f}\n'
        return output

    def add_subject(self, subject):
        """
        adds a subject to the list, if there are less than 3 subjects
in the list

        :param: subject (Subject): the new subject to be added.
        :raises: OverflowError: if the list is already full
        """
        if len(self._subjects) < 3:
            self._subjects.append(subject)
        else:
            raise OverflowError

    def take_subject(self, index):
        """
        returns the subject at the specified index, if it exists.

        :param: index (int): the index of the subject.
        :return: Subject or None
        :raises: IndexError  if the index doesn't exists in the list
        """
        if index < len(self._subjects):
            return self._subjects[index]
        raise IndexError

    def count_subjects(self):
        """
        counts the number of subjects in the list

        :return: length (int)
        """
        return len(self._subjects)

    @property
    def student(self):
        """ returns the student-object """
        return self._student

    @student.setter
    def student(self, value):
        """ Sets the student-object """
        self._student = value
```

school_class.py

```python
""" Provides the class SchoolClass """


class SchoolClass:
    """
    A schoolclass with the students

    Attributes
    ----------
    students: List
        a list of student-objects
    designation: String
        the designation of this schoolclass
    """

    def __init__(self, designation):
        """
        constructs the object
        :param: designation(string): the designation of this
schoolclass
        """
        self._designation = designation
        self._students = []

    def add_student(self, student):
        """
        adds a student to the students list
        :param: student(Student) the student-object to add
        :raises: OverflowError:  if the student-list is already full
        """
        if self.count_students() < 20:
            self._students.append(student)
            student.school_class = self
        else:
            raise OverflowError

    def take_student(self, index):
        """
        returns the student identified by the index
        :param: index(int):  the index of the student
        :raises: IndexError: if the index does not exist
        """
        if index < self.count_students():
            return self._students[index]
        raise IndexError

    def count_students(self):
        """
        counts the number of students in the list
        :return: size of the list(int)
        """
        return len(self._students)

    def show_student_list(self):
        """ shows a list of all student names """
        output = ''
        for student in self._students:
            output += f'{student.name}\n'
        return output

    def show_student_report(self, name):
        """
        Shows the grades for the student identified by his name
        :param name: The name of the student to be shown
        :return:
        """
        print('----')
        for student in self._students:
            if student.name == name:
                return student.report.show_overview()
        return f'Student {name} nicht gefunden'

    @property
    def designation(self):
        """ returns the designation """
        return self._designation
```