Homework 10

01286233 Web Programming

Software Engineering Program,

Department of Computer Engineering,

School of Engineering, KMITL

67011352 Theepakorn Phayonrat

Code

HW10_67011352_Theepakorn.py

```
import persistent
from typing import Optional

class Course (persistent.Persistent):
    def __init__(self, id, name: str = "", credit: int = 0) ->
      ↪  None:
        self.id = id
        self.name = name
        self.credit = credit
        self.grading = [
            {"Grade": "A", "min": 80, "max": 100},
            {"Grade": "B", "min": 70, "max": 79},
            {"Grade": "C", "min": 60, "max": 69},
            {"Grade": "D", "min": 50, "max": 59},
            {"Grade": "F", "min": 0, "max": 49},
        ]

    def __str__(self) -> str:
        return f"ID: {self.id:<6} Course: {self.name:<30} Credit:
          ↪  {self.credit:<2}"

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

    def getCredit(self) -> int:
        return self.credit

    def printDetail(self) -> None:
        print(self.__str__())

    def scoreGrading(self, score) -> str:
        for g in self.grading:
            if g["min"] <= score and score <= g["max"]:
                return g["Grade"]
        return "F"

    def setGradeScheme(self, scheme: list) -> None:
```

```
        self.grading = scheme

class Student(persistent.Persistent):
    def __init__(self, id: int, name: str) -> None:
        self.enrolls = []
        self.id = id
        self.name = name

    def enrollCourse(self, course: Course) -> "Enrollment":
        e = Enrollment(course = course, student = self)
        self.enrolls.append(e)
        return e

    def getEnrollment(self, course) -> Optional["Enrollment"]:
        for e in self.enrolls:
            if e.course == course:
                return e
        return None

    def printTranscript(self) -> None:
        print("<-=- Transcript -=->")
        print(f"ID: {self.id:<6} Name: {self.name}")
        print("Course list")
        for e in self.enrolls:
            e.printDetail()
        gpa = self.getGPA()
        print(f"Total GPA is: {gpa:.2f}")

    def getGPA(self) -> float:
        numerator = 0
        denominator = 0
        for e in self.enrolls:
            text = e.getGrade()
            if text == "A":
                numerator += 4 * e.course.getCredit()
                denominator += e.course.getCredit()
            # elif text == "B+":
            #     numerator += 3.5 * e.course.getCredit()
                # denominator += e.course.getCredit()
            elif text == "B":
                numerator += 3 * e.course.getCredit()
```

```
                denominator += e.course.getCredit()
            # elif text == "C+":
            #     numerator += 2.5 * e.course.getCredit()
            #     denominator += e.course.getCredit()
            elif text == "C":
                numerator += 2 * e.course.getCredit()
                denominator += e.course.getCredit()
            # elif text == "D+":
            #     numerator += 1.5 * e.course.getCredit()
            #     denominator += e.course.getCredit()
            elif text == "D":
                numerator += 1 * e.course.getCredit()
                denominator += e.course.getCredit()
            else:
                denominator += e.course.getCredit()
        return numerator/denominator



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

class Enrollment(persistent.Persistent):
    # def __init__(self, course: Course, student: Student, grade:
      ↪  str = "", score = 0) -> None:
    def __init__(self, course: Course, student: Student, score = 0)
      ↪  -> None:
        self.course = course
        # self.grade = grade
        self.student = student
        self.score = score

    def getCourse(self) -> Course:
        return self.course

    # def getGrade(self) -> str:
    #     return self.grade

    def getGrade(self) -> str:
        return self.course.scoreGrading(self.score)
```

```
    def geScore(self) -> int:
        return self.score

    def printDetail(self) -> None:
        print(f"ID: {self.course.id:<6} Course:
           ↪  {self.course.name:<30} "
            f"Credit: {self.course.getCredit():<2} Score:
              ↪  {self.geScore():<2} Grade: {self.getGrade():<2}")

    # def setGrade(self, grade: str) -> None:
    #     self.grade = grade

    def setScore(self, score: int) -> None:
        self.score = score


# Testing


import BTrees._OOBTree
import ZODB, ZODB.FileStorage
import transaction

storage = ZODB.FileStorage.FileStorage("mydata.fs")
db = ZODB.DB(storage)
connection = db.open()
root = connection.root

if __name__ == "__main__":

    #Added Data
    root.courses = BTrees._OOBTree.BTree()
    root.courses[101] = Course(101, "Computer Programming", 4)
    root.courses[201] = Course(201, "Web Programmin", 4)
    root.courses[202] = Course(202, "Software Engineering
       ↪  Principle", 5)
    root.courses[301] = Course(301, "Artificial Intelligent", 3)

    root.courses[202].setGradeScheme([
            {"Grade": "A", "min": 90, "max": 100},
            {"Grade": "B", "min": 75, "max": 89},
```

```
        {"Grade": "C", "min": 60, "max": 74},
        {"Grade": "D", "min": 50, "max": 59},
        {"Grade": "F", "min": 0, "max": 49},
])

root.courses[301].setGradeScheme([
        {"Grade": "A", "min": 90, "max": 100},
        {"Grade": "B", "min": 75, "max": 89},
        {"Grade": "C", "min": 56, "max": 74},
        {"Grade": "D", "min": 50, "max": 55},
        {"Grade": "F", "min": 0, "max": 49},
])

root.students = BTrees._OOBTree.BTree()
root.students[1101] = Student(1101, "Mr. Christian de
  ↪ Nenvillette")
root.students[1101].enrollCourse(root.courses[101]).setScore(3
  ↪ 4)
root.students[1101].enrollCourse(root.courses[201]).setScore(8
  ↪ 8)
root.students[1101].enrollCourse(root.courses[301]).setScore(6
  ↪ 9)

root.students[1102] = Student(1102, "Mr. Zhong Li")
root.students[1102].enrollCourse(root.courses[101]).setScore(1
  ↪ 00)
root.students[1102].enrollCourse(root.courses[201]).setScore(9
  ↪ 9)
root.students[1102].enrollCourse(root.courses[202]).setScore(6
  ↪ 6)

root.students[1103] = Student(1103, "Mr. Dvalinn Durinson")
root.students[1103].enrollCourse(root.courses[101]).setScore(1
  ↪ 01)
root.students[1103].enrollCourse(root.courses[201]).setScore(6
  ↪ 9)
root.students[1103].enrollCourse(root.courses[202]).setScore(3
  ↪ 4)
root.students[1103].enrollCourse(root.courses[301]).setScore(5
  ↪ 6)
```

```
root.students[1110] = Student(1110, "Mr. Name ForExample")
root.students[1110].enrollCourse(root.courses[101]).setScore(7
   ↪ 5)
root.students[1110].enrollCourse(root.courses[201]).setScore(8
   ↪ 1)
root.students[1110].enrollCourse(root.courses[202]).setScore(8
   ↪ 1)
root.students[1110].enrollCourse(root.courses[301]).setScore(5
   ↪ 7)

transaction.commit()


# Print Data
courses = root.courses
for c in courses:
    course = courses[c]
    course.printDetail()
print()

students = root.students
for s in students:
    student = students[s]
    student.printTranscript()
    print()
transaction.commit()
```

Result

0.0.1 Setup and Run

```
python -m venv .venv
source .venv/bin/activate
pip install ZODB
python HW10_67011352_Theepakorn.py
```

0.0.2 Output

```
ID: 101    Course: Computer Programming           Credit: 4
ID: 201    Course: Web Programmin                 Credit: 4
ID: 202    Course: Software Engineering Principle Credit: 5
ID: 301    Course: Artificial Intelligent         Credit: 3

<-=- Transcript -=->
ID: 1101   Name: Mr. Christian de Nenvillette
Course list
ID: 101    Course: Computer Programming           Credit: 4  Score: 34 Grade: F
ID: 201    Course: Web Programmin                 Credit: 4  Score: 88 Grade: A
ID: 301    Course: Artificial Intelligent         Credit: 3  Score: 69 Grade: C
Total GPA is: 2.00

<-=- Transcript -=->
ID: 1102   Name: Mr. Zhong Li
Course list
ID: 101    Course: Computer Programming           Credit: 4  Score: 100 Grade: A
ID: 201    Course: Web Programmin                 Credit: 4  Score: 99 Grade: A
ID: 202    Course: Software Engineering Principle Credit: 5  Score: 66 Grade: C
Total GPA is: 3.23

<-=- Transcript -=->
ID: 1103   Name: Mr. Dvalinn Durinson
Course list
ID: 101    Course: Computer Programming           Credit: 4  Score: 101 Grade: F
ID: 201    Course: Web Programmin                 Credit: 4  Score: 69 Grade: C
ID: 202    Course: Software Engineering Principle Credit: 5  Score: 34 Grade: F
ID: 301    Course: Artificial Intelligent         Credit: 3  Score: 56 Grade: C
Total GPA is: 0.88

<-=- Transcript -=->
ID: 1110   Name: Mr. Name ForExample
Course list
ID: 101    Course: Computer Programming           Credit: 4  Score: 75 Grade: B
ID: 201    Course: Web Programmin                 Credit: 4  Score: 81 Grade: A
ID: 202    Course: Software Engineering Principle Credit: 5  Score: 81 Grade: B
ID: 301    Course: Artificial Intelligent         Credit: 3  Score: 57 Grade: C
Total GPA is: 3.06
```