Assignment 6

Goals

The goal of this assignment is to work with object-oriented programming in Python.

Instructions

You will be doing your work in Python for this assignment. You may choose to work on this assignment on a hosted environment (e.g. [tiger](https://tiger.cs.niu.edu/jupyter/)) or on your own local installation of Jupyter and Python. You should use Python 3.8 or higher for your work. To use tiger, use the credentials you received. If you work remotely, make sure to download the .py files to turn in. If you choose to work locally, [Anaconda](https://www.anaconda.com/download) is the easiest way to install and manage Python. If you work locally, you may launch Jupyter Lab either from the Navigator application or via the command-line as jupyter-lab.

In this assignment, we will be implementing a collection of classes that work together to orchestrate course scheduling in a university setting. There will be a central registrar that coordinates scheduling for students and professors with methods to add and remove persons from courses. You will need to test your classes to make sure they work properly (See Section 6 for a summary of the items to be implemented and some non-comprehensive test code.)

Due Date

The assignment is due at 11:59pm on Tuesday, November 2.

Submission

You should submit the completed notebook file required for this assignment on [Blackboard](https://webcourses.niu.edu/). The filename of the notebook should be a6.ipynb.

Details

Please make sure to follow instructions to receive full credit. Because you will be writing classes and adding to them, you do not need to separate each part of the assignment. Please document any shortcomings with your code. You may put the code for each part into one or more cells. Note that CS 503 Students must complete Section 5 which is optional for CS 490 students.

0. Name & Z-ID (5 pts)

The first cell of your notebook should be a markdown cell with a line for your name and a line for your Z-ID. If you wish to add other information (the assignment name, a description of the assignment), you may do so after these two lines.

1. Academic Classes (20 pts)

Create four classes related to the people involved in university course scheduling: Academic, Student, GraduateStudent, and Instructor. Academic is the base class which has a first name, last name, and campus\_id. It also has a field max\_credits which should be initialized to the default value for the type of person (more on this later in the paragraph). While it will be used as an abstract base class (not intended to be instantiated), you do not need to flag it as such (but may). Student is an Academic that has a level (“Freshman”, …, “Senior”), and GraduateStudent is a Student whose level is always “Graduate”. Instructor is an Academic that has a rank (“Assistant Professor”, …, “Instructor”). You must use **inheritance** properly here. Each student should have a default maximum number of credits of 16, each graduate student a default maximum of 12, and each instructor a default maximum of 9. Use a class attribute, MAX\_CREDITS to store this default, but **only one** place in the code should assign to the max\_credits field.

Hints:

* Make use of the base class’s constructor. Don’t rewrite what it is already doing.
* Remember when accessing a class attribute, self will look up attributes according to the MRO so with multiple definitions of MAX\_CREDITS, it will “do the expected thing”.

2. Course Class (15 pts)

Create a Course class that stores information about a course: department, course number, course name, section, number of credits, times. The course class also keeps track of the instructor (which may be unassigned) and all students enrolled. Note that times will be submitted as a list of tuples of the form (day : str, start\_time : int, end\_time : int). Add a method to produce a string representation that lists the department, course number, and section along with the times. The class should have enroll and drop methods to add/remove a student from the course, and a change\_time method to update the times of the course.

Hints:

* When fields are tracked but not assigned in the constructor, make sure they are still initialized.
* Unlike some other languages, Python allows multiple comparisons without boolean operators: a < b < c checks that b is between a and c.

3. Schedule Class (20 pts)

Create a Schedule class that tracks all courses an Academic is involved in. The constructor should take an optional list of courses to be added to the schedule. It should have a credits **property** (getter only) that returns the sum of the number of credits in all courses. It should have an add\_course method that adds the course to the schedule  (check is only for CS 503, see Section 5)

Update the Academic class to add a schedule to each instance plus add\_course and remove\_course methods. The add\_course method should first check if the course to be added will exceed the maximum number of credits for the person. If it doesn’t, the course should be added to the schedule. If it does conflict, raise an exception. Override the add\_course method in the Student to enroll (add) the student to the Course and in the Instructor class to set the instructor of the Course. The remove\_course should drop a student from the Course, and unset the instructor of the course. Note that these methods are different! Also, add support for adding and dropping courses using the + and - operators.

Hints:

* To ensure that a property cannot be set, do not add a setter.
* credits can be computed via a single line (think about list comprehensions/generator expressions)
* Use super() to use the Academic add/remove\_course methods while also doing the (different) extra steps for students/instructors.
* Remember the dunder methods when supporting operators.

4. Registrar Class (15 pts)

Create a Registrar class that orchestrates the scheduling process. The registrar should keep track of all academic persons and all courses. It should have add\_persons, add\_courses, add\_person\_to\_course, and remove\_person\_from\_course methods. The add/remove person to/from course methods should take a campus id, department, course\_number, and section, and then add the person referenced by the id to the course referenced by the three other pieces of information. You should call the add/remove\_course methods, but note that polymorphism will be in effect here, meaning the method called will be different for students and instructors.

Hints:

* Because the add/remove methods take the campus\_id and course details (department, course\_number, and section), it probably makes sense to store courses and academics in dictionaries.

5. [CS 503 Only] Changing Course Times (15 pts)

We will now add logic to change course times and raises an exception if this is not possible. Add a check\_time\_conflicts **static** method to the Course class that given two time structures (list of tuples), checks if any times conflict, returning True if the two lists conflict and False otherwise. Two times conflict if they have meetings on the same day such that the starting time of one meeting is in between the starting and ending times of another meeting. For example, (“Mon”, 11, 13) and (“Mon”, 12, 14) conflict, but (“Mon”, 11, 13) and (“Mon”, 13, 15) do not.

In the Schedule class, update the add\_course method to check whether the course conflicts with the person’s current schedule before adding the course, raising an exception if it does.

In the Registrar class, add a change\_course\_time method that calls the change\_time method implemented in the Course class, and that Course method must check that the proposed new time does not cause any conflicts with any currently enrolled students’ or the assigned instructor’s schedules. If it does, the method should raise an exception. If it doesn’t, the time of the course should be changed.

Hints:

* Registrar.change\_course\_time must checked enrolled students and the instructor.

6. Summary and Testing

The final list of classes and methods/properties to be added. Note that all classes should have constructors that properly initialize objects, and the instance fields are not listed here.

* Academic:
  + add\_course
* Student:
  + add\_course
* GraduateStudent
* Instructor
  + add\_course
* Course:
  + enroll
  + drop
  + change\_time
  + [503] check\_time\_conflicts (static)
  + [support for string representation]
* Schedule:
  + credits (property)
  + add\_course
  + remove\_course
  + [support for +]
  + [support for -]
* Registrar:
  + add\_persons
  + add\_courses
  + add\_person\_to\_course
  + remove\_person\_from\_course
  + [503] change\_course\_time

The following code should help you test your work (for 490 students, the errors for conflicts will not show up, and you will not have change\_course\_time methods):

s1 = Student("z143", "Catherine", "Smith", "Senior")

s2 = Student("z352", "Niraj", "Kumar", "Sophomore")

s3 = GraduateStudent("z785", "Divya", "Bharti")

s4 = GraduateStudent("z982", "James", "O'Brien")

i1 = Instructor("a421", "Jennifer", "Martinez", "Professor")

i2 = Instructor("a572", "Jonathan", "Jones", "Instructor")

c1 = Course("CSCI", 1543, "Programming Principles in Python", 1, 3, [("Mon", 10, 12), ("Wed", 10, 12)])

c2 = Course("CSCI", 1342, "Computer Networks", 2, 4, [("Tue", 14, 16), ("Thu", 14, 16), ("Fri", 12, 13)])

c3 = Course("CSCI", 1352, "Computer Graphics", 1, 3, [("Tue", 10, 12), ("Thu", 10, 12)])

c4 = Course("SOCI", 1230, "Introduction to Sociology", 1, 3, [("Mon", 11, 13), ("Thu", 11, 13)])

c5 = Course("POLS", 1100, "American Politics", 2, 3, [("Tue", 10, 12), ("Thu", 10, 12)])

c6 = Course("SOCI", 1450, "Classical Sociological Theory", 1, 3, [("Mon", 12, 13), ("Wed", 12, 13), ("Fri", 12, 13)])

r = Registrar()

r.add\_persons([s1,s2,s3,s4,i1,i2])

r.add\_courses([c1,c2,c3,c4,c5, c6])

r.add\_person\_to\_course("a572", "SOCI", 1230, 1)

r.add\_person\_to\_course("a572", "POLS", 1100, 2) *# error due to conflict*

r.change\_course\_time("SOCI", 1230, 1, [("Mon", 9, 11), ("Wed", 9, 11)])

r.add\_person\_to\_course("a572", "SOCI", 1450, 1)

r.add\_person\_to\_course("a421", "CSCI", 1543, 1)

r.add\_person\_to\_course("a421", "CSCI", 1342, 2)

r.add\_person\_to\_course("a421", "CSCI", 1352, 1) *# error due to max credits*

r.add\_person\_to\_course("z785", "SOCI", 1230, 1)

r.add\_person\_to\_course("z785", "CSCI", 1352, 1)

r.add\_person\_to\_course("z143", "CSCI", 1543, 1)

r.add\_person\_to\_course("z143", "CSCI", 1342, 2)

r.add\_person\_to\_course("z143", "SOCI", 1230, 1) *# error due to conflict*

r.change\_course\_time( "SOCI", 1230, 1, [("Tue", 11, 13), ("Thu", 11, 13)]) *# error due to conflict*

r.change\_course\_time( "SOCI", 1230, 1, [("Tue", 12, 14), ("Thu", 12, 14)])

r.add\_person\_to\_course("z143", "SOCI", 1230, 1)

Extra Credit

* Add support for printing schedules and course enrollments