CS1210 Computer Science I: Fundamentals
Lab 6: Object-Oriented Modeling

Introduction

Recall our in-class discussion of object-oriented programming. In this assignment, you will be completing as simple object-oriented model of students and classes. Much of the code is provided in the template file; your job is to complete the methods for the two defined objects. First, some background.

Imagine the model consists of two objects, Class() and Student(). Instances of the Class() object correspond to individual courses, like CS1210. Each instance of Class() has a *name*, *sch*, an enrollment *cap*, a *roster*, and a *waitlist*. These variables associated with the object describe the state of the class, with the *roster* consisting of a dictionary of enrolled students (indexed by student), and the *waitlist* consisting of a list of students trying to get into the class (sound familiar?). Instances of the Student() class correspond to individual students. Each instance of Student() has a *name* and a *transcript*, where the former is self explanatory and the latter consists of a dictionary of the student's enrollments, indexed by class.

For this assignment, you will need to implement four methods:

Class.enroll()
Class.drop()
Student.enroll()
Student.drop()

A few examples should help make these clearer: the template file contains more information about the class and the methods you will need to write for this particular class.

Consider the following initial trace, which establishes one class, *C*, with enrollments capped at only two students, and four students, *S*1 through *S*4.

```
>>> C=Class("CS1210",cap=2)
>>> S1=Student("Joe")
>>> S2=Student("Henry")
>>> S3=Student("Mary")
>>> S4=Student("Amy")
>>> C.enroll(S1)
>>> C.enroll(S2)
>>> C.enroll(S3)
>>> C.enroll(S4)
```

The net result of the four enrollment actions should leave *S*1 and *S*2 in the class, with *S*3 and *S*4 on the waitlist:

```
>>> C
<Class: CS1210>
>>> C.roster
{<Student: Joe>: True, <Student: Henry>: True}
>>> C.waitlist
[<Student: Mary>, <Student: Amy>]
>>> S1
<Student: Joe>
>>> S1.transcript
{<Class: CS1210>: 'I'}
```

```
>>> C.drop(S1)
True
>>> C.roster
{<Student: Joe>: False, <Student: Henry>: True, <Student: Mary>: True}
>>> C.waitlist
[<Student: Amy>]
>>> S1.transcript
{<Class: CS1210>: 'W'}
>>> S3.trancript
{<Class: CS1210>: 'I'}
```

This trace reveals much of the internal structure of the two classes. The object *C* of type Class contains two main values, a roster, which is a dictionary, indexed by student, indicating the student's status in the class. The second value, waitlist, is an ordered list of student waiting to be added to the class once a slot opens up, as we see happens for Mary when Joe drops the class. Note how a student who drops remains on the roster, but their status is changed from True to False. The object *S* of type Student contains a single value, transcript, a dictionary of enrollments indexed by class, where the value of the dictionary is the student's grade, initially 'I'.

Methods

For paired assignments, you should always login as one of the two partners and work in that partner's directory (in other words, don't try to switch from one machine to the other). In every lab, you will turn in just **one** solution, with both partner's names in it. **Important: you must upload your solution prior to the end of discussion section; no late labs are accepted.**

(1) Download the Python file template provided and save it to your local machine (on the Desktop is fine).

(2) Open the file using IDLE3.

(3) The file contains a special function *hawkid*() along with function *signatures* for each of the functions in the assignment.

(4) Complete the *hawkid*() function according to the instructions given in the template file. **Important: a correct version of this function is required by the autograder to properly credit you for your work.**

(5) Complete each function by replacing the placeholder body of that function with the appropriate expression (you should alternate which partner is "driving" and which partner is "navigating" on each function or at least for every two functions).

(6) You should test each of your functions in the Python REPL.

(7) You should make sure your Python file loads and runs properly; files with syntax or indentation errors will receive no credit.

(8) Upload your Python file to ICON. Note that only the last file updated will be graded. **Important: you need upload only one file to one partner's ICON account; just be sure you include all the partner's names in the** *hawkid*() **function.**

Because we give partial credit, you should consider uploading your Python file to ICON several times (each time making sure the file loads and runs without error) over the course of the lab period to make sure all completed work gets graded.

Finally, don't forget to fill out the lab 6 survey, which is open and available to you until midnight next Monday (October 22). Failure to fill out the survey may result in less-than-full-credit on the assignment.