**COSC 1336, Lab 10 Instructions, Classes and Object-Oriented Programming**

For this lab, you will create and use a class. You will test the class by making several instances of the class. It has 3 parts. Complete ***all 3*** parts.

**Background:** A class is a group of individual data items, bundled together into one object; and a collection of methods, which can be performed on the object. Consider a clock. It “contains” hours, minutes and seconds. You can set each item, and read the time. Once available, you can advance the time forward, or set it back (“fall back” or “spring forward” when changing between daylight savings time and standard time). Or you can set the time to any value.

**Part 1:** Create a class, Clock. The Clock class has the following attributes, also called “data members”:

* hour
* minute
* second

The Clock class should have the following methods:

* \_\_init\_\_ (hr, min, sec)  
  This is the initializer or constructor. It is automatically called when a new instance of a clock is created. Example: myclock = Clock(8, 15, 30)
* setHour(hr) This is a mutator, or setter. It sets the hour of the clock.
* getHour() This is an accessor. It returns the clock’s hour.
* setMinute(min) Another mutator. It sets the minute of the clock.
* getMinute() Another accessor. It returns the clock’s minute value.
* setSecond(sec) Mutator; it sets the second of the clock.
* getSecond() Accessor; it returns the clock’s second value.
* \_\_str\_\_() This converts the class into a string for printing.  
  Example: print(myclock) outputs: hours=8, minutes=15, seconds=30

To test your class, create at least 3 instances of clock. Set each to a different time. For example: stopwatch = Clock(0, 0, 0); watch = Clock(10, 30, 0); Test each method by changing the value of the hour, minute and second of each instance of clock. Get each value with the accessors. Finally, print each instance to verify the \_\_str\_\_ method.  
Note: The methods above will also need the self parameter. See Chapter 11 for implementation details.

After you have this working, for safety, you can save this part to the file: **DDHH\_L10\_Lastname\_Clock\_p1.py**

**Part 2:** This is an enhancement to the code completed in part 1. One of the advantages of encapsulating data within a class is that it can be protected from the foibles of the outside world. That is, it can be validated and kept correct and accurate. For example, valid hours are from 0 to 23; minutes from 0 to 59; seconds from 0 to 59. These rules can and should be enforced.

In Part 2, you will enhance the class in three ways:

**Part 2a:** Currently, the mutators allow any numbers to be entered. Invalid hour, minute and second values can be entered. Enhance your class to avoid invalid values. Modify all methods that set data members. Validate the input data first. For example, clock1 = Clock(34, -100, 4345) should NOT create a new instance of clock1 with invalid data. Instead, develop a policy to adjust the invalid data to something valid. If all values are valid, accept them as provided, However, if some values are incorrect, you could convert invalid settings to 0, or to the nearest valid value.

The constructor \_\_init\_\_() and all setters: setHour, setMinute, setSecond methods should be enhanced to prevent invalid data for hour, minute and second to be allowed inside the instance of the class. Your constructor can call the setters to prevent duplicated code.

**Part 2b:** The data attributes: hour, minute and second are currently exposed to the “outside world”. That is, any code external to the Clock class can modify the data members. Example: watch.hour = 123 is allowed. This violates the concept of data hiding and allows the validation checks you just added to be bypassed. This is easy to fix. Change the names of all data members of the Clock class to begin with underscore-underscore (\_\_). You will now have: \_\_hour, \_\_minute, \_\_second. The \_\_ makes them private so they are hidden.

**Part 2c:** The \_\_str\_\_ method as described in Part 1 prints the data awkwardly: “hours=8, minutes=15, seconds=30”. Better is: **8:15:30 AM**. If the time is hours=22, minutes=45, seconds=30, print: 10:45:30 PM.  
Adjust the \_\_str\_\_method to print in the nicer format. Hours 0:0 to 11:59 should be AM. Hours 12:00 to 23:59 should be PM.

Test your new Clock class with both valid and invalid data.

**Summary: One file should be submitted for this lab:**

**DDHH\_L10\_Lastname\_Clock.py**