ACIT 2515 – Object Oriented Programming - Lab 2 (Friday Set Only)

Instructor	Mike Mulder (mmulder10@bcit.ca)
Total Marks	25
Due Dates	Parts A and B - Thursday, Jan. 24, 2019 by midnight

Goals

- Create a test plan and unit tests for an existing class, ensuring comprehensive test coverage
- Create a new class based on the specification defined in a unit test (Test Driven Development)

Part A – Unit Testing (10 Marks)

In your Pod of 4

1. Get a copy of the Course class (course.py) for Lab 2 from D2L.
2. Develop a test plan that ensures good test coverage for the Course class. For each method (including the __init__ constructor), identify:
    a. Test(s) that demonstrate 'success'
    b. Test(s) that demonstrate 'alternate', 'failure' or 'edge' cases
3. Give each of your tests a unique number (e.g., 010A, 020A, 020B) to allow traceability to method and test (i.e., 010 corresponds to a method and A corresponds to a test for that method).
4. Get your instructor to review and approve your test plan before proceeding.

In a group of 2 (i.e., pair programming)

1. Create a new project in PyCharm called Lab2.
2. Load the course.py file from D2L into your project.
3. Create a new TestCourse class (test_course.py) for the Course class using the Python built-in unittest framework. **Include both your name and your partner's name in a comment in the class.**
4. Implement your test plan as unit tests following best practices, such as:
    a. One method per test
    b. Only use public methods from the class under test in assertions
    c. DocString for the class and for each method
    d. The DocString for your methods should trace back to the test plan
5. Ensure your unit tests:
    a. Run successfully (i.e., no failures)
    b. Implement your test plan.
    c. Traces back to your test plan through the test numbers and code comments.
    d. Your unit tests should NOT access private instance variable or call private methods on the Course class.

Grading Summary	
• All run successfully (i.e., no failures) against an unmodified Course class (from D2L). • Implements your test plan. • Traces back to your test plan through the test numbers and code comments.	10 marks
Failed tests	-1 marks each
Missing tests	-1 marks each
Missing traceability (to test plan)	-1 marks each

Individual Submission

Upload the following to D2L in a **zipfile** called **lab2a.zip** (Activities -> Assignments -> Lab 2):

- Test Plan: test_course.txt (or .pdf)
- Unit Test: test_course.py
- Screen Shot: Successful test results in console

Part B – Test Driven Development (15 Marks)

This is to be done individually and will require completion outside of class.

1. Get a copy of the TestSchool class (test_school.py) and School class (school.py) for Lab 2 from D2L and load them into your Lab2 project in PyCharm:
    a. TestSchool defines the unit tests for the School class.
    b. School is an empty class.
2. Implement the School class to meet the specifications defined in the TestSchool class. *Include at least one **Constant** (Class Variable) and one **Static Method** in your implementation.*

Grading Summary	
All TestSchool tests (unmodified from D2L) run successfully against your implementation of the School class.	5 marks
School class implementation meets the intention of the specification in the TestSchool tests (i.e., you did not tailor it only to work only with the test data in TestSchool). Includes: • At least one Constant (1 mark) • At least one Static Method (1 mark)	5 marks
Use of JavaDoc (class and methods)	2 marks
Use of naming best practices (classes, variables and methods)	3 marks

Individual Submission

Upload the following to D2L (Activities -> Assignments -> Lab 2):

- school.py

Documentation Best Practices

Use the following documentation practices below for this lab.

Class Documentation	Add a comment describing what the class represents.
	
	Use DocString as per the following example:
	
	`class Point:`
	`  """Represents a point in 2D geometric coordinates"""`
	
	`  def __init__(self, x=0, x=y):`
	`  ...`

Method Documentation	Add a comment describing what the method does.
	
	`def __init__(self, x=0, x=y):`
	`  """Initialize the position of a new point. The x and y`
	`  Coordinates can be specified. If they are not, the`
	`  point defaults to the origin. """`
	
	`def move(self, x, y):`
	`  """Move the point to a new position in 2D space. """`
	`  self.x = x`
	`  self.y = y`

Docstring Reference: https://www.python.org/dev/peps/pep-0257/

Naming Best Practices

Use the following naming practices for this lab.

Class Name	CapitalizedWords (aka CamelCase)
Instance Variables	lower_case_with_underscores **Note:** Use _lower_case_with_underscores for internal (i.e., private) instance variables.
Methods	lower_case_with_underscores **Note:** Use _lower_case_with_underscores for internal (i.e., private) methods.

Reference Style Guide: https://www.python.org/dev/peps/pep-0008/