ACIT 2515 – Object Oriented Programming - Lab 4
Composition (Friday Set Only)

Instructor	Mike Mulder (mmulder10@bcit.ca) Also available on Slack.
Total Marks	25
Due Dates	Thursday, Feb. 7, 2019 by midnight

Goals

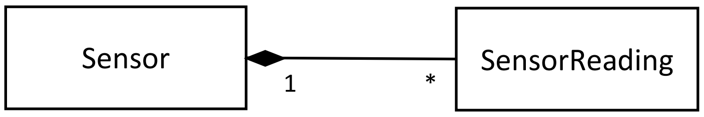
- Apply the Object Oriented Programming technique of Composition.
- Apply the Unit Testing technique of Mocking object methods.
- Continue to exercise good Python programming practices including naming conventions, documentation and unit testing.

Overview

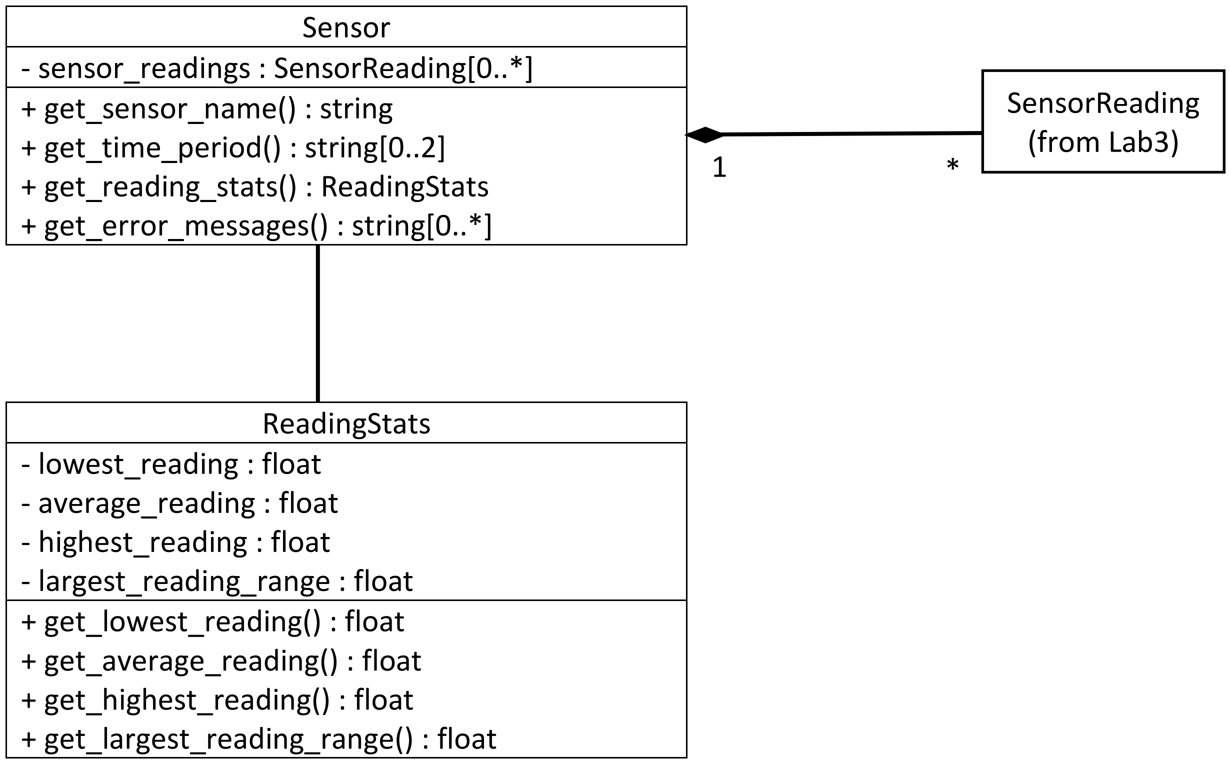
In Lab 3, you refactored the sensor_results.py Python script to use a SensorReading class to hold sensor reading data from a CSV file, where one row in the file corresponded to a single sensor reading. Today you are going to continue your OOP refactoring by defining and integrating a Sensor class.

This Lab is a continuation of the work you did in Lab 3. **You need the sensor_results.py, sensor_results.csv and sensor_reading.py files from Lab 3 as your starting point.**

You have represented your high-level design using Composition as per the following simplified UML Class diagram:



You have completed the detailed design of the Sensor class and an associated ReadingStats class in the following UML Class diagrams:


Sensor
- sensor_readings : SensorReading[0..*]
+ get_sensor_name() : string
+ get_time_period() : string[0..2]
+ get_reading_stats() : ReadingStats
+ get_error_messages() : string[0..*]
1
*
SensorReading (from Lab3)
ReadingStats
- lowest_reading : float
- average_reading : float
- highest_reading : float
- largest_reading_range : float
+ get_lowest_reading() : float
+ get_average_reading() : float
+ get_highest_reading() : float
+ get_largest_reading_range() : float

The new Sensor class will hold instances of the SensorReading class you created in Lab 3 (i.e., Composition) and will provide the following capabilities to the sensor_results.py script:

- Load the readings from the CSV file. This should be done in the constructor with the filepath of the CSV file as the parameter.
- Provide the sensor name (i.e., the sensor model)
- Provide the time period of the results (i.e., list where the first item is the start time period and the second item is the end time period)
- Return the overall minimum, average and maximum temperatures plus the largest temperature range from the readings. **It must return these values in a ReadingStats object.**
- Return any error messages associated with the readings

The sensor_results.py script must be refactored to create a Sensor object, giving it the filepath of the CSV file containing the temperature readings. It can then use the methods on the Sensor object to get the information needed to print its report.

<u>The generated report from sensor_results.py should NOT change as a result of your refactoring.</u>

Part A – Class Implementation and Test (17 marks)

Implement the Sensor (sensor.py) and ReadingStats (reading_stats.py) classes in Python that match the UML specifications above (**12 marks**). Your classes must include:

- Constants for any Magic Numbers (hint: column indices in CSV row data)
- Private Instance Variables
- Constructor/Initializer
- Public Methods
- Private Methods (only if needed)

Any parameters to your Constructor for the Sensor class only must have validation to ensure they have a valid format.

Implement a unit test for your Sensor class only (test_sensor.py) using the unittest framework (**5 marks**). Your unit test class must include:

- A success test and parameter validation test (if applicable) for each method.
- setUp and tearDown methods to create a test fixture and print out a "logPoint" message before/after each test (see lecture notes) to improve traceability.
- A mock for the csv sensor reading data. You do not want your unit tests dependent on the presence of a CSV file.

Hint:

- You will need to mock the csv.reader method such that it returns a fixed list of readings.
- You will need to create a constant list of readings as test data (use the readings from the sensor_results.csv file as the basis for this list).

```
TEST_READINGS = [
   ["2018-09-23 19:56:01.345","1","ABC Sensor Temp M301A", "20.152", "21.367", "22.005","OK"],
   ...
```

- There is a sample at the end of this Lab write-up.
- Reference on Mocking:
  - https://code.i-harness.com/en/docs/python~3.6/library/unittest.mock

Part B – Integration (8 marks)

In the sensor_results.py script, refactor the code to create and call methods on a single instance of your new Sensor class to generate its output to the console. It should only have a single main function where this is done, and the existing methods (load_data_from_csv, display_sensor_name, display_time_period, display_temperature_stats and display_error_readings) should be deprecated and removed.

Note that sensor_results.py is where the report output will be printed (using data retrieved from the Sensor object). **There should be no printing to the console in the Sensor class.**

Make sure all the outputs from the sensor_results.py script from before and after your refactoring are exactly the same.

Grading Summary

Part A – Class Implementation and Test • Sensor Class (10 marks) • ReadingStats Class (2 marks) • Unit Test Class for Sensor Only (5 marks)	17 marks
Part B – Integration • Refactoring sensor_results.py to use Sensor Class (4 marks) • No Change in Outputs from sensor_results.py (4 marks)	8 marks
Marks will be subtracted poor programming practices, including: • Violations of naming conventions • Missing or invalid DocString • Failing unit tests • Unnecessary print statements left in code Note: Not applicable to the sensor_results.py script.	-1 mark each
Total	**25 marks**

Submission

Upload the following to D2L in a zipfile (Activities -> Assignments -> Lab 4):

- The file containing your Sensor class (**sensor.py**)
- The file containing your ReadingStats class (**reading_stats.py**)
- The file containing your sensor result unit test (**test_sensor.py**)
- The file containing your SensorReading class (**sensor_reading.py**)
    - Note this will not be marked but is needed to test your code. **If it is not submitted you will not get any marks for the new unit test or the output of sensor_results.py.**
- The updated **sensor_results.py** script (note: the provided csv file and sensor_reading module should be unchanged for this lab).

Hints

- sensor_results.py should create an instance of your new Sensor class. The Sensor object should load the CSV results upon creation.
- When your refactoring is complete:
    - sensor_results.py should only have a main method that prints out the contents of the report in the required format. The contents of the report include labels and formatting plus data from the Sensor object.
    - The Sensor object should be the source of the data displayed in the report. It should not print anything itself nor should it return formatting and labels specific to the report. It is purely a source of data about the sensor readings.
- Remember that you new Sensor class is a reusable component. It can be the source of temperature sensor reports that output the data in many different formats.

Code to Mock the Reading of the csv file

```
from unittest import TestCase
from sensor import Sensor
import inspect
import csv
from unittest.mock import MagicMock
from unittest.mock import patch,mock_open


class TestSensor(TestCase):

    TEST_READINGS = [
            ["2018-09-23 19:56:01.345", "1", "ABC Sensor Temp M301A", "20.152", "21.367",
"22.005", "OK"],
            ["2018-09-23 19:57:02.321", "2", "ABC Sensor Temp M301A", "20.163", "21.435",
"22.103", "OK"],
            ["2018-09-23 19:58:01.324", "3", "ABC Sensor Temp M301A", "20.142" ,"21.528",
"21.803", "OK"],
            ["2018-09-23 19:59:03.873", "4", "ABC Sensor Temp M301A", "20.212", "21.641",
"22.017", "OK"],
            ["2018-09-23 20:00:01.453", "5", "ABC Sensor Temp M301A", "100.000", "100.000",
"100.000", "HIGH_TEMP"],
            ["2018-09-23 20:01:01.111", "6", "ABC Sensor Temp M301A", "21.244", "21.355",
"22.103", "OK"],
            ["2018-09-23 20:02:02.324", "7", "ABC Sensor Temp M301A", "21.112", "22.345",
"22.703", "OK"],
            ["2018-09-23 20:03:02.744", "8", "ABC Sensor Temp M301A", "20.513", "21.745",
"22.105", "OK"],
            ["2018-09-23 20:04:01.123", "9", "ABC Sensor Temp M301A", "20.333", "21.348",
"21.943", "OK"],
            ["2018-09-23 20:04:01.999", "10", "ABC Sensor Temp M301A", "20.332", "21.445",
"22.013", "OK"],
            ["2018-09-23 20:04:02.001", "11", "ABC Sensor Temp M301A", "-50.000", "-50.000",
"-50.000","LOW_TEMP"] ]

    # This mocks the builtin file open method in python always return '1' for the file data
    # We don’t care about the returned file data since we are mocking the csv reader as well.
    @patch('builtins.open', mock_open(read_data='1'))
    def setUp(self):

        # This mocks the csv reader to return our test readings
        csv.reader = MagicMock(return_value=TestSensor.TEST_READINGS)
        self.sensor1 = Sensor("testresults.csv")

        self.logPoint()
```

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/