ACIT 2515 – Object Oriented Programming - Lab 5 (Friday Set)
Inheritance and Polymorphism

Instructor	Mike Mulder (mmulder10@bcit.ca) Also available on Slack.
Total Marks	25
Due Dates	Thursday, Feb. 14, 2019 by midnight

You must use your Lab 4 code as your starting point for this lab.

Goals

- Apply the Object Oriented Programming Principles of Inheritance and Polymorphism (and the Template Method design pattern).
- To read and develop objects based on UML Class notation and relationships.
- Continue to exercise good Python programming practices including naming conventions, documentation and unit testing.

Overview

Your company has decided to add a line of pressure sensors to their product offering, in addition to the existing temperature sensors. Since you have refactored the sensor_results.py script to be more object oriented, you believe to can easily refactor that code again using inheritance and polymorphism to support both temperature and pressure sensors.

Your approved design is expressed below in UML format. You are now ready to refactor your code once again to conform to the design.

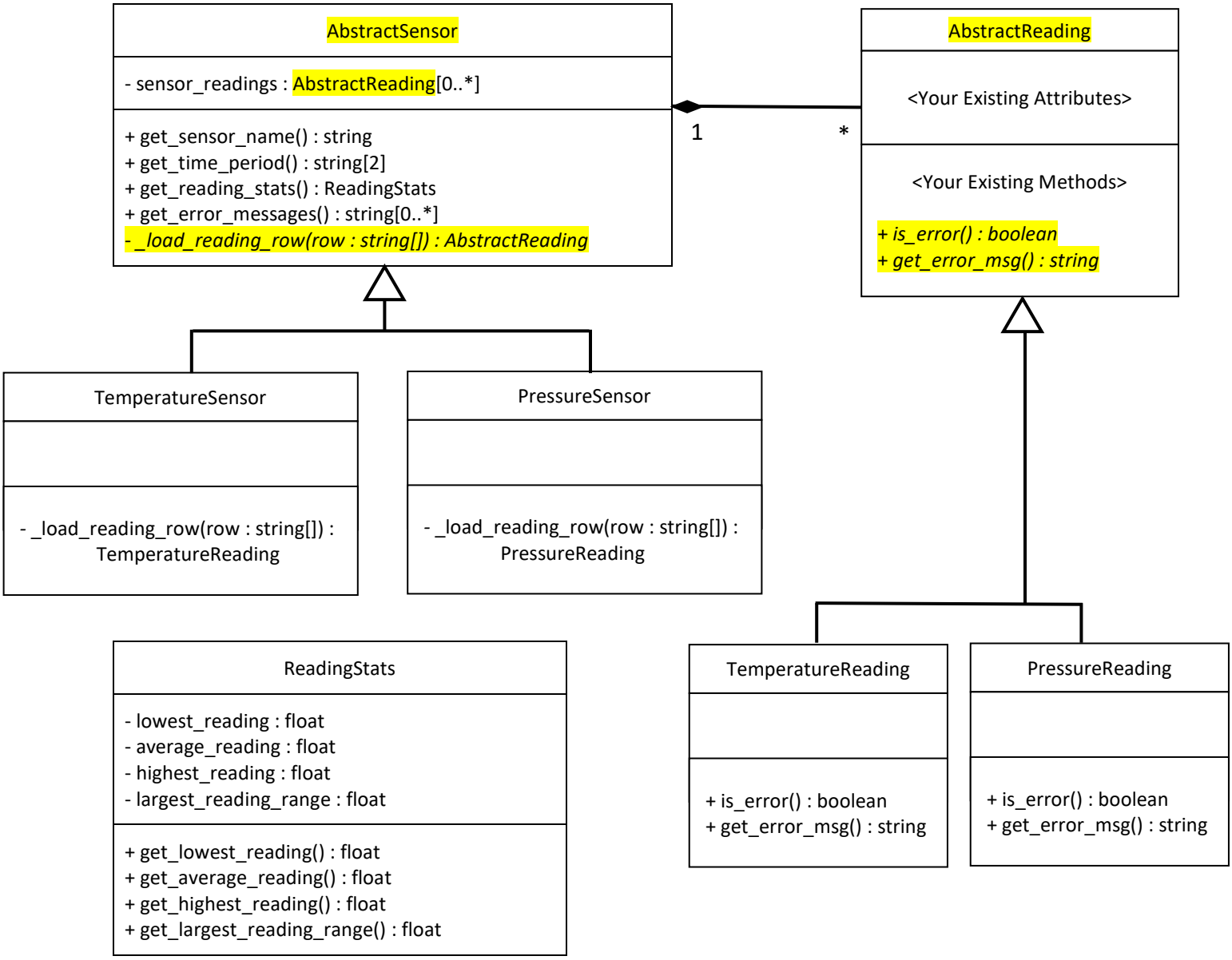
The reading data for the pressure sensor in the csv file is slightly different than that of the temperature sensor.

Temperature Sensor:
```
2018-09-23 19:59:01.873,1,ABC Sensor Temp M301A,20.212,21.641,22.017,OK
2018-09-23 20:00:02.453,2,ABC Sensor Temp M301A,100.000,100.000,100.000,HIGH_TEMP
2018-09-23 20:00:03.563,3,ABC Sensor Temp M301A,-50.000,-50.000,-50.000,LOW_TEMP
```

Pressure Sensor:
```
1,2018-09-23 19:59:01.234,ABC Sensor Pres M100,49.512,51.841,55.418,GOOD
2,2018-09-23 20:00:02.452,ABC Sensor Pres M100,100.000,100.000,100.000,HIGH_PRESSURE
3,2018-09-23 20:01:03.876,ABC Sensor Pres M100,0.000,0.000,0.000,LOW_PRESSURE
```

The sequence number is in a different location and the status values have different values for pressure sensor readings.

You have represented your design with the following UML Class diagram:



Notes:

- Methods in italics are abstract methods (which must be implemented in a child class).
- Items highlighted in yellow show key changes to the design from your previous lab(s).

Instructions

Sensor Readings

- Rename the existing SensorReading class to AbstractReading and make abstract methods for handling the error readings as the temperature and pressure sensor readings have different status values (is_error and get_error_msg methods).
- Create new TemperatureReading (temperature_reading.py) and PressureReading (pressure_reading.py) classes that are child classes of the AbstractReading parent class. They will provide concrete implementations of the is_error and get_error_msg methods.
- TemperatureSensor:
    - is_error returns False for a status of "OK" and True otherwise.
    - get_error_message returns the formatted error message for a temperature reading (i.e., High Temperature (100°C) at 2018/09/23 20:00, Sequence: 5)
    - Define and use constants for the temperature sensor status values
- PressureSensor:
    - is_error returns False for a status of "GOOD" and True otherwise.
    - get_error_message returns the formatted error message for a pressure reading (i.e., High Pressure (100 kPa) at 2018/09/23 20:00, Sequence: 5)
    - Define and use constants for the pressure sensor status values
- You do NOT need to submit any unit tests for the AbstactReading, TemperatureReading or PressureReading classes.
- ***Make sure the attribute and method names in AbstractReading are NOT specific to temperature or pressure readings. For example, a method called get_avg_temp should be renamed get_avg_reading. Likewise, a method called get_temp_range should be renamed as get_reading_range.***

Sensor

- Rename the existing Sensor class to AbstractSensor with an abstract method for loading a row representing a single reading from the csv file (_load_reading_row).
- AbstractSensor must be updated to use the is_error and get_error_msg methods from AbstractReading (if you have not already done so in the previous labs).
    - is_error – to determine if the reading is an error in AbstractSensor methods get_reading_stats and get_error_messages
    - get_error_msg – to get the error message of a reading in AbstractSensor method get_error_messages
- Create new TemperatorSensor (temperature_sensor.py) and PressureSensor (pressures_sensor.py) classes that are child classes of the AbstractSensor class. They will provide concrete implementations of the _load_reading_row method specific to each of the temperature and pressure sensor csv reading formats.
- Create unit tests for both the TemperatureSensor (test_temperature_sensor.py) and PressureSensor (test_pressure_sensor.py) classes using mocked csv data. These unit tests will be identical to your previous unit test for the Sensor class except:

    - TestTemperatureSensor will create instances of TemperatureSensor
    - TestPressureSensor will create instances of PressureSensor
    - TestPressureSensor will have TEST_READINGS that contain pressure reading data (see section Pressure Readings Test Data of lab write-up).
    - **Sample unit tests are available on D2L (under Week 5). You may use them if you wish but make sure they pass otherwise you will not receive any marks for that grading item.**
- Do NOT include a unit test for AbstractSensor.

sensor_results.py

- Refactor this script to create both a TemperatureSensor and a PressureSensor object and generate a report for each.
- Make sure to pass the "**temperature_results.csv**" filename to the constructor for TemperatureSensor and the "**pressure_results.csv**" filename to the constructor for PressureSensor when creating these objects. These csv files have been provided in the Lab 5 zipfile.
- See below for the expected output from sensor_results.py.

Expected Output:

```
Sensor: ABC Sensor Temp M301A
Period: 2018/09/23 19:56 to 2018/09/23 20:04
Lowest Temp: 20.142000°C
Average Temp: 21.57878°C
Highest Temp: 22.703000°C
Largest Temp Range: 1.940000°C
Error Messages:
  High Temperature (100°C) at 2018/09/23 20:00, Sequence: 5
  Low Temperature (-50°C) at 2018/09/23 20:04, Sequence: 11

Sensor: ABC Sensor Pres M100
Period: 2018/09/23 19:56 to 2018/09/23 20:06
Lowest Pressure: 50.142000 kPa
Average Pressure: 51.58633 kPa
Highest Pressure: 55.017000 kPa
Largest Pressure Range: 4.805000 kPa
Error Messages:
  High Pressure (100 kPa) at 2018/09/23 20:00, Sequence: 5
  Low Pressure (0 kPa) at 2018/09/23 20:06, Sequence: 11
```

Grading Summary

Sensor Readings Implementation • AbstractReading, TemperatureReading and PressureReading classes (6 marks)	6 marks
Sensor Implementation • AbstractSensor, TemperatureSensor and PressureSensor classes (9 marks)	13 marks

• Unit Tests for TemperatureSensor and Pressure Sensor classes (4 marks)	
Integration – sensor_results.py • Refactoring and adding in the new Pressure Report (4 marks) • Correct report output (2 marks)	6 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

The following files must be submitted as a **zipfile** called **lab5.zip**:

- sensor_results.py
- abstract_reading.py
- temperature_reading.py
- pressure_reading.py
- abstract_sensor.py
- temperature_sensor.py
- test_temperature_sensor.py
- pressure_sensor.py
- test_pressure_sensor.py
- reading_stats.py

Sufficient code must be submitted to run sensor_results.py otherwise no marks will be received for the lab.

Pressure Readings Test Data

You may use this as the test data for your unit tests. It matches the readings in pressure_results.csv.

```
TEST_READINGS = [
    ["1", u"2018-09-23 19:56:01.345", "ABC Sensor Pres M100", "50.152", "51.367", "52.005",
"GOOD"],
    ["2", "2018-09-23 19:57:02.321", "ABC Sensor Pres M100", "50.163", "51.435", "52.103",
"GOOD"],
    ["3", "2018-09-23 19:58:01.224", "ABC Sensor Pres M100", "50.142", "51.528", "51.803",
"GOOD"],
    ["4", "2018-09-23 19:59:03.843", "ABC Sensor Pres M100", "50.212", "51.641", "52.017",
"GOOD"],
```

```
    ["5", "2018-09-23 20:00:01.143", "ABC Sensor Pres M100", "100", "100", "100",
"HIGH_PRESSURE"],
    ["6", "2018-09-23 20:01:01.111", "ABC Sensor Pres M100", "51.244", "51.355", "52.103",
"GOOD"],
    ["7", "2018-09-23 20:02:02.324", "ABC Sensor Pres M100", "51.112", "52.345", "52.703",
"GOOD"],
    ["8", "2018-09-23 20:03:02.744", "ABC Sensor Pres M100", "50.513", "51.745", "52.105",
"GOOD"],
    ["9", "2018-09-23 20:04:01.321", "ABC Sensor Pres M100", "50.333", "51.348", "51.943",
"GOOD"],
    ["10", "2018-09-23 20:05:01.999", "ABC Sensor Pres M100", "50.332", "51.445", "52.013",
"GOOD"],
    ["11", "2018-09-23 20:06:02.022", "ABC Sensor Pres M100", "0", "0", "0", "LOW_PRESSURE"] ]
```

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/