University of Zurich[UZH]

s.e.a.l. software evolution & architecture lab

Informatics I – HS18

Exercise 8

Submission Details

- **Submission Format:** ZIP file containing your .py solution files
- **Submission Deadline:** 12:00, Tuesday 20th November
- The name of the ZIP file must have the following format: firstname_lastname_studentidentificationnumber_info1_exercise_8.zip, e.g. *hans_muster_12345678_info1_exercise_8.zip*.
- Your ZIP file should contain the following files: `task_1.py` and `task_2.py`. **Do not rename these files.**
- **Important:** Please follow the naming conventions very strictly, otherwise we will not be able to grade your submission.
- Please submit the assignment using OLAT. You may upload multiple solutions, but note that **only the last submission** will be evaluated.
- Your submissions will be tested on systems running **Python 3.7.X**. Make sure that your solutions work on this version.

1 Task: UZH Airlines (5 Points)

In this task you will model the information system of aircrafts and control tower of an airport. Let's begin with the class `Aircraft` after which all types of planes will be modeled. Each aircraft can carry a certain `number_of_passengers` and has a model `name`. Save these as protected instance variables as each plane subclass should have access to these. Add getter methods to access these variables:

- `get_number_of_passengers`
- `get_name`

To keep things simple for now, we will only add two different types of aircrafts, but more may be needed in future. Add classes `IntercontinentalAircraft` and `ShortHaulAircraft`. Each `IntercontinentalAircraft` has a certain `cargo_hold` measured in tonnes in addition to the previously defined instance variables. The cargo hold should be defined as a private instance variable.

Because of reasons, each `ShortHaulAircraft` has additionally a serial number which is unique for each plane (**hint: class variable**). Add a getter `get_serial_number`.

The on-board system on both `IntercontinentalAircraft` and `ShortHaulAircraft` can estimate the fuel consumption for a certain trip in kilometers. Add a method `calculate_amount_of_fuel(self, km)` which will compute how much fuel is necessary to complete the trip. The aircrafts however compute the amount differently:

- `IntercontinentalAircraft`: a long haul aircraft consumes 0.25 liters of fuel per km per passenger, plus 2 liters per km per tonne of cargo
- `ShortHaulAircraft` are more efficient: 0.1 liters per km per passenger

Additionally, each airplane should hold a manifest of what they are transporting for customs (define the `manifest` property which returns a string):

- `IntercontinentalAircraft`: `Intercontinental flight {name}: passenger count {passengers}, cargo load {load}`, e.g. `Intercontinental flight Boeing-747: passenger count 500, cargo load 100`
- `ShortHaulAircraft`: `Short haul flight serial number {serial number}, name {name}: passenger count {passengers}`, e.g. `Short haul flight serial number 1, name Airbus-A220: passenger count 85`

Lastly, add the control tower `ControlTower` which keeps an eye on a list of aircrafts. It should be possible to add aircrafts to observe (`add_aircraft(self, aircraft)`) and list all flights and their manifests (`get_manifests` which returns a list containing the manifest of each airplane)

Task: Implement the classes, following the instructions above.

```
1 if __name__ == '__main__':
2     intercontinental_flight = IntercontinentalAircraft(500, "Boeing-747",
      100)
3     short_haul_flight = ShortHaulAircraft(110, "Airbus-A220")
4     short_haul_flight2 = ShortHaulAircraft(85, "Airbus-A220")
5
6     assert short_haul_flight.get_serial_number() == 0
7     assert short_haul_flight2.get_serial_number() == 1
```

```
8
9     assert short_haul_flight.get_number_of_passengers() == 110
10    assert short_haul_flight.get_name() == "Airbus-A220"
11
12    assert intercontinental_flight.get_number_of_passengers() == 500
13    assert intercontinental_flight.get_name() == "Boeing-747"
14
15    assert intercontinental_flight.calculate_amount_of_fuel(10000) ==
      3250000.
16    assert short_haul_flight.calculate_amount_of_fuel(250) == 2750.
17
18    assert intercontinental_flight.manifest == "Intercontinental flight
      Boeing-747: passenger count 500, cargo load 100"
19    assert short_haul_flight2.manifest == "Short haul flight serial number
       1, name Airbus-A220: passenger count 85"
20
21    tower = ControlTower()
22    tower.add_aircraft(intercontinental_flight)
23    tower.add_aircraft(short_haul_flight)
24    tower.add_aircraft(short_haul_flight2)
25
26    air_traffic_report = tower.get_manifests()
27    for aircraft in air_traffic_report:
28        print(aircraft)
29
30    # prints:
31    # Intercontinental flight Boeing-747: passenger count 500, cargo load
      100
32    # Short haul flight serial number 0, name Airbus-A220: passenger count
       110
33    # Short haul flight serial number 1, name Airbus-A220: passenger count
       85
```

Listing 1: Example usage.

2 Task: University Management (5 Points)

To familiarize yourself further with the concept of inheritance, in this task you will implement a simple university management software that manages students (enrolled and alumni) and lecturers.

Before you start coding, also read the detailed instructions carefully. It is important that you follow the instructions precisely and name your classes, variables and methods exactly as indicated.

Instructions:

- Create a class `UniPerson`:
  - It must be initialized with a `name` argument. Store the name as a protected instance variable (for reference, see lecture slides page 62). Also create a private instance variable `__inbox` which is an empty list.
  - Define a method `receive_email(text)` which just stores (i.e., appends) the received text in the inbox.
  - Define a method `read_emails` which returns a list of all stored emails and empties the inbox. See the code for examples.
  - Overwrite the `__str__` method to return `Name: xyz`. See the code for examples.
- Create a class `Student`:
  - It must inherit from `UniPerson` and take the arguments `name`, `start_year`, `has_graduated` and `ects`. Call the constructor of the superclass. Initialize a public variable `has_graduated` and a private variable `__ects`.
  - Also in the constructor, initialize a private variable `__legi_nr`. It must be a string with the following format: `<start_year>-<five digit number>`. The five digit number must be increased for each new student with the same `start_year`, and start from zero for each student with a new `start_year`. To create a five digit number with zero-padding, check out Python's string method `zfill`[1]. Some examples: The 1st student of 2018 will have 2018-00000, the 5th student will have 2018-00004, the 555th will have 2018-00554, etc. You can assume that your university is rather small, so don't worry about running out of available five-digits numbers for any given year.
    Hint: You may want to use a class variable that helps you count how many students are created with the same `start_year`.
  - Override the `__str__` method to return a more accurate description, including the superclass's `__str__` output, the legi number, whether the student has graduated (True or False) and the number of ECTS. You are free to choose a readable format yourself, it just needs to include the above-mentioned information.
- Create a class `Lecturer`:
  - It must inherit from `UniPerson` and take the arguments `name` and `lecture_name`. Call the constructor of the superclass and store the lecture name in a private instance variable `__lecture_name`.
  - Override the `__str__` method in a similar way as in the `Student` class, so that it includes the superclass's `__str__` output and the lecture name.
- Finally, create a class `UniManagement`:
  - It must be initialized with no arguments, and an empty list `__persons` must be created upon initialization.

[1] https://docs.python.org/3/library/stdtypes.html?highlight=zfill#str.zfill

- Define a method `add_person(person)` that checks whether the argument is an instance of `UniPerson` and if yes, adds it to the persons list.
- Define a method `list_persons` that returns a list of string descriptions (`__str__`) of all persons in the list.
- Define a method `send_email(text)` that sends an email to every person in the list.
- Define a method `count_alumni` that returns the number of students in the list that have already graduated.

Check the following snippets to see how to use the classes and expected outputs.

```
if __name__ == '__main__':
    p1 = UniPerson("Hans Muster")
    assert p1.__str__() == "Name: Hans Muster"

    p1.receive_email("Email 1")
    p1.receive_email("Email 2")
    assert p1.read_emails() == ["Email 1", "Email 2"]
    assert p1.read_emails() == []  # Because inbox was emptied after
    reading the first time

    s1 = Student("Student 1", 2017, False, 40)
    assert "Student 1" in s1.__str__()
    assert "2017-00000" in s1.__str__()
    assert "False" in s1.__str__()
    assert "40" in s1.__str__()

    s2 = Student("Student 2", 2017, True, 120)
    assert "Student 2" in s2.__str__()
    assert "2017-00001" in s2.__str__()
    assert "True" in s2.__str__()
    assert "120" in s2.__str__()

    s3 = Student("Student 3", 2016, True, 180)
    assert "Student 3" in s3.__str__()
    assert "2016-00000" in s3.__str__()
    assert "True" in s3.__str__()
    assert "180" in s3.__str__()

    mgmt = UniManagement()

    lecturer = Lecturer("Prof. Dr. Harald Gall", "Informatik 1")

    mgmt.add_person(s1)
    mgmt.add_person(s2)
    mgmt.add_person(s3)
    mgmt.add_person(lecturer)

    assert mgmt.count_alumni() == 2

```

```
39      mgmt.send_email("This test email is sent to all university persons.")
40      assert s1.read_emails() == ["This test email is sent to all university
         persons."]
41      assert s2.read_emails() == ["This test email is sent to all university
         persons."]
42      assert s3.read_emails() == ["This test email is sent to all university
         persons."]
43      assert lecturer.read_emails() == ["This test email is sent to all
        university persons."]
```

Listing 2: Example usage.

Task: Create a file `task_2.py` and implement the classes, following the instructions above.