University of St. Gallen

Institute of Computer Science

Fundamentals of Computer Science (3,125), HS2019
R. Seiger, I. Mizutani, S. Mayer, B. Weber
ronny.seiger@unisg.ch

Assignment 2: Object-Oriented Programming

Deadline: Oct 1, 2019; 12:00 CET

Introduction After consolidating your basic procedural programming skills in Python in the previous exercises, we now take a look at creating larger programs that are more than just a script with some functions. The object-oriented programming (OOP) paradigm relies on creating an abstract view of the problem you want to solve with your program based on self-defined data types (classes) and instances of these classes that interact with each other.

Rationale In this exercise you will learn to create custom data types known as *Classes* and instances of these classes known as *Objects*. We will define relations between the classes and let the objects interact. A simple extension of the examples will illustrate the concept of specializing classes known as *Inheritance*.

Prerequisites Basic understanding of OOP and the provided template files.

(1) Your first Object-oriented program (5 Points)

Here we create a simple student and course management system using OOP concepts. Complement the missing parts of the classes including attributes, methods and constructors for initialization in individual files in one Python package called `oop_a2_t1`. `client_t1.py` is provided as a sample client script for testing the package. After completing all the tasks, the output of `client_t1.py` would look like Figure 1.

- Task 1.1: Complete the file `oop_a2_t1/student.py` and define a class `Student` having a name (`name`) and a student number (`number`) as instance attributes.
- Task 1.2: Complete the file `oop_a2_t1/department.py` and define a class `Department` having a name (`name`), a department code (`code`), a list of courses (`courses`) and a class instance method `add_course` to add a new course to the `courses` list. The `add_course()` method should take a `Course` class (see Task 1.3) instance as the argument.
- Task 1.3: Complete the file `oop_a2_t1/course.py` and define a class `Course` having a name (`name`), a course code (`code`), a number of credit points (`credit`), and a maximum number of students (`student_limit`). The `Course` class should also have a list of students (`students`), a class instance method to enroll a new student to the course (`add_student()`), and another method to remove a student from the course by passing the student number (`remove_student_by_number()`). These two methods should raise `Exception` [1] to let client programs know errors occurred during the run-time. Use two Exceptions already defined in the file; `add_student()` should raise `CourseCapacityFullError` when the course already has the maximum number of students enrolled, and `remove_student_by_number()` must raise `StudentNotEnrolledError` when there's no student with the given number enrolled in the course. Finally, complete `to_string()` method, which returns a string representation of the instance attributes with the number of students enrolled, their names, and numbers. Refer to the sample output as shown in Figure 1.

```
ERROR: Student Alice not enrolled in Mathematics 1000
ERROR: Mathematics 1000 is already full
==Couse information==
Name: Mathematics 1000
Code: MAM1000W
Maximum students: 10
Number of Students enrolled: 10
Enrolled students:
- Bob <1337>
- Student_0 <1338>
- Student_1 <1339>
- Student_2 <1340>
- Student_3 <1341>
- Student_4 <1342>
- Student_5 <1343>
- Student_6 <1344>
- Student_7 <1345>
- Student_8 <1346>
```

Figure 1: An output from client_t1.py

(2) Classes with Inheritance (5 Points)

We will now create some new generalized classes that provide some common attributes and methods shared among all types of objects as well as some specialized classes that will inherit the common attributes. Work on the files (modules) in `oop_a2_t2` package to complete the tasks.

- Task 2.1: Complete the file `oop_a2_t2/person.py` and define a class `Person` that has a `name`, `surname`, and `number`.
- Task 2.2: Complete the file `oop_a2_t2/student.py` and define a new `Student(Person)` class that inherits the name, surname and number from the `Person` class (call `super()` method). In addition, this new `Student` class should have a list `enrolled_courses` to store the courses the student has enrolled, and a method `enroll_in()` to add a `Course` class instance (from Task 1.3) to the list. The `enroll_in` method should also raise `StudentAlreadyEnrolledError` when the student was already enrolled in the given course. Finally, define another instance attribute `academic_status` to store one of the two "constant" class attributes `UNDERGRADUATE` or `POSTGRADUATE`. Define those two constant class attributes, and make the constructor to take it as the first argument to indicate the academic status – the constructor signature should look like `Student(academic_status, name, surname, number)`.
- Task 2.3: Complete the file `oop_a2_t2/staff_member.py` and define a class `StaffMember` that inherits all the attributes from `Person` class. Define class attributes again to distinguish between a staff member being `PERMANENT` or `TEMPORARY`.
- Task 2.4: Complete the file `oop_a2_t2/lecturer.py` and a class `Lecturer` that inherits all the attributes from `StaffMember`. A lecturer should also have a list `assigned_courses` and a method `assign_to` to add a new course to this list.
- Task 2.5: Complete the file `client_t2.py` to test your new application which does the following. If necessary, implement methods in the classes.

  1.) Create a new `oop_a2_t1.course.Course` object.

[1] https://docs.python.org/3/tutorial/errors.html#exceptions

2.) Create a new postgraduate student as an `oop_a2_t2.student.Student` object, enroll the student in the new course, and print the name, the academic status of student, the courses enrolled, and the type of the student object using the `type()` method. Also check `StudentAlreadyEnrolledEroor` is properly raised.

3.) Check if the student object is an instance of the `Person` class using the `isinstance()` method.

4.) Create a new object of `Lecturer` who has a permanent position, assign the lecturer to the course, and then print the lecturer's name, employment type, and assigned course.

5.) Check if the lecturer is an instance of the class `Person`, and if the lecturer is an instance of class `StaffMember`.

(3) **Hand-in Instructions** Create a `zip` file including all the Python files and additional files described in the individual tasks. Make sure the Python files contain your ID and names as with the first assignment. Add a file `README.txt` to the zip file with your explanations, additional answers and comments. Upload the zip file to Canvas. See Figure. 2 for the list of files should be included in the zip file.

```
assignment2.zip
├── client_t2.py
├── oop_a2_t1
│   ├── __init__.py
│   ├── course.py
│   ├── department.py
│   └── student.py
├── oop_a2_t2
│   ├── __init__.py
│   ├── lecturer.py
│   ├── person.py
│   ├── staff_member.py
│   └── student.py
└── README.txt
```

Figure 2: The required files for submission