Michael Kemery

2024.08.

Foundations of Programming, Python

Assignment_07

branchingTacos/IntroToProg-Python-Mod07 (github.com)

Assignment 07 – Classes and Objects

Introduction

This week, continued focusing on enhancing the course registration program without adding significant functionality. We particularly focused on adding new classes and objects to the program – this is to introduce us to concepts related to object-oriented programming within the context of Python. By introducing the `Person` and `Student` classes, we aimed to create a more organized codebase that not only simplifies data handling but also adheres to best practices in software development. This approach enables a more modular design, making the program easier to maintain and extend in future iterations.

Main Program

Similar to the last several weeks, I began this week's assignment by reviewing the starter code, the requirements document, and identifying where the new data classes and error handling could be implemented.

Once I had an idea of how the code was to be structured, I started to outline `Person` and `Student` classes. Initially this was done by hand via sketching the class and methods within each class. This included detailing how we are handling the first and last name data and using the getters and setters.

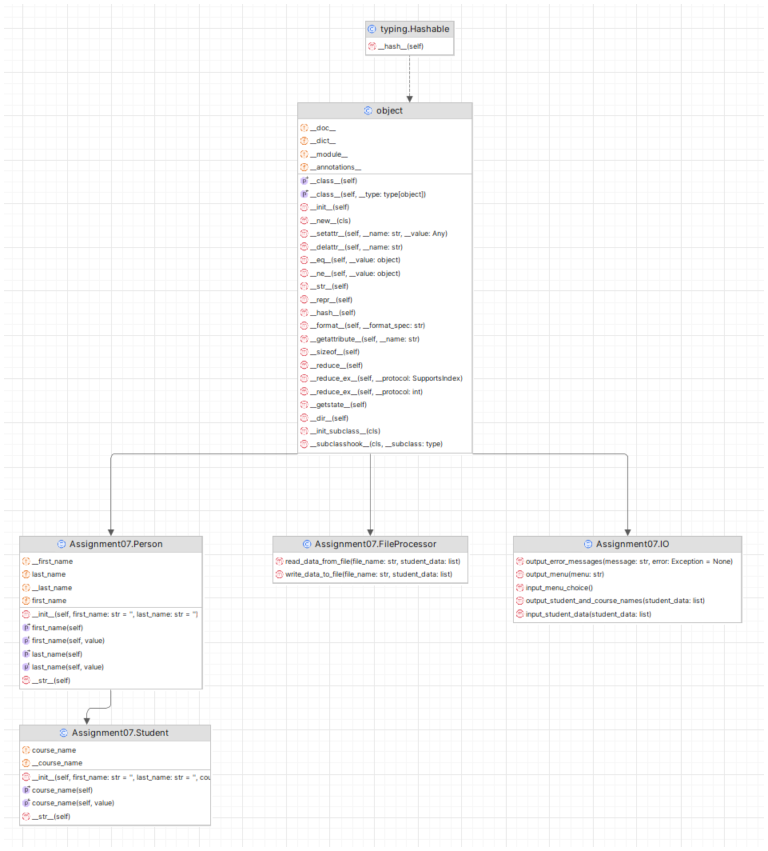
Because a student is also a person, we wanted to ensure that the student class those methods that were relevant (e.g., first and last name). The `course_name` variable is not part of the Person class, it is instantiated in the Student class. This is because a student is a type of Person. Therefore, if we wanted to reuse the Person class in a different context, we could with minimal changes to the class itself.

Once I had the Person and Student classes defined, I then went through the other classes and updated where appropriate. i.e., changing some of the pointers from a general dictionary holding student information to using the student class and methods to manage the data (see Code Snippet 1).

```
def input_student_data(student_data: list):
        try:
            student_first_name = input("Enter the student's first name: ")

            student_last_name = input("Enter the student's last name: ")

            course_name = input("Please enter the name of the course: ")

            ## change from dictionary to using student object
            student = Student(first_name=student_first_name, \

                     last_name=student_last_name, course_name=course_name)

             student_data.append(student)

      #            student = {"FirstName": student_first_name,
      #                            "LastName": student_last_name,
      #                            "CourseName": course_name}
      #            student_data.append(student)
```

Just to ensure that I was thinking about the organization properly I looked at PyCharm's auto-generated UML diagram (see Illustration 1). This helped me verify that the `Student` class was set up to inherit correctly from the `Person` class. It also allowed me see the relationship between the other classes as well as the methods within each one. Once the classes were properly integrated and program worked as expected I then went back to examine the error handling.

Illustration 1:


typing.Hashable
__hash__(self)
object
__doc__
__dict__
__module__
__annotations__
__class__(self)
__class__(self, __type: type[object])
__init__(self)
__new__(cls)
__setattr__(self, __name: str, __value: Any)
__delattr__(self, __name: str)
__eq__(self, __value: object)
__ne__(self, __value: object)
__str__(self)
__repr__(self)
__hash__(self)
__format__(self, __format_spec: str)
__getattribute__(self, __name: str)
__sizeof__(self)
__reduce__(self)
__reduce_ex__(self, __protocol: SupportsIndex)
__reduce_ex__(self, __protocol: int)
__getstate__(self)
__dir__(self)
__init_subclass__(cls)
__subclasshook__(cls, __subclass: type)
Assignment07.Person
__first_name
last_name
__last_name
first_name
__init__(self, first_name: str = '', last_name: str = '')
first_name(self)
first_name(self, value)
last_name(self)
last_name(self, value)
__str__(self)
Assignment07.FileProcessor
read_data_from_file(file_name: str, student_data: list)
write_data_to_file(file_name: str, student_data: list)
Assignment07.IO
output_error_messages(message: str, error: Exception = None)
output_menu(menu: str)
input_menu_choice()
output_student_and_course_names(student_data: list)
input_student_data(student_data: list)
Assignment07.Student
course_name
__course_name
__init__(self, first_name: str = '', last_name: str = '', cou
course_name(self)
course_name(self, value)
__str__(self)

Challenges / Modifications to the program:

This week I opted not to explore modifying the program because this is the area that while I conceptually and logically understand why we should use classes and object – I have always had difficultly implementing them appropriately.

Summary

This week's focus was on enhancing the course registration program by introducing new classes and objects. This effort aimed to reinforce object-oriented programming concepts in Python. By adding the Person and Student classes, the codebase was made more organized, facilitating simplified data handling and adhering to best practices in software development. This modular approach not only streamlines maintenance but also lays the groundwork for future extensions of the program and helps build the knowledge and skills required to create more complex programs.