Jeisson Guerrero Estrada

02/28/2024

Python 100

Assignment 07

Github: Click here for an external link!

Object-Oriented Programming

Introduction

OOP is a programming paradigm that uses classes and objects to organize code. In Python, I implemented these principles in a Course Registration Program using data classes and structured error handling. The Student class inherits from the Person class, which encapsulates properties for the first and last names. The Student class adds a property for the course name, with constructors, getters, setters, and string representation methods to ensure proper data manipulation.

Constructors

Constructors initialize object instances. I created constructors for Person and Student classes. Person constructor formats first and last names, and Student constructor uses

super() to call Person constructor and initialize course name.Constructors are essential in initializing object instances. In my implementation, I have created constructors for both the Person and Student classes. The Person constructor takes parameters for the first name and last name, ensuring that the names are properly formatted and do not contain any numbers. On the other hand, the Student class constructor uses the super() function to invoke the parent class (Person) constructor and initializes the course name property. See (Figure 1) for reference.

```
# TODO Create a Person Class
class Person:
    """
    A class to represent a Person

    Properties:
    name (str): The name of the person


    Change Log:
    - Jeisson Guerrero Estrada, 2.28.24: Created a class Person.
    """

    # TODO Add first_name and last_name properties to the constructor (Done)

    def __init__(self, first_name: str = "", last_name: str = ""):
        self.__first_name = first_name
        self.__last_name = last_name

    # TODO Create a getter and setter for the first_name property (Done)
```

```
    @property
    def first_name(self):
        return self.__first_name.title()

    @first_name.setter
    def first_name(self, value: str):
        if value.isalpha() or value == "":
            self.__first_name = value
        else:
            raise ValueError("The First Name should not contain any
numbers.")

    # TODO Create a getter and setter for the last_name property (Done)

    @property
    def last_name(self):
        return self.__last_name.title()

    @last_name.setter
    def last_name(self, value: str):
        if value.isalpha() or value == "":
            self.__last_name = value
        else:
            raise ValueError("The Last name should not contain any numbers.")

    # TODO Override the __str__() method to return Person data (Done)

    def __str__(self):
        return f"{self.__first_name} {self.__last_name}"
```

FIGURE 1

Properties

In Python, properties are used to provide controlled access to class attributes. In my code, I have implemented properties for the first name, last name, and course name. These properties have both getters and setters, and they incorporate simple validation logic. For example, the setter for the first and last names ensures that the input contains only alphabetical characters. If the input is otherwise, a ValueError is raised. This approach guarantees data integrity and consistency. See (Figure 2) for reference.

```
# TODO Create a Student class the inherits from the Person class (Done)

class Student(Person):
    """
    A class to represent a Student

    Properties:
    name (str): The name of the student
    Course (str): The course name

    Change Log: - Jeisson Guerrero Estrada, 2.28.24: Created a class Student.
    """

    # TODO call to the Person constructor and pass it the first_name and
last_name data (Done)
```

```
    def __init__(self, first_name: str = "", last_name: str = "",
course_name: str = ""):
        super().__init__(first_name=first_name,last_name=last_name)

        # TODO add a assignment to the course_name property using the
course_name parameter (Done)
        self.__course_name = course_name

    # TODO add the getter for course_name (Done)

    @property
    def course_name(self):
        return self.__course_name

    # TODO add the setter for course_name (Done)

    @course_name.setter
    def course_name(self, value: str):
        if value.isalpha() or value == "":
            self.__course_name = value
        else:
            raise ValueError("The Course Name should not contain any
numbers.")


    # TODO Override the __str__() method to return the Student data (Done)

    def __str__(self):
        return f"{super().__str__()}, {self.course_name},"  # using super()
to get parent class string representation
```

FIGURE 2

Inheritance

Inheritance is a fundamental concept in Object-Oriented Programming (OOP) that helps to reuse code. The Student class, which is a type of Person, inherits from the Person class. This demonstrates an "is-a" relationship between the two classes. By doing this, I was able to use the attributes and methods of the Person class in the Student class, which promotes code modularity and reduces redundancy. Such a hierarchical structure enhances the readability and maintainability of the code. See (Figure 2) for reference.

FIGURE 2

Summary

To summarize, this project gave me a practical understanding of how to apply OOP principles in programming. Implementing constructors, properties, and inheritance in the Course Registration Program demonstrated the flexibility and potential of OOP in Python. Through this project, I not only reinforced my theoretical knowledge but also learned the importance of designing and organizing software for better maintainability and robustness.