Nicholas Plaza

08/13/2024

IT FDN 110 A Su 24: Foundations Of Programming: Python

Module 7

https://github.com/nplaza80/IntroToProg-Python-Mod07.git

Classes and Functions 2

Introduction

This Python script demonstrates the use of data classes, specifically in the context of managing student course registrations, with a strong emphasis on structured error handling. The program is organized into distinct classes—Person, Student, FileProcessor, and IO—each fulfilling a specific role within the application. The Person class handles basic personal information such as first and last names, ensuring data integrity through property setters that validate the input. The Student class inherits from Person and adds functionality to manage the course name, further demonstrating the object-oriented programming principles of inheritance and encapsulation. (Figure1)

Figure 1: Displaying four core Classes of program

Classes and Functions:

FileProcessor

The script also includes the FileProcessor class, which is responsible for reading and writing student data to and from a JSON file. This class uses structured error handling to manage potential issues that may arise during file operations, such as file not found errors or issues with file content. The read_data_from_file method reads the file and converts the data into Student objects, while the write_data_to_file method serializes the student data back into JSON format for storage. This separation of concerns ensures that the data processing logic is isolated from other parts of the program, making the code more maintainable and easier to debug. (Figure2)

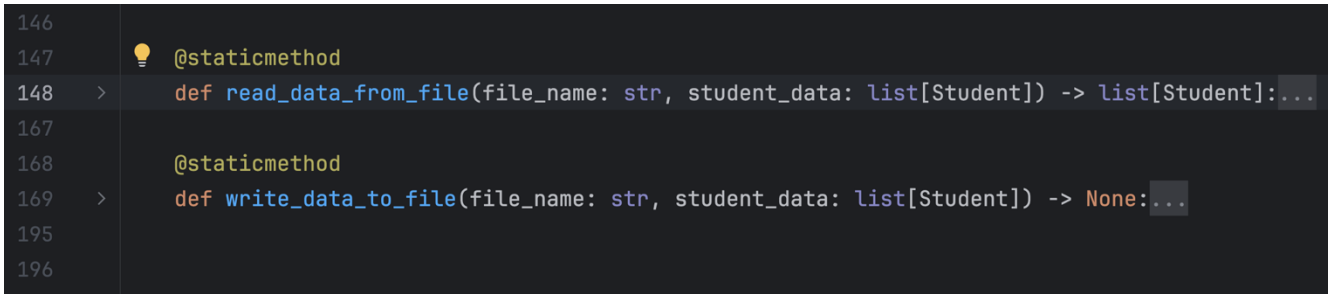

Figure 2: File processor with Read/Write Functions

IO: Input and Output

The IO class manages all user interactions, including displaying menus, capturing user input, and presenting data to the user. This class also incorporates error handling to manage invalid user inputs, ensuring that the program remains robust even when users make mistakes. For example, when the user inputs a menu choice, the input_menu_choice method checks for valid options and provides feedback if an invalid choice is made. Similarly, the input_student_data method ensures that the user-provided names and course names are correctly formatted and free from errors before adding them to the system. (Figure 3-4)

```python
    @staticmethod
    def output_error_messages(message: str, error: Exception = None) -> None:
        """ This function displays a custom error messages to the user

        ChangeLog: (Who, When, What)
        NPlaza, August 11, 2024, Created
        :param message: string with message data to display
        :param error: Exception object with technical message to display
        :return: None
        """
        print(message, end="\n\n")
        if error is not None:
            print("-- Technical Error Message -- ")
            print(error, error.__doc__, type(error), sep='\n')

    @staticmethod
    def output_menu(menu: str) -> None:
        """ This function displays the menu of choices to the user

        ChangeLog: (Who, When, What)
        NPlaza, August 11, 2024, Created
        :return: None
        """
        print(menu)
        print()  # Adding extra space to make it look nicer.

    2 usages
    @staticmethod
    def output_student_and_course_names(student_data: list[Student]) -> None:
        """ This function displays the student and course names to the user

        ChangeLog: (Who, When, What)
        NPlaza, August 11, 2024, Created

        :param student_data: list of dictionary rows to be displayed

        :return: None
        """

        print("-" * 50)
        for student in student_data:
            print(f'Student {student.first_name} {student.last_name} is enrolled in {student.course_name}')
        print("-" * 50)
```

Figure 3: IO Output functions

```python
    @staticmethod
    def input_menu_choice() -> str:
        """ This function gets a menu choice from the user

        ChangeLog: (Who, When, What)
        NPlaza, August 11, 2024, Created

        :return: string with the users choice
        """
        choice = "0"
        try:
            choice = input("Enter your menu choice number: ")
            if choice not in ("1", "2", "3", "4"):  # Note these are strings
                raise Exception("Please, choose only 1, 2, 3, or 4")
        except Exception as e:
            IO.output_error_messages(e.__str__())  # Not passing e to avoid the technical message
        return choice


    @staticmethod
    def input_student_data(student_data: list[Student]) -> list[Student]:
        """ This function gets the student's first name and last name, with a course name from the user

        ChangeLog: (Who, When, What)
        NPlaza, August 11, 2024, Created

        :param student_data: list of dictionary rows to be filled with input data

        :return: 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: ")
            student = Student(student_first_name, student_last_name, course_name)
            student_data.append(student)
            print()
            print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
        except ValueError as e:
            IO.output_error_messages(message="One of the values was not the correct type of data!", error=e)
        except Exception as e:
            IO.output_error_messages(message="Error: There was a problem with your entered data.", error=e)
        return student_data
```

Figure 4: IO Input functions

The main Body

The main body of the script ties everything together, starting by loading any existing student data from the file into the program. It then enters a loop where it continuously presents the menu to the user, processes their choices, and performs actions such as registering a new student, displaying current registrations, and saving data to the file. The loop continues until the user chooses to exit the program, ensuring that all actions are completed before the program ends. The use of classes and functions to separate the different responsibilities within the code is a clear example of the separation of concerns principle, which enhances the script's readability, maintainability, and scalability. (Figure 5)

```
# Main Body of Script ------------------------------------------------------ #
students: list[Student] = []  # a table of student data
menu_choice: str = ''  # Hold the choice made by the user.

# When the program starts, read the file data into a list of lists (table)
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)

# Present and Process the data
while True:
    IO.output_menu(menu=MENU)
    menu_choice = IO.input_menu_choice()

    if menu_choice == "1":  # Input user data
        students = IO.input_student_data(student_data=students)
        continue
    elif menu_choice == "2":  # Present the current data
        IO.output_student_and_course_names(student_data=students)
        continue
    elif menu_choice == "3":  # Save the data to a file
        FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
        continue
    elif menu_choice == "4":  # Stop the loop
        break
    else:
        print("Please only choose option 1, 2, 3, or 4")

print("Program Ended")
```

Figure 5: Main body is significantly reduced by building reusable functions into classes.

Summary

This Python script is a well-structured program designed to efficiently manage student course registrations by leveraging data classes and comprehensive error handling. It organizes its functionality into distinct classes: Person for handling personal details like first and last names, Student for managing additional course-related information, FileProcessor for reading and writing student data to and from a JSON file, and IO for managing all user interactions. This approach ensures a clear separation of concerns, which enhances the program's maintainability and readability. The script includes robust validation of user inputs, processes file operations with careful error handling, and provides clear feedback to users, making the program both reliable and user-friendly. The main script ties these components together, allowing for seamless student registration, data display, and data storage, ensuring that all processes are completed efficiently before the program exits.