Phuong Tran (Eric)

November 23, 2024

IT FDN 110 A

Assignment 06

GitHubURL: https://github.com/erictran03/IntroToProg-Python-Mod06

Working with Functions, Classes, and Structured Error Handling

Introduction

In Assignment 06, I built upon the foundations established in Assignment 05, which demonstrated the use of dictionaries, JSON files, and exception handling. The focus of Assignment 06 was on enhancing the modularity and organization of the program using functions, classes, and the separation of concerns programming pattern. These improvements resulted in cleaner, more maintainable code. This document explains the steps taken to complete the assignment, the knowledge gained, and how the changes improved the overall program functionality.

New Static Methods and Classes

One of the biggest improvements in this version is the use of static methods inside two classes: FileProcessor and IO. Static methods don't rely on instances of a class, which makes them great for organizing related functionality:

The FileProcessor Class

This class handles file operations like reading and writing data to the JSON file. The methods read_data_from_file (Figure 1) and write_data_to_file (Figure 2) ensure that the program can safely interact with the file system while handling potential errors.

```
def read_data_from_file(file_name: str, student_data: list):  1 usage
    """ This function reads data from a json file and loads into a list of dictionary rows
        ChangeLog: (Who, When, What)
        Phuong Tran,11.23.2024,Created function
        :return: list
        """
    try:
        file = open(file_name, "r")
        student_data = json.load(file)
        file.close()
    except Exception as e:
        IO.output_error_messages(message="Error: There was a problem with reading the file.", error=e)
    finally:
        if file.closed == False:
            file.close()
        return student_data
```

Figure 1: Read data from a Json file and loads into a list of dictionary rows

```python
def write_data_to_file(file_name: str, student_data:list):  1 usage
    """ This function write data to json file with data from a list of dictionary rows
            ChangeLog: (Who, When, What)
            Phuong Tran,11.23.2024,Created function
           :return: none
           """
    try:
        file = open(file_name, "w")
        json.dump(student_data, file)
        file.close()
        IO.output_student_courses(student_data=student_data)
    except Exception as e:
        message = "Error: There was a problem with writing to the file.\n"
        message += "Please check that the file is not open by another program."
        IO.output_error_messages(message=message,error=e)
    finally:
        if file.closed == False:
            file.close()
```

Figure 2: Write data to Json file with data from a list of dictionary rows

Using static methods helped me separate file operations from the rest of the program. This way, the main program doesn't need to worry about file handling logic—it just calls the method.

The IO Class

The IO class is a collection of static methods that handle all user interactions in the program. This includes displaying menus, collecting input, and printing error or feedback messages. By using static methods, the program ensures that these functionalities are centralized, reusable, and easy to maintain.

1. **output_error_message** (Figure 3)
   This method displays custom error messages to the user. It can also show technical details about the error, which is helpful for debugging. The method prints a user-friendly error message. If an exception is passed to the method, it also displays details about the exception, such as its type and a description of the error.

```python
@staticmethod  5 usages
def output_error_messages(message: str, error: Exception = None):
    """ This function displays the custom error messages to the user
    ChangeLog: (Who, When, What)
    Phuong Tran,11.23.2024,Created function
    :return: None
    """
    print(message, end="\n\n")
    if error is not None:
        print("-- Technical Error Message -- ")
        print(error, error.__doc__, type(error), sep='\n')
```

Figure 3: Display the custom error messages to the user

2. **output_menu** (Figure 4)

This method displays the program's main menu in a clean and readable format. It takes the MENU constant (a pre-formatted string) as input and prints it with extra spacing for better readability.

```
@staticmethod  1 usage
def output_menu(menu: str):
    """ This function displays the menu of choices to the user
    ChangeLog: (Who, When, What)
    Phuong Tran,11.23.2024,Created function
    :return: None
    """
    print()  # Adding extra space to make it look nicer.
    print(menu)
    print()  # Adding extra space to make it look nicer.
```

Figure 4: Display the menu of choices to the user

3. **input_menu_choice** (Figure 5)

This method gets the user's menu choice and validates it to ensure that only valid options (1, 2, 3, or 4) are accepted.

```
@staticmethod  1 usage
def input_menu_choice():
    """ This function gets a menu choice from the user
    :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
```

Figure 5: Get menu choice from the user

4. **output_student_courses** (Figure 6)

This method displays the list of students and their registered courses in a neat, formatted output.

```
@staticmethod  2 usages
def output_student_courses(student_data: list):
    """ This function displays the student and course names to the user
    ChangeLog: (Who, When, What)
    Phuong Tran,11.23.2024,Created function
    :return: None
    """
    # Process the data to create and display a custom message
    print("-" * 50)
    for student in student_data:
        print(f'Student {student["FirstName"]} '
              f'{student["LastName"]} is enrolled in {student["CourseName"]}')
    print("-" * 50)
```

Figure 6: Display the student and course names to the user

5. **input_student_data** (Figure 7)

This method collects and validates a student's first name, last name, and course name. Once validated, it adds the data to the list of students.

```python
@staticmethod  1 usage
def input_student_data(student_data: list):
    """ This function gets the student's first and last name, with a course name from the user
    ChangeLog: (Who, When, What)
    Phuong Tran,11.23.2024,Created function
    :return: list
    """
    try:
        student_first_name = input("Enter the student's first name: ")
        if not student_first_name.isalpha():
            raise ValueError("The last name should not contain numbers.")
        student_last_name = input("Enter the student's last name: ")
        if not student_last_name.isalpha():
            raise ValueError("The last name should not contain numbers.")
        course_name = input("Please enter the name of the course: ")
        student = {"FirstName": student_first_name,
                   "LastName": student_last_name,
                   "CourseName": 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 7: Get the student's first and last name with course name from the user

Handling Menu Options with Static Methods

In Assignment 06, I organized the menu options so each one calls the appropriate method from the IO or FileProcessor class. This made the code cleaner and reduced the number of repetitive if-else statements in the main program.

1. **Register a Student (Option 1):**

   This option calls the IO.input_student_data method to collect and validate the student's name and course information. Once the input is validated, the data is added to the students list (Figure 8).

```python
# Input user data
if menu_choice == "1":  # This will not work if it is an integer!
    students = IO.input_student_data(student_data=students)
    continue
```

Figure 8: Input user data - Option 1

2. **Show Current Data (Option 2):**

   This option calls the IO.output_student_courses method to display the list of registered students in a formatted output (Figure 9).

```python
# Present the current data
elif menu_choice == "2":
    IO.output_student_courses(students)
    continue
```

Figure 9: Present the current data - Option 2

3. **Save Data to JSON File (Option 3):**

   This option calls the FileProcessor.write_data_to_file method to save the students list to the JSON file, ensuring that all data is properly stored (Figure 10).

```
# Save the data to a file
elif menu_choice == "3":
    FileProcessor.write_data_to_file(file_name=FILE_NAME,student_data=students)
    continue
```

Figure 10: Save the data to a file - Option 3

4. **Exit the Program (Option 4) and Handle Invalid Menu Choice:**

   This option simply breaks out of the program loop, ending the program.

Handling each menu option by calling specific methods made the code easier to follow and debug. By keeping the main loop simple, I can focus on fixing issues in individual methods without worrying about breaking the entire program.

Testing the Program

I tested the program by running it in both PyCharm and Terminal and entering multiple student records. I checked for the following scenarios:

- Registering students with valid and invalid input.
- Displaying the list of registered students, including cases where no students are registered.
- Saving the data to the JSON file and verifying its contents in a text editor.
- Handling file errors, such as missing files or corrupted JSON data.

All tests were successful, and the program handled errors gracefully, providing helpful feedback to the user.

Summary

This assignment demonstrated the importance of organizing code into functions and classes, as well as adhering to the separation of concerns programming pattern. By modularizing the program, I was able to make the code more maintainable and scalable. The use of classes streamlined functionality, while structured error handling improved reliability. These skills are essential for writing professional-grade Python programs.