

Course Registration - Classes and Objects

Introduction

Assignment 07 involves creating a Python script which gives the user the following options:

1. Register a student for a course
2. Show current data
3. Save data to a file
4. Exit the program

In addition to the above actions, the program will initially open up an existing file named **"Enrollments.json"**, store the existing data in a variable (list of objects), and write it back to the truncated file (list of dictionaries) with any additional data the user provides.

Code

The decision making process of the code allowing the user to have multiple options is handled through **if** statements. The ability given to the user to make multiple choices is handled through an infinite looping process using the **while True** statement, until the user decides to close the program.

```
# ----- #  
# Title: Assignment07  
# Desc: This assignment demonstrates using data classes  
# with structured error handling  
# Change Log: (Who, When, What)  
#   JNoumeh,11/30/2023,Created Script  
#   <Your Name Here>,<Date>,<Activity>  
# ----- #
```

Figure 1: Script Header

```

# Present and Process the data
while True:

    # Present the menu of choices
    IO.output_menu(menu=MENU)

    menu_choice = IO.input_menu_choice()

    # Input user data
    if menu_choice == '1':
        IO.input_student_data(student_data=students)

    # Present the current data
    elif menu_choice == '2':
        IO.output_student_courses(students)

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

    # Stop the loop
    elif menu_choice == '4':
        break
    else:
        print('Please only choose option 1, 2, or 3')

print('Program Ended')

```

Figure 2: Script Processing Overview

The program utilizes classes, practicing encapsulation, to carry out the function of the program. Using the separation of concerns pattern, the script is divided into three sections: Data, Processing, and Presentation.

Data

The code begins with the initialization of constants and variables. Constants are set with predefined values which do not change, and variables are initialized to empty values with a string or list data type. Note the usage of triple apostrophe for the purpose of preserving the format of multi-line strings.

In addition to the data, the json module is imported to make handling the files more convenient.

```

import json

# Data ----- #
# Define the Data Constants
MENU: str = ''

---- Course Registration Program ----
    Select from the following menu:
        1. Register a Student for a Course.
        2. Show current data.
        3. Save data to a file.
        4. Exit the program.
-----
'''

FILE_NAME: str = "Enrollments.json"

# Define the Data Variables
students: list = [] # a table of student data
menu_choice: str = '' # hold the choice made by the user.

```

Figure 3: Initialization of Constants and Variables

Two classes also exist in this section. The **Person** class uses a constructor to initialize the student first and last name to an empty string. It then uses a getter and setter method to assign each variable its respective value while error handling. The **Student** class inherits the **Person** class and is therefore a subclass of it. In addition to inheriting the attributes associated with the **Person** class, the **Student** class creates and assigns a value to the student course variable. Note that the intent for these classes are for objects to be created from them.

```

class Person:
    """A class representing person data..."""

    # TODO Add first_name and last_name properties to the constructor (Done)
    def __init__(self, student_first_name: str = '', student_last_name: str = ''):
        """Initialises student name to empty strings..."""
        self.student_first_name = student_first_name
        self.student_last_name = student_last_name

    # TODO Create a getter and setter for the first_name property (Done) as in the Student class
    @property
    # (Use this decorator for the getter or accessor)
    def student_first_name(self):...

    @student_first_name.setter
    def student_first_name(self, value: str):
        if value.isalpha() or value == "": # is character or empty string
            self.__student_first_name = value
        else:
            raise ValueError("The last name should not contain numbers.")

    # TODO Create a getter and setter for the last_name property (Done)
    @property
    def student_last_name(self):
        return self.__student_last_name.title() # formatting code

    @student_last_name.setter
    def student_last_name(self, value: str):
        if value.isalpha() or value == "": # is character or empty string
            self.__student_last_name = value
        else:
            raise ValueError("The last name should not contain numbers.")

    # TODO Override the __str__() method to return Person data (Done)
    def __str__(self):
        return f'{self.student_first_name},{self.student_last_name}'

```

Figure 4: Person Class

```

class Student(Person):
    """A class representing student data...."""

    # TODO call to the Person constructor and pass it the first_name and last_name data (Done)
    def __init__(self, student_first_name: str = '', student_last_name: str = '', course_name: str = ''):
        super().__init__(student_first_name=student_first_name, student_last_name=student_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)
    4 usages (2 dynamic)
    @property # getter
    def course_name(self):
        return self.__course_name

    # TODO add the setter for course_name (Done)
    4 usages (2 dynamic)
    @course_name.setter # setter
    def course_name(self, value: str):
        self.__course_name = value

    # TODO Override the __str__() method to return coma separated string Student data (Done)
    # Override the Parent __str__() method behavior to return a coma-separated string of data
    def __str__(self):
        return f'{self.student_first_name},{self.student_last_name},{self.course_name}'

```

Figure 5: Student Subclass

Processing

This section of the code contains the class **FileProcessor** containing two static methods. The first method, **read_data_from_file**, carries out the function of opening a specified file with pre-existing json data and storing this to a variable called **student_data** as a list of objects. The other method, **write_data_to_file**, writes the data from a variable called **student_data** to a truncated version of the prior file. During the process, the data goes from a list of objects to a list of dictionaries suitable for the json file. Both of these are static methods containing structured error handling.

```

# Processing ----- #
2 usages
class FileProcessor:
    """ ... """

    1 usage
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """This function reads data from a json file and loads it into a list of objects..."""
        try:
            file = open(file_name, "r")
            list_of_dictionary_data = json.load(file)
            for student in list_of_dictionary_data:
                student_object: Student = Student(student_first_name=student["FirstName"],
                                                    student_last_name=student["LastName"],
                                                    course_name=student["CourseName"])
                student_data.append(student_object)
            file.close()
        except FileNotFoundError as e:
            IO.output_error_messages(message='Text file must exist before running this script!', error=e)
        except Exception as e:
            IO.output_error_messages(message='Error: There was a non-specific error!', error=e)
        finally:
            if file.closed == False:
                file.close()
        return student_data

```

Figure 6: Read from File Method

```

@staticmethod
def write_data_to_file(file_name: str, student_data: list):
    """This function writes data (list of objects) to a json file with data from a list of dictionary rows..."""
    try:
        list_of_dictionary_data: list = []
        for student in student_data:
            student_json: dict \
                = {"FirstName": student.student_first_name,
                  "LastName": student.student_last_name,
                  "CourseName": student.course_name}
            list_of_dictionary_data.append(student_json)
        file = open(file_name, "w")
        json.dump(list_of_dictionary_data, file)
        file.close()
    except TypeError as e:
        IO.output_error_messages(message="Please check that the data is a valid JSON format", error=e)
    except Exception as e:
        IO.output_error_messages(message="There was a non-specific error!", error=e)
    finally:
        if file.closed == False:
            file.close()

```

Figure 7: Write to File Method

Presentation

This section of the script contains the class **IO** which handles the static methods used to manage user input and output.

```
# Presenting ----- #
11 usages
class IO:
    """
    A collection of presentation layer functions that manage user input and output
    output_error_messages: prints param:message to the user and prints error information
    | if param:error contains one

    ChangeLog: (Who, When, What)
    JNoumeh,12.2.2023,Created Class
    """
```

Figure 8: Presentation: Class IO

The first method, **output_error_message**, develops a template for error handling to be called in other portions of methods containing exception blocks. This block of code prints the error, any documentation associated with it and its type.

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

    ChangeLog: (Who, When, What)
    JNoumeh,11/25/2023,Created method

    """
    print(message, end="\n\n")
    if error is not None:
        print("-- Technical Error Message -- ")
        print(error, error.__doc__, type(error), sep='\n')
```

Figure 9: IO Class - Error Message Method

The next method, **output_menu**, prints the menu out to the user. Naturally, the following method, **input_menu_choice**, requests the option from the menu the user desires. An **if** statement assists the user in making a choice the script can acknowledge. The stored variable **choice** is then returned to make it globally accessible.

```
@staticmethod
def output_menu(menu: str):
    """ Prints the menu to the user

    ChangeLog: (Who, When, What)
    JNoumeh,12.2.2023,Created Class

    :param menu: string with menu options for user to choose from

    :return: None
    """
    print()
    print(menu)
    print()

1 usage
@staticmethod
def input_menu_choice():
    """ prompts the user to enter a menu choice and stores that choice

    ChangeLog: (Who, When, What)
    JNoumeh,12.2.2023,Created Class

    :return: string with user choice
    """
    choice = "0"
    try:
        choice = input('Enter your menu choice number: ')
        if choice not in ('1', '2', '3', '4'):
            raise Exception('Please, choose only 1, 2, 3, or 4')
    except Exception as e:
        IO.output_error_messages(e.__str__())

    return choice
```

Figure 10: IO Class - Output/Input Menu

In anticipation of the first menu choice which allows the user to input a registration for a student, a method named **input_student_data** is created. This method requests the necessary information from the user while using structured error handling to help the user avoid any errors in inputting names. This process creates an instance of the **Student** subclass to append to the list of objects variable **student_data**.

```
@staticmethod
def input_student_data(student_data: list):
    """ Retrieves student name and course and appends it to a list in dictionary format

    ChangeLog: (Who, When, What)
    JNoumeh, 12.2.2023, Created Class

    :param student_data: list containing dictionaries

    :return: list of dictionaries appended with user data
    """
    try:
        # Input the data
        student = Student()
        student.student_first_name = input("Enter the student's first name: ")
        student.student_last_name = input("Enter the student's last name: ")
        student.course_name = input('Please enter the name of the course: ')
        student_data.append(student)

    except ValueError as e:
        IO.output_error_messages(message="One of the values was 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 11: IO Class - User Inputs

The final method in the class **IO** is **output_student_courses**. This method uses the variable **student_data**, which is a list of objects, to call out the attributes for each student.

```

@staticmethod
def output_student_courses(student_data: list):
    """ Prints out the student and course stored in a list of dictionaries

        ChangeLog: (Who, When, What)
        JNoumeh,12.2.2023, Created Class

    :param student_data: list of dictionaries

    :return: None
    """
    print('-'*50)
    for student in student_data:
        message = "{} {} is enrolled in {}".format(
            student.student_first_name, student.student_last_name, student.course_name)
        print(message)
    print('-'*50)

```

Figure 12: IO Class - Registration Output

Process

```
# Start of main body
# When the program starts, read the file data into a list of dictionaries
# Extract the data from the file
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)

# Present and Process the data
while True:

    # Present the menu of choices
    IO.output_menu(menu=MENU)

    menu_choice = IO.input_menu_choice()

    # Input user data
    if menu_choice == '1':
        IO.input_student_data(student_data=students)

    # Present the current data
    elif menu_choice == '2':
        IO.output_student_courses(students)

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

    # Stop the loop
    elif menu_choice == '4':
        break
    else:
        print('Please only choose option 1, 2, or 3')

print('Program Ended')
```

Figure 13: Script Process

Figure 13 shows the process of the script utilizing the classes and methods previously discussed. The first step occurs before the while loop is initiated. That is to read and store the data from the **FILE_NAME** constant set to “**Enrollments.json**”. The variable list **student_data** receives its first set of data (objects) in this line of code.

The next line begins the indefinite loop (**while True**) of presenting the menu of options then processing which option the user chooses with a series of **if** statements. The returned local

variable **choice** is assigned to the global variable **menu_choice** through **IO.input_menu_choice()**.

If the user decides to register a student for a course (Option 1), the data gets appended to the global variable **student_data**. This will add the data to the previously stored data read from the json file. Option 2 displays all information stored in **student_data** at the time of selection. Option 3 will write this information back to a truncated "**Enrollments.json**" file. Option 4 terminates the program.

Summary

The Python script is able to read a json file with existing data, save that data to a list, add additional data per user input and finally write this information back to the same file and save it. The following attached pages display the successful run of the code within both the Pycharm IDE and Command Prompt.

Pycharm Run

```
---- Course Registration Program ----
Select from the following menu:
  1. Register a Student for a Course.
  2. Show current data.
  3. Save data to a file.
  4. Exit the program.
-----

Enter your menu choice number: 2
-----

Bob Smith is enrolled in Python 100
Sue Jones is enrolled in Python 100
-----

---- Course Registration Program ----
Select from the following menu:
  1. Register a Student for a Course.
  2. Show current data.
  3. Save data to a file.
  4. Exit the program.
-----

Enter your menu choice number: 1
Enter the student's first name: Sally
Enter the student's last name: Mae
Please enter the name of the course: Python 100
```

```
---- Course Registration Program ----
Select from the following menu:
  1. Register a Student for a Course.
  2. Show current data.
  3. Save data to a file.
  4. Exit the program.
-----

Enter your menu choice number: 3
```

```
---- Course Registration Program ----
Select from the following menu:
  1. Register a Student for a Course.
  2. Show current data.
  3. Save data to a file.
  4. Exit the program.
-----

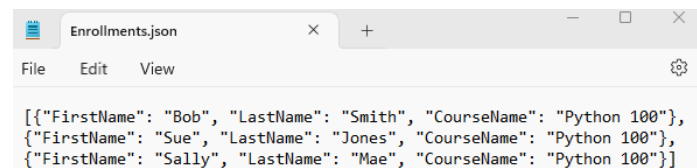
Enter your menu choice number: 2
-----

Bob Smith is enrolled in Python 100
Sue Jones is enrolled in Python 100
Sally Mae is enrolled in Python 100
-----

---- Course Registration Program ----
Select from the following menu:
  1. Register a Student for a Course.
  2. Show current data.
  3. Save data to a file.
  4. Exit the program.
-----

Enter your menu choice number: 4
Program Ended

Process finished with exit code 0
```



The screenshot shows a web browser window with a single tab titled "Enrollments.json". The address bar is empty, and the page content displays a JSON array of three objects, each representing a student's enrollment. The objects are: {"FirstName": "Bob", "LastName": "Smith", "CourseName": "Python 100"}, {"FirstName": "Sue", "LastName": "Jones", "CourseName": "Python 100"}, and {"FirstName": "Sally", "LastName": "Mae", "CourseName": "Python 100"}.

```
[{"FirstName": "Bob", "LastName": "Smith", "CourseName": "Python 100"}, {"FirstName": "Sue", "LastName": "Jones", "CourseName": "Python 100"}, {"FirstName": "Sally", "LastName": "Mae", "CourseName": "Python 100"}]
```

Command Prompt Run

```
---- Course Registration Program ----
Select from the following menu:
  1. Register a Student for a Course.
  2. Show current data.
  3. Save data to a file.
  4. Exit the program.
-----

Enter your menu choice number: 2
-----
Bob Smith is enrolled in Python 100
Sue Jones is enrolled in Python 100
Sally Mae is enrolled in Python 100
-----

---- Course Registration Program ----
Select from the following menu:
  1. Register a Student for a Course.
  2. Show current data.
  3. Save data to a file.
  4. Exit the program.
-----

Enter your menu choice number: 1
Enter the student's first name: Taylor
Enter the student's last name: Guitar
Please enter the name of the course: Python 100

---- Course Registration Program ----
Select from the following menu:
  1. Register a Student for a Course.
  2. Show current data.
  3. Save data to a file.
  4. Exit the program.
-----

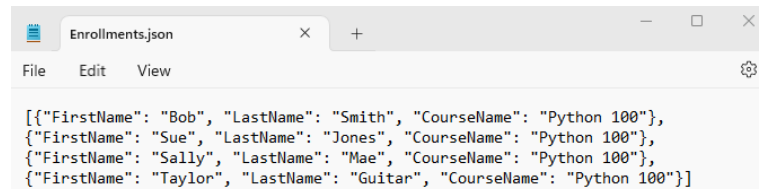
Enter your menu choice number: 2
-----
Bob Smith is enrolled in Python 100
Sue Jones is enrolled in Python 100
Sally Mae is enrolled in Python 100
Taylor Guitar is enrolled in Python 100
-----
```

```
---- Course Registration Program ----
Select from the following menu:
  1. Register a Student for a Course.
  2. Show current data.
  3. Save data to a file.
  4. Exit the program.
-----
```

Enter your menu choice number: 3

```
---- Course Registration Program ----
Select from the following menu:
  1. Register a Student for a Course.
  2. Show current data.
  3. Save data to a file.
  4. Exit the program.
-----
```

Enter your menu choice number: 4
Program Ended



The screenshot shows a web browser window with a single tab titled 'Enrollments.json'. The address bar is empty, and the page content displays a JSON array of four objects, each representing a student's enrollment. The objects are: Bob Smith in Python 100, Sue Jones in Python 100, Sally Mae in Python 100, and Taylor Guitar in Python 100. The browser's menu bar (File, Edit, View) and a settings icon are visible at the top.

```
[{"FirstName": "Bob", "LastName": "Smith", "CourseName": "Python 100"}, {"FirstName": "Sue", "LastName": "Jones", "CourseName": "Python 100"}, {"FirstName": "Sally", "LastName": "Mae", "CourseName": "Python 100"}, {"FirstName": "Taylor", "LastName": "Guitar", "CourseName": "Python 100"}]
```