Alejandro Guillen

November 28, 2023

IT FDN 110 B

ASSIGNMENT 07: Classes and Objects

Introduction

A class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made.

Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by their class) for modifying their state.

Class: The class is a user-defined data structure that binds the data members and methods into a single unit. Class is a **blueprint or code template for object creation**. Using a class, you can create as many objects as you want.

Object: An **object is an instance of a class**. It is a collection of attributes (variables) and methods. We use the object of a class to perform actions.

Objects have two characteristics: They have states and behaviors (object has attributes and methods attached to it) Attributes represent its state, and methods represent its behavior. Using its methods, we can modify its state.

Every object has the following property:

- **Identity**: Every object must be uniquely identified.
- **State**: An object has an attribute that represents the state of an object, and it also reflects the property of an object.
- **Behavior**: An object has methods that represent its behavior.

Step 1: Define Libraries, constants, variables.

First, it was to define the libraries JSON, constants, variables and organize the code according to the template for this module and define the input and outputs. The variables were set to empty string, student_data for row was set as dictionary.

```
import json

# Define the Data Constants
FILE_NAME: str = 'Enrollments.json'
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.
----------------------------------------
'''

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

Step 2: Class Person

This class represents person data, it is the super class. I has two attributes: first_name and last_name. I added a constructor, getter, and setter for each one.

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

    Properties:
        first_name (str): The student's first name.
        last_name (str): The student's last name.

    ChangeLog:
        - AGuillen, 11.28.2023: Created the class.
    """

    # TODO 2 Add first_name and last_name properties to the constructor
    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
```

```
    @property  # (Use this decorator for the getter or accessor)
    def first_name(self):
        return self.__first_name.title()  # formatting code

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

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

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

    # TODO Override the __str__() method to return Person data
    def __str__(self):
        return f'{self.first_name},{self.last_name}'
```

Step 3: Class Student

This class represents student data. It has three attributes: first_name and last_name inherited from class Person and course_name. I added a constructor, getter, and setter for course_name.

```
# TODO Create a Student class the inherits from the Person class
class Student(Person):
    """
    A class representing student data.

    Properties:
        first_name (str): The student's first name.
        last_name (str): The student's last name.
        course_name (str): The course_name of the student.

    AGuillen,11.28.2023,Created Class
    """

    def __init__(self, first_name: str = '', last_name: str = '', course_name:
str = '') -> None:
        # TODO call to the Person constructor and pass it the first_name and
last_name data
        super().__init__(first_name=first_name, last_name=last_name)
        # TODO add a assignment to the course_name property using the
course_name parameter
        self.course_name = course_name
```

```
    # TODO add the getter for course_name
    @property
    def course_name(self):
        return self.course_name

    # TODO add the setter for course_name
    @course_name.setter
    def course_name(self, value: float):
        self.course_name = value

    # TODO Override the __str__() method to return the Student data
    def __str__(self) -> str:
        # return f'{self.first_name},{self.last_name},{self.course_name}'
        return f"You have registered {self.first_name} {self.last_name} for
{self.course_name}."
```

Step 4: Class FileProcessor

The program includes a class named FileProcessor, inside this class I define read_data_from_file() and write_data_to_file(). When the program starts, the program opens the file: "Enrollments.json".

If the file exists is automatically read and then held into the list of dictionary student_data. The program opens File_NAME object using the open function in mode read. Also, I have added the exception handled in this part, due to the object FILE_NAME should exist before and it must have data. If the file does not exist, so will catch the exception FileNotFoundError or another exception, it will run the function: IO.output_error_messages and then will create the file as well. This function read_data_from_file() return student_data.

The function write_data_to_file() opens the file in mode write, use dump() function to save the data from students list to file and close. Also, each student row from students_data list is returned and displayed into choice 3 (menu_choice)

```
class FileProcessor:
    """
    A collection of processing layer functions that work with Json files

    AGuillen,11.28.2023,Created Class
    """

    @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 dictionary rows

        AGuillen,11.28.2023,Created function
        AGuillen,11.28.2023,Converted list of dictionaries to list of student
```

```
        :param file_name: string data with name of file to read from
        :param student_data: list of dictionary rows to be filled with file data

        :return: list
        """
        try:
            file = open(file_name, "r")
            student_data = json.load(file)
            print('The file data was loaded successfully')
            file.close()

        except FileNotFoundError as e:
            IO.output_error_messages("Text file must exist before running this
script!", e)
        except Exception as e:
            IO.output_error_messages("There was a non-specific error!", e)
        finally:
            if file.closed == False:
                file.close()
        return student_data

    @staticmethod
    def write_data_to_file(file_name: str, student_data: list[Student]):
        """ This function writes data to a json file with data from a list of
dictionary rows

        AGuillen,11.28.2023,Created function

        :param file_name: string data with name of file to write to
        :param student_data: list of dictionary rows to be writen to the file

        :return: None
        """
        try:
            file = open(file_name, "w")
            json.dump(student_data, file)
            file.close()
            print("The following data was saved to file!")
            for student in student_data:
                print(f'Student {student["FirstName"]} '
                      f'{student["LastName"]} is enrolled in
{student["CourseName"]}')
        except TypeError as e:
            IO.output_error_messages("That value is not the correct type of
data!", e)
        except Exception as e:
            IO.output_error_messages("There was a non-specific error!", e)
        finally:
            if not file.closed:
                file.close()
```

Step 5: ***Class IO***

The program includes a class named IO, inside this class I define output_error_messages () to handle errors, output_menu() to display the menu, input_menu_choice() to input the choice number, add_data_to_table() to input the student to student_data list and output_current_data() to display the current data.

```
class IO:
    """
    A collection of presentation layer functions that manage user input and
output

    AGuillen,11.28.2023
    """

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

        AGuillen,11.28.2023,Created function

        :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_messages(message: str):
        """
        This function displays a custom messages to the user

        :param message:

        :return: None
        """
        print(message, end="\n")

    @staticmethod
    def output_menu(menu: str):
        """ This function displays the menu of choices to the user

        AGuillen,11.28.2023,Created function

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

```
    @staticmethod
    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__())  # passing the exception
object to avoid the technical message

        return choice

    @staticmethod
    def output_current_data(student_data: list):
        """ This function displays the current data to the user

        :return: None
        """
        print('=' * 40)
        for student in student_data:
            # print(f"FirstName: {student['FirstName']}, LastName:
{student['LastName']}", CourseName: {student['CourseName']})
            print(f"{student.first_name} {student.last_name},
{student.course_name}")
        print('=' * 40, end='\n\n')

    @staticmethod
    def output_student_courses(student_data: list[Student]):
        """
        Outputs all students currently stored in the students directory

        :param student_data: The students list of dictionaries
        :return: None
        """
        print("-" * 50)
        print("\nThe current data is:")
        # print(f"List Data: {student_data}")
        for student in student_data:
            print(f'Student {student["FirstName"]} {student["LastName"]} is
enrolled in {student["CourseName"]}')

    @staticmethod
    def input_student_data(student_data: list[Student]):
        """ This function gets the first name, last name, and course name from
the user

        AGuillen,11.28.2023,Created function

        :param student_data: list of dictionary rows to be filled with input
data
        :return: list updated
        """
        while True:
            try:
```

```
                first_name = input("Enter the student's first name: ")
                if not first_name.isalpha():
                    raise ValueError("The first name should not contain
numbers.")
                break
            except ValueError as e:
                IO.output_error_messages("Only use names without numbers", e)
        while True:
            try:
                last_name = input("Enter the student's last name: ")
                if not last_name.isalpha():
                    raise ValueError("The last name should not contain
numbers.")
                break
            except ValueError as e:
                IO.output_error_messages("Only use names without numbers", e)
        course_name = input("Enter the name of the course: ")
        student_data.append({"FirstName": first_name, "LastName": last_name,
"CourseName": course_name})
        print(f"You have registered {first_name} {last_name} for
{course_name}.")

        return student_data
```

Step 6: Load the program:

When the program starts, it loads the data from the file Enrollmen.json. Also the data is passed to students list. It allows to use this data for display the data and update the file after a student is enrolled.

```
# Beginning of the main body of this script
students = FileProcessor.read_data_from_file(file_name=FILE_NAME,
student_data=students)
```

Step 7: While loop:

This while loop allows the user to choose for the different options according to the menu:

Choice 1: Runs the function

students = IO.input_student_data(student_data=students)

Add a new student and enroll in a course. Also this option display the data using the function: IO.output_student_courses(student_data=students)

Choice 2: Runs the function

IO.output_student_courses(student_data=students)

to display the current students enrolled.

Choice 3: Runs the function

FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students) to add a new student and enroll in a course.

Choice 4: break out the while loop and display the message "Program Ended".

IO.output_messages("Program Ended")

```
while True:
    IO.output_menu(menu=MENU)

    menu_choice = IO.input_menu_choice()

    if menu_choice == "1":  # Display current data
        # IO.input_student_data(student_data=students)
        students = IO.input_student_data(student_data=students)
        print("\n The data was updated")
        IO.output_student_courses(student_data=students)

        continue

    elif menu_choice == "2":  # Get new data (and display the change)
        # students = IO.input_student_data(student_data=students)
        # IOProcessor.output_current_data(student_data=student_table)
        IO.output_student_courses(student_data=students)

        continue

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

    elif menu_choice == "4":  # End the program
        IO.output_messages("Program Ended")
        break  # out of the while loop

    else:
        IO.output_messages("Please only choose option 1, 2, or 3")
```

Summary

- I have learned in this module about how to make an application more flexible to other applications and keep our code clear, and reusable. Python is an Object-Oriented Programming language, so everything in Python is treated as an object. An object is a real-life entity. It is the collection of various data and functions that operate on those data.
- Using classes and objects is a better practice because it promotes encapsulation, reusability, and predictable behavior in the functions. Also, the code is more reusable, and more maintainable code.