Hannah Brown
6/1/2025
IT FDN 110 A
Assignment 07
https://github.com/hkbrown42/hkbrown42-IntroToProg-Python-Mod07

Assignment 07 - Classes and Objects

Introduction

This week we got more in-depth into classes, including the different kinds of classes (like data classes, presentation classes, and processing classes), as well as getting into creating custom objects that can make complex code easier to read, reuse, and maintain. This document describes the process of how I modified the starter file to include several new classes, as well as adding getter and setter functions and modifying the code to use custom objects.

Initial Setup

Since I was once again starting with a starter file instead of from scratch, doing the initial setup of the code was quick and straightforward - I verified the json module was imported and made sure that the constants and global variables were defined correctly. I also copied the provided “Enrollments.json” file into the A07 project, so that there will be starting data to work with.

Modifying the Program

My process for this program was a little different than previous assignments, since the bulk of the program’s code will not be changing. Once I had the constants and variables set up, I moved sequentially through each of the TODO items in the code, updating the code and adding/modifying descriptive docstrings as necessary.

First, remaining in the Data section of the code, I created a new class called Person, with the properties first_name and last_name. I defined the constructor method and added these properties to it, then created the getter and setter functions for both the first_name and last_name properties, using private attributes. Finally, I created an override for the __str__() method so that it will return the person’s first name and last name.

Next, still in the Data section, I created a second new class called Student as a subclass of Person. I defined the constructor method by calling the constructor from the Person class, and added the course_name property. Since the Student class inherits from the Person class, I only need to create getter and setter functions for the course_name property, since the getter and setter functions for first_name and last_name were defined in the Person class. Finally, I created an override here as well for the __str__() so it will return the student’s first name, last name, and course name.

```python
class Person:  1 usage
    """
    A class representing a person's data

    Properties:
    - first_name (str): the person's first name
    - last_name (str): the person's last name

    Change Log:
    HBrown, 5/31/2025, Created class
    """

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


    @property  4 usages (2 dynamic)
    def first_name(self):
        return self._first_name.title()
    @first_name.setter  3 usages (2 dynamic)
    def first_name(self, name: str):
        if name.isalpha() or name == "":
            self._first_name = name
        else:
            raise ValueError("The first name cannot contain numbers!")


    @property  4 usages (2 dynamic)
    def last_name(self):
        return self._last_name.title()
    @last_name.setter  3 usages (2 dynamic)
    def last_name(self, name: str):
        if name.isalpha() or name == "":
            self._last_name = name
        else:
            raise ValueError("The last name cannot contain numbers!")


    def __str__(self):
        return f"{self.first_name},{self.last_name}"
```

Figure 1: Definition of Person class.

```python
class Student(Person):  3 usages
    """
    A class representing a student's data

    Properties:
    - first_name (str): the student's first name
    - last_name (str): the student's last name
    - course_name (str): the name of the course the student is registering for

    Change Log:
    HBrown, 5/31/2025, Created class
    """

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


    @property  5 usages (2 dynamic)
    def course_name(self):
        return self._course_name.title()
    @course_name.setter  4 usages (2 dynamic)
    def course_name(self, name: str):
        self._course_name = name


    def __str__(self):
        return f"{self.first_name},{self.last_name},{self.course_name}"
```

Figure 2: Definition of Student subclass.

Moving on to the Processing section, in the FileProcessor class, I then modified the read_data_from_file() method so that when the program runs and the initial JSON data is loaded in, it converts the list of dictionaries from the JSON file into a list of Student objects. I also added additional error handling here to account for if the file does not actually exist.

```
try:
    # Get a list of dictionary rows from the data file
    file = open(file_name, 'r')
    json_students = json.load(file)

    # Convert the list of dictionary rows into a list of Student objects
    for student in json_students:
        student_object: Student = Student(first_name=student["FirstName"],
                                          last_name=student["LastName"],
                                          course_name=student["CourseName"])
        student_objects.append(student_object)
    file.close()
except FileNotFoundError as e:
    IO.output_error_messages(
        message="Please make sure this file exists!",error=e)
except Exception as e:
```

Figure 3: Modified code for read_data_from_file() method.

Next, remaining in the Processing section of the code and the FileProcessor class, I modified the code to convert the Student objects back into dictionaries before writing the data to the file. This is necessary because custom objects cannot be written to a JSON file; any custom objects must be converted to a standard data format before being written to a file.

```
try:
    student_data_dictionary: list = []
    for student in student_data:
        student_json: dict = {
            "FirstName": student.first_name,
            "LastName": student.last_name,
            "CourseName": student.course_name
        }
        student_data_dictionary.append(student_json)

    file = open(file_name, 'w')
    json.dump(student_data_dictionary, file, indent=2)
```

Figure 4: Modified code for write_data_to_file() method.

Now I can move onto the Presentation section of the code, into the IO class. I modified the output_student_and_course_names() method to access data from the Student object, rather than from the list of dictionary data. Although the code is modified, the program's output here will still appear the same to the user as the previous version of the code.

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

Figure 5: Modified code for output_student_and_course_names() method.

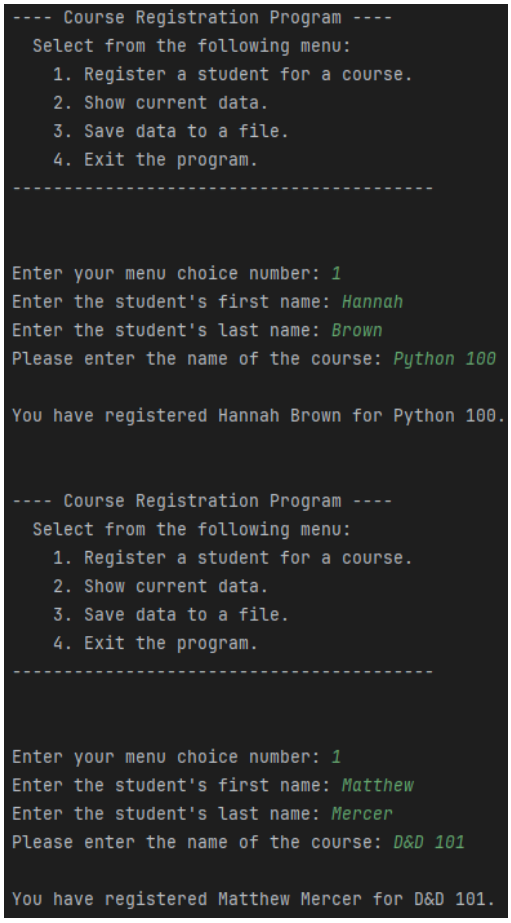
Next, I modified the code in the input_student_data() method to also use Student objects instead of dictionary objects. Again, the prompts for data input will appear the same as the previous version to the user.

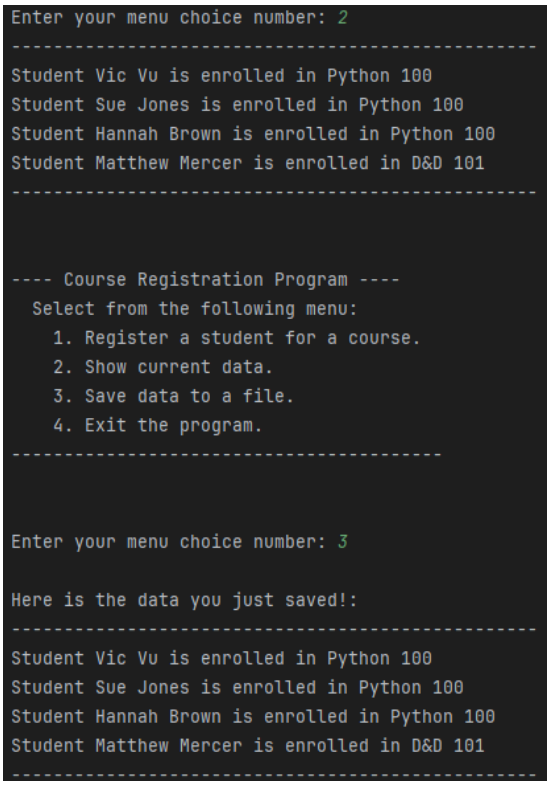
```
try:
    student = Student()
    student.first_name = input("Enter the student's first name: ")
    student.last_name = input("Enter the student's last name: ")
    student.course_name = input("Please enter the name of the course: ")
    student_data.append(student)
    print()
    print(f"You have registered {student.first_name} "
          f"{student.last_name} for {student.course_name}.")
```

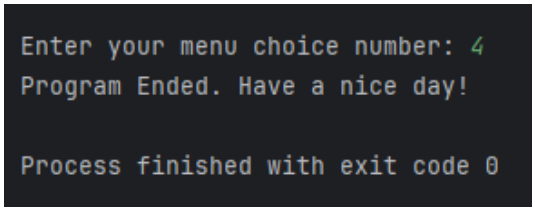
Figure 6: Modified code for input_student_data() method.

Testing

Once I had completed modifying the code, I ran the program in PyCharm, making sure each menu option selection behaved as expected. I tried entering in multiple registrations, and verified that all the new data was saved to the JSON file correctly after running the program.

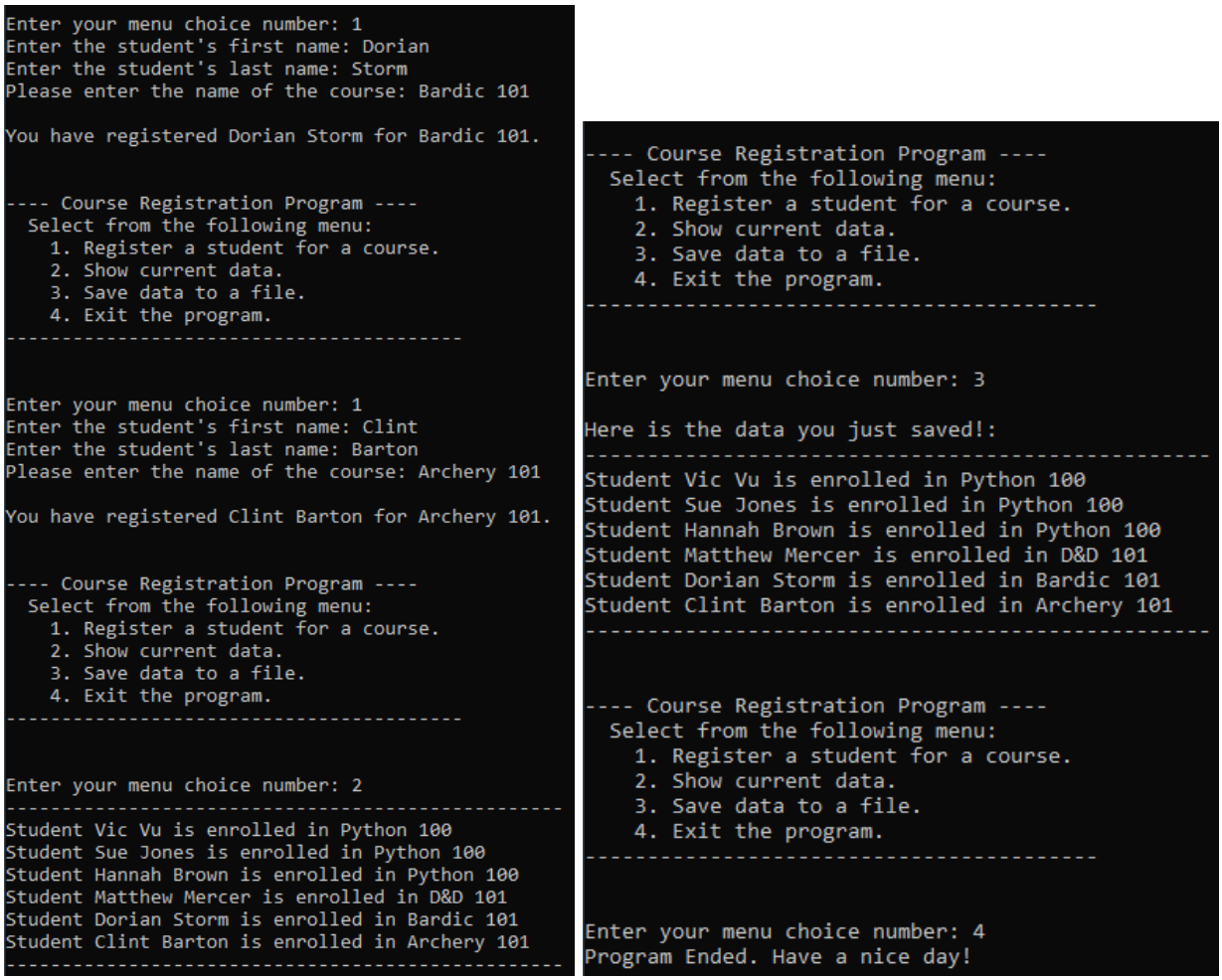






Figures 7-9: Program running in PyCharm.

After I was satisfied that the program ran here, I then opened the command prompt and ran the program there, again verifying that each menu option worked, and verifying the JSON file contained the new saved data after the program ran.



Figures 10-11: Program running in command prompt.

Enrollments.json × Assignment07.py

```
[
  {
    "FirstName": "Vic",
    "LastName": "Vu",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Sue",
    "LastName": "Jones",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Hannah",
    "LastName": "Brown",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Matthew",
    "LastName": "Mercer",
    "CourseName": "D&D 101"
  }
]
```

Enrollments.json × Assignment07.py

```
[
  {
    "FirstName": "Vic",
    "LastName": "Vu",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Sue",
    "LastName": "Jones",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Hannah",
    "LastName": "Brown",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Matthew",
    "LastName": "Mercer",
    "CourseName": "D&D 101"
  },
  {
    "FirstName": "Dorian",
    "LastName": "Storm",
    "CourseName": "Bardic 101"
  },
  {
    "FirstName": "Clint",
    "LastName": "Barton",
    "CourseName": "Archery 101"
  }
]
```

Figures 12-13: Verifying JSON files contain correct data after running the program twice.

Summary

Having the lab demos felt crucial to me on this one - I'm not sure that I would have been able to figure out exactly how to utilize the custom objects in this one without having the lab demo effectively as a template to go off of. I'm continuing to find it interesting how the output of the program looks exactly the same as the previous version of the program did to the user, showing that there are many different ways to structure and organize a script.