Mohammad Ammar Bharmal

March 10, 2024

IT FDN 110 B

Assignment 08

Demonstrating how to create the multi-module applications and test these modules systematically

Introduction

This week is about demonstrating how to create the multi-module applications and test them systematically. The application is designed to get employee data from the user and store the information in the database (json file). The following report is the breakdown on how I wrote and tested this program step-by-step. The program is created in PyCharm 2022.2.1 with Python 3.12.2.

Creating the Program

The program is split into two sections as the heading of the report mentions – the multi-module application and testing of these modules. The first section is the implementation of the multi-module application. In both sections, the program is created the same way as the previous assignments, so I have not gone into the details of variables used, commenting style and other nitty gritty elements of the program.

Multi-module Application

Using the *Assignment08_Starter.py*, I split the program into four modules: *main.py*, *data_classes.py*, *presentation_classes.py* and *processing_classes.py*. Since there are multiple ways to write a program in Python to achieve the same results, I chose to start it by providing a brief description of what the program is about as a comment in the script. After that, I described what the program does and how to use it.

The *main.py* is the file which executes all the other modules to run the complete program. It contains global constants such as FILE_NAME and MENU and other supporting variables including list. It asks the user to choose from four options when the program is run as given below:

1. Show current employee rating data.
2. Enter new employee rating data.
3. Save data to a file.
4. Exit the program.

The *main.py* imports all the other modules to execute the programs. On opting menu choice 1, the application prompts the user to enter the employee's first name and last name, followed by the review date and rating. On menu choice 2, the application presents the collected data. Data collected for menu choice 1 is added to a two-dimensional list table (list of Employee objects). All data in the list is displayed when menu choice 2 is used. When the application starts, the contents of the *EmployeeRatings.json* are automatically read into a two-dimensional list table (a list of Employee object rows). On menu choice 3, the application saves its data to the file named *EmployeeRatings.json* and on menu choice 4, the application ends.

In the programs, the application provides structured error handling when the file is read into the list of employee rows, when the user enters a first name and when the user enters a last name. Moreover, the application also provides structured error handling when the user enters a review date in specific YYYY-MM-DD date format and when the user enters a review rating (1,2,3,4, or 5). The structured error handling is also provided when the data is written to the file.

In *data_classes.py* module, there are two classes: Person and Employee. The Person class has only two variables: "*first_name*" and "*last_name*". The Employee class borrows the base class from Person and has two more variables: "*review_date*" and "*review_rating*". The classes also raise value errors if the datatype is not respected as declared.

The third module, *presentation_classes.py*, comprises of only one class which has five functions. The name of the class is *IO* and it carries out the input and output functionalities in the application. The functions enable menu output, menu choice input, and data input into and data output from the file using json. Likewise, the fourth module, *processing_classes.py*, processes files by reading and writing into them through two functions – one for each functionality. The name of the class is *FileProcessor*.

Each module has its own importing requirements based on the nature of algorithm processing. In figures 1.1.a to 1.1.d show the import files, class names and their descriptions in each module respectively.

```
1   # ------------------------------------------------------------------------------------------ #
2   # Title: Assignment08-Starter files
3   # # Description: A collection of classes for managing the application
4   # ChangeLog: Mohammad Ammar Bharmal, 03.07.2024, Created Script
5   # RRoot,1.5.2030,Created Script
6   # ------------------------------------------------------------------------------------------ #
7
8   from data_classes import Employee
9   from presentation_classes import IO
10  from processing_classes import FileProcessor
```

Figure 1.1.a: main.py

```python
from datetime import date

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

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

    ChangeLog: Mohammad Ammar Bharmal, 03.09.2024, Created the class
    - RRoot, 1.1.2030: Created the class.
    """
```

Figure 1.1.b.i: data_classes.py – class Person

```python
class Employee(Person):
    """
    A class representing employee data.

    Properties:
    - first_name (str): The employee's first name.
    - last_name (str): The employee's last name.
    - review_date (date): The data of the employee review.
    - review_rating (int): The review rating of the employee's performance (1-5)

    ChangeLog: Mohammad Ammar Bharmal, 03.09.2024, Created the class
    - RRoot, 1.1.2030: Created the class.
    """
```

Figure 1.1.b.ii: data_classes.py – class Employee

```python
from data_classes import Employee

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

    ChangeLog: Mohammad Ammar Bharmal, 03.09.2024, Created Class
    RRoot,1.1.2030,Created Class
    """
```

Figure 1.1.c: presentation_classes.py

```python
import json
from data_classes import Employee

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

    ChangeLog: Mohammad Ammar Bharmal, 03.09.2024, Created Class
    RRoot,1.1.2030,Created Class
    """
```

Figure 1.1.d: processing_classes.py

Testing the Modules

There is a dedicated file to test the functional modules of the program. The names of the testing files are *test_data_classes.py*, *test_presentation_classes.py* and *test_processing_classes.py*. Each file tests the

implemented algorithm in the primary functional modules of the program and the functions implemented in them.

The testing is implemented using *unittest* library. The *test_presentation_classes.py* tests the *presentation_classes.py* module where the user input is required. To make the testing independent of the user interference, the *mock* object of *unittest* is used by importing the *patch* decorator. The user inputs are pre-defined using the *patch*. Each module has its own importing requirements based on the nature of algorithm processing. In figures 1.2.a to 1.2.c show the import files, class names and their descriptions in each module respectively.

```
import unittest
from data_classes import Person, Employee

class TestPerson(unittest.TestCase):
    """
    Testing the class representing person data.

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

    ChangeLog: Mohammad Ammar Bharmal, 03.09.2024, Created the class
    - RRoot, 1.1.2030: Created the class.
    """
```

Figure 1.2.a.i: test_data_classes.py – class TestPerson

```
class TestStudent(unittest.TestCase):
    """
    Testing the class representing employee data.

    Properties:
    - first_name (str): The employee's first name.
    - last_name (str): The employee's last name.
    - review_date (date): The data of the employee review.
    - review_rating (int): The review rating of the employee's performance (1-5)

    ChangeLog: Mohammad Ammar Bharmal, 03.09.2024, Created the class
    - RRoot, 1.1.2030: Created the class.
    """
```

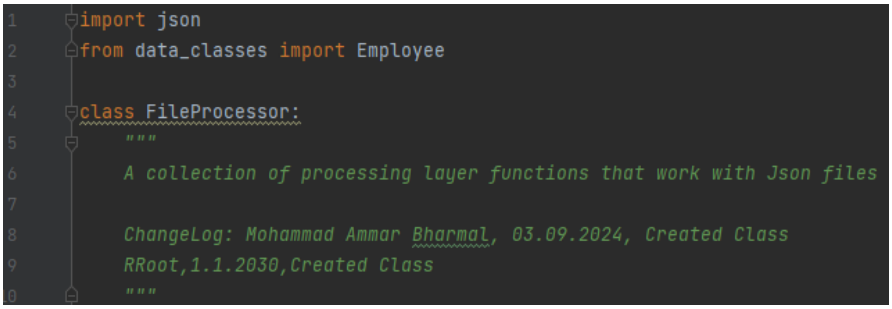
Figure 1.2.a.ii: test_data_classes.py – class TestEmployee

```
import unittest
from unittest.mock import patch
from presentation_classes import IO
from data_classes import Employee

class TestIO(unittest.TestCase):
    """
    Testing the collection of presentation layer test functions that manage user input and output

    ChangeLog: Mohammad Ammar Bharmal, 03.09.2024, Created Class
    RRoot,1.1.2030,Created Class
    """
```

Figure 1.2.b: test_presentation_classes.py



```python
import json
from data_classes import Employee

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

    ChangeLog: Mohammad Ammar Bharmal, 03.09.2024, Created Class
    RRoot,1.1.2030,Created Class
    """
```

Figure 1.2.c: test_processing_classes.py

Presenting and processing the data

Multi-module Application

The application is initiated by running the *main.py* file on IDE PyCharm. The user interface displays the menu for the user to choose what they want to do with the application. Whatever menu choice user inputs, it is collected as a string (string datatype for the input data), and the menu functions from *presentation_classes.py* module are called. Based on the user demand of writing the employee details into the *json* file or storing the information, the related function is called from the *processing_classes.py* module. The *EmployeeRatings.json* related is called back and forth to read or write employee information to or from it.

Testing the Modules

In testing modules, the application processes the testing modules the same way as the main application. It calls the functions as needed but the difference is the use of the *unittest* library. The *patch* decorator in *test_presentation_classes.py* file takes the test inputs and verifies the main modules. If there is an error in the main modules, the application displays it accordingly and the corrections are made by the algorithm writer.

Testing the Program

Multi-module Application

Figure 2.1.a shows how the menu is displayed and information is asked by the application. When the user chooses option 2, the employee details are captured and then displayed on the screen as shown in figure 2.1.b. Figure 2.1.c presents the confirmation of saving the data when the user chooses menu option 3.

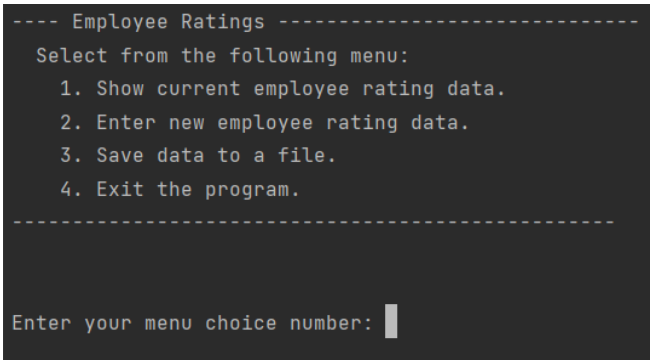


Figure 2.1.a: Presenting menu to the user and asking to input the desired option

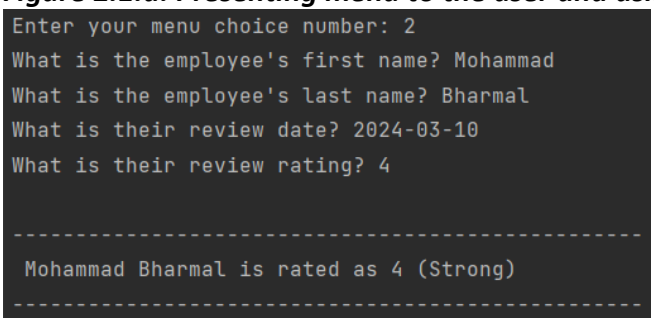


Figure 2.1.b: User inputs are processed and displayed by the application

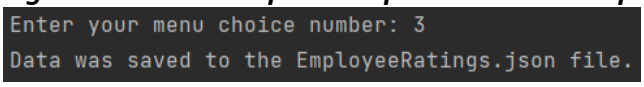


Figure 2.1.c: User inputs are saved by the application into the json file

Testing the Modules

In testing modules, the application is run using *unittest* command. The command to test the *processing_classes.py* module through the *test_processing_classes.py* file is shown in figure 2.2.a.

```
python -m unittest .\test_processing_file.py
```

Figure 2.2.a: Command for the user to run test_processing_classes.py

The test reports for the coverage of testing are also generated using the command shown below for *test_processing_classes.py* file:

python -m coverage run --include=processing_classes.py -m unit test .\test_processing_classes.py

followed by:

python -m coverage report

Figure 2.2.b to 2.2.d show the coverage reports for *test_data_classes.py*, *test_presentation_classes.py* and *test_processing_classes.py* respectively.

```
Name              Stmts   Miss  Cover
-----------------------------------
data_classes.py      48      0   100%
-----------------------------------
TOTAL                48      0   100%
```

Figure 2.2.b: Coverage of test_data_classes.py is 100%

```
Name                        Stmts   Miss  Cover
-------------------------------------------------
presentation_classes.py        57     30    47%
-------------------------------------------------
TOTAL                          57     30    47%
```

Figure 2.2.c: Coverage of test_presentation_classes.py is 47%

```
Name                      Stmts   Miss  Cover
-----------------------------------------
processing_classes.py        35     10    71%
-----------------------------------------
TOTAL                        35     10    71%
```

Figure 2.2.d: Coverage of test_processing_classes.py is 71%

The details of the coverage can also be seen in the *html* reports. The command for *html* report for the *test_data_classes.py* file is given below. We refer to *index* file by opening it in an internet browser to get into the details of the coverage.

```
python -m coverage html
```

The figure 2.3 shows the report in the *index* file. The details of the *index* file are shown in figure 2.4.

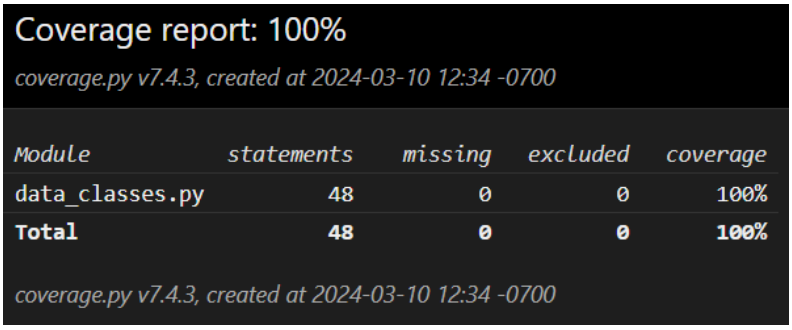


Figure 2.3: Coverage report of index file from html reporting

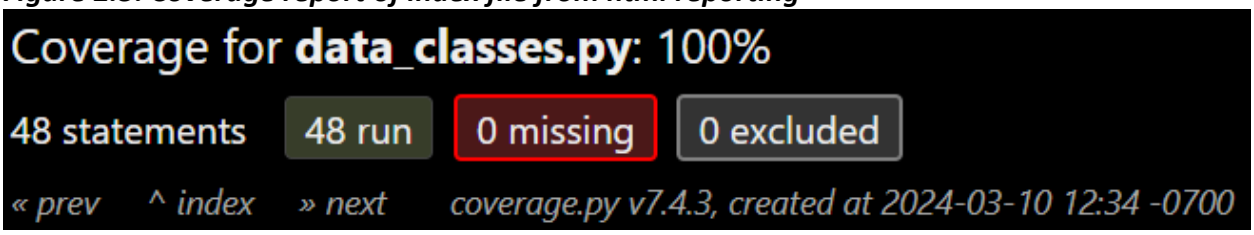


Figure 2.4: Details of the coverage of index file from html reporting

The application validation is important so understand the credibility of the application. If the details of two employees are input into the application and are saved using the menu choice 3, the *json* file should store that information. The application is run and validated from the *EmployeeRatings.json* file. Figure 2.5 shows the *json* file as the final validation of the application.

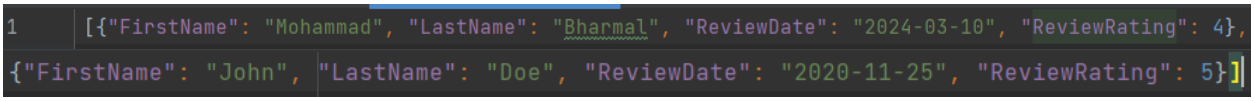


Figure 2.5: Validation of the application through the output of the json file

Summary

The assignment 8 demonstrates how to create the multi-module applications and test them systematically. The application is designed to get employee data from the user and store the information in the database (json file). Moreover, the opening of the program and commenting to code helps improve the longevity of the program. It makes the program look more professional, well documented and easy to debug/reuse in future.