Design and Programming

Assignment Two

Please read the COMPLETE document before commencing any work.

Preamble

In the previous Case Study exercise, you were introduced to the problem of dealing with notifications for library members. We had established that there were three types of notification required by the specification:

1. Notification that a reserved book had become available.
2. Notification that an ordered book had become available.
3. Notification that a book return was late and of a resultant fine.

During the design and implementation cycle or sometime thereafter it is likely that some stakeholder will point out the need for a further notification – when a membership card becomes available for a member, they will need to be informed that they can collect it. Discovering such requirements late in the development cycle can be costly if it requires compromising code that has already been written. This is where using good design patterns is important. They can ensure that the system can be built in such a way that new requirements can be met without heavy editing of the code that has already been completed.

I do not, of course, expect you to produce a complete library system. You currently have two files, **books.csv** and **bookloans.csv** representing books possessed by a library and loans of books made by the library over a time period. To this I have added **members.csv** which contains a list of members with card numbers and membership numbers. Any member with a card number of 0 is awaiting issue of a membership card. Note that the CSV files are encoded as UTF8.

Task One

We have not learned about databases and hence will have to make do with the storage methods covered during the course. For purposes of this exercise, you may open the files and utilise their data in-memory, saving the information back to file if it has been updated. You should think about providing an interface to the data that will allow other forms of data storage to be 'plugged in' in the future. Note that the JSON and PICKLE storage mechanisms will not deal with user-defined classes. Thus, to create any objects the relevant data will have to be extracted from the storage structure employed.

Tasks

1. Provide code that will allow a member to borrow a book, recording the results as a new bookloan. You can provide both Book and Member classes with a method **scan()** which will return the id of an instance of each respectively. Do not worry about representing a membership card. Cards can be an attribute of Member. Use dummy data to test the code and state the preconditions and postconditions of the operation in a **docstring**.

2. Provide code that will allow a member to return a book, again ensuring that necessary data is stored. Test the functionality with appropriate dummy data and again provide pre and postconditions as a **docstring**.

3. Provide functionality that will allow a member of the public to apply for membership. This will involve the storage of information on the membership request. Each day, a list of new member details is sent to an external print company who produces membership cards. Cards are delivered to the library approximately three days later, when the membership card-number is recorded. The format of the card number is made up of the membership number followed by a single digit indicating the sequence number of the card associated with the member. 1 means the first card issued to the member, 2 the second and so on. There is an obvious flaw here, but you can ignore it for purposes of the exercise.

4. Provide functionality to allow a member to reserve a book. Assume that they have the number of the book available. This will require permanent storage as it may be some time before a book becomes available.

5. Using the research that you conducted on design patterns, implement a notification system along the lines described in the Preamble to this assignment. You can assume that all notifications are sent by email, providing a test **sendEmail()** method which merely prints to the console the message passed as an argument. Test the system and show or describe how it is capable of coping with the situations described in the preamble and how it might be sufficiently flexible to deal with future notification requirements.

Parts 1 to 4 attract a maximum of 10 marks each and part 5, 20 marks. Within each part, 60% of the marks are awarded for code and functionality and 40% for good design and documentation. Submit your work as a Jupyter Notebook. You should also submit your data files (as JSON). It is not necessary to include the original CSV files, and your code should function without them. Your code should assume that the data files are in the same directory as the Jupyter Notebook. The JSON files should be placed in a ZIP file (using the ZIP format). Please do not submit RAR or other file formats. There are thus two files to submit – your Jupyter Notebook (.ipynb) and the data.zip file.