Personal Information:

Name: **Beatriz Glaser**

Student number: **1021033**

Study program: **Digital Systems and Design - Bachelor's Program**

Date: **05.05.2022**

General Description:

This project is a reservation system for any kind of Bowling Alley. It allows bowling companies to create their own "registers", which are saved in binary files. Their customers can then access the user-program, which loads a specific bowling company register and allows users to modify it.

Users can sign up, log in, make reservations, look at their valid reservations and cancel previous bookings. Once the user is done, the program uploads any changes to the bowling register (binary file) before exiting the program.

Originally, I would have like to have completed a graphic interface for the program. However, I was not able to implement that. On the project you can see and run different GUI windows, but they are not connected to the main program. For this reason, I believe I completed the project on the easy version (only with a text-based interface). On the program you can see an attempt at two medium-level features: an incomplete graphic interface and the saving of historical customer/service data (all reservations can be found saved on the customer's owner object).

Instructions for the user:

As briefly mentioned on the README file, you first have to add a text file to your repository and run the create_alley.py directory. This process creates a specific Bowling Alley register, which will be accessed by the customers/user. This directory allows the creation of different registers so it is also useful for testing.

The main function in the main.py directory starts by opening the file saved and loading the Bowling Alley register. Therefore, if you wish to load different registers, you need to modify the name of the file being opened by the function.

Once you run the main function in the main.py directory, you will see a series of user-instructions and input opportunities that guide the user through the register. From there on, all instructions are given on the text-based interface.

Additionally, by running the main_GUI.py directory you can explore through an illustrative graphical interface of the program.

External libraries:

This program only uses PyQt5 and other inbuilt python libraries (e.g.: pickle and datetime).

Structure plan:

For simplicity, I will be referring to the BowlingAlley class as BA.

The program itself is done in such way that it is easy to store its data on a single file. My classes are intertwined so that by saving a BA-object, you save information for the entire program (all customers, lanes and reservations that have ever been made). How is this done?

When initializing a BA-object, the class automatically creates a determined number of lane-objects and add them to a list, which is stored as an attribute of the BA-object. Each Lane-object is automatically set as empty (by the attribute "is_empty"). Another important attribute of the BA class is a Customer-object list. Every time a Customer-object is created (through the "add_customer" method) and stored on this attribute.

Every Customer-object stores a list of reservation-objects made by this customer. Reservation-objects are initialized by the method "add_reservation" on the Customer class. Each reservation is set by default as valid (by the attribute "is_valid") and assigned an empty lane.

The main.py function is responsible for running most of the program. When running it, the program reads a binary file that contains a BA-object. At the beginning of every run, the program gets the current day and time and compares it to the date and time of every valid reservation. If the date and time of such reservation has passed, it changes the "is_valid" attribute of the reservation to False, invalidating the reservation.

During the user-interaction the user can choose to log in to an existing account (aka Customer-object) or sign up (initialize a new Customer-object). Once logged in, the user can check reservations based on their validity, and make new reservations based on the BA availability. **The reservation system is done so that the user cannot reserve a lane if all lanes for that specific time are occupied.**

Before exiting, the program always saves the updated BA-object to the file that was previously loaded. I believe integrating the classes was the most efficient way of saving and loading from files (since I could use pickle). An alternative could have been storing the information using json, however that would have required a lot more coding to manually serialize each class. It could have gotten quite complex and confusing.

For the non-functional GUI interface, I used QMainWindow and QWidget classes. I decided to open new windows when PushButtons are clicked, because I thought it would reduce complexity (easier than constantly deleting and creating widgets). I chose to add only a few widgets to each window to illustrate a very clear and easily managed user interface. Nevertheless, I could combine/implement the program and the graphical interface.

Algorithms:

The program mostly uses simple mathematical comparisons ("<" smaller than, ">" greater than, "==" equals to and "!=" not equal). There is also some complicated algorithms involved in saving and loading the BA objects into binary files (converting to binary and back to human-readable text).

Data Structures:

As previously mentioned before, I used pickle-library to save and load BA-objects ("registers") to binary files. I believe that was the most efficient and clean way of saving and loading information.

The json option has already been discussed previously. A third option would have been to manually write a function that would write down all the BA register information in a human-readable way on a text file. Then, I would have needed another function to read all the information and line by line (manually) re-initialize each attribute of every object. Since my program can have a theoretically infinite amount of customer, lane and reservation objects, this process would take a long time and hundreds of lines of code. By using binary files and pickle I automated all that process.

Files:

As mentioned before, the program handles binary files. Attached to the repository are three example files that should facilitate testing:

- empty.b = empty file to check error handling if no BA object is loaded by the main function
- saved_file.b = file containing a BA-object with a pre-registered customer (email: test@gmail.com, password: test_test) and reservation (date: 06/05/2022, time: 14)
- saved_file_2.b = file containing a BA-object with no pre-registrations

If necessary, any extra files for further testing could be created with the create_alley.py directory, like previously explained.

NOTE: Remember to change the filename of the file being opened by the main function (line 263)!

Testing:

Thorough testing has been done for this project. During the coding itself, the program was constantly tested by trying different input-cases and checking whether or not the correct information was being processed and saved. Once the coding was done, a third party tested the code multiple times trying to break it.

It is noticeable on the code that there is a lot of error handling when it comes to incorrect user input. Throughout the entire main.py directory the user is forced to input correctly (using a lot of while-loops and try/except), in order to ensure the program will not crash. This was achieved with exhausting testing of numerous possible inputs.

Another important aspect of the testing process was checking whether or not the valid-expired qualities of the reservations were working properly. As previously mentioned, the reservation system is done so that the user cannot reserve a lane if all lanes for that specific time are occupied. And so, a useful testing note is: set a low count of lanes when creating a BA-object on create_alley.py, so that it is more easily checked whether or not all lanes have been booked for a specific time.

The known shortcomings and flaws in the program:

1- SHORTCOMING: The program only checks and changes the validity of each reservation once (at the beginning) per run. This means that if a reservation expires (its reserved time passes) while the program is still running, it will still be marked/counted as valid. This problem disappears every time the program is re-run. If I were to continue the project, I could fix this problem by implementing time-checks more recurrently throughout the program.

2- FLAW: The program crashes if you try to load a non-binary file.

3- SHORTCOMING: It is hard to “exit” the program midway. If the user begins an action (e.g.: “sign up” or “cancel reservation”), they cannot give up. Every action must be pursued until completion. While not a problem per se, it can be quite annoying if you accidentally start an action. This could be fixed by adding more if-conditions and connections between the main and the other functions on the main.py directory.

4- SHORTCOMING: There is no way to edit user information. While the Customer class has the necessary methods for editing Customer attributes, I did not have time to implement such options on the main program.

5- FLAW/SHORTCOMING: When saving the BA-object to a .text file, I get the following announcement banner: “The file was loaded in a wrong encoding: ‘UTF-8’”. I have exhaustedly researched its meaning and how to fix it, but I could not figure it out. I noticed this might cause some problems when looking at the “saved_file.b” and “saved_file_2.b” from the Git repository. However, given any problems, these files can be easily created following the instructions on create_alley.py.

6- SHORTCOMING: Reservations can only be made for exact hours (e.g.: 14:00:00).

Best and worst areas:

I believe the best/strongest part of the project is the wrong input error handling. The whole program is carefully done so that it evaluates every user input and uses while-loops to force the user to input correctly. Examples of these include checking if the input is an integer when it has to be, not allowing the input of an invalid date (e.g.: 30th of February), and not allowing the registering of an already-registered email.

Another strong part of my project is the program structure. The relationship between classes is clear and facilitates understanding how the program works. The functions on main.py directory are equivalent to the functions in GUI_windows.py, which I believe also help simplifying the structure of the program.

One of the weakest parts of my program is the lack of flexibility. The program is not easily quittable (you cannot just give up) and you cannot change any Customer-object information. These are extra features that would have improved the program quality. I believe that if I had had more time, I could have easily implemented them.

The weakest part of my program is the graphical interface. I have very little experience with it; therefore, it was hard for me to think of “the best way” to implement it. I thought it would be smart to create separate windows, but that ended up increasing the complexity of the code when it came to integrating the program onto the GUI windows. Throughout the project I learned a lot about creating graphical interfaces in python, however I believe I planned more than I could handle given my previous knowledge on the topic. I should have focused on simpler ideas and built a more solid background experience.

Changes to the original plan:

There have been multiple changes from the original plan. I did not have a very realistic idea for this project when planning. It was not easy to stick to my original schedule, due to parts taking a lot more time than I thought was needed. Multiple changes came from the graphical interface part, as I was not able to implement it.

I also had to change some classes (add and remove attributes and methods) as I realized some were missing or left over. I also removed classes from my plan all together (such as Files-class and Writefiles-class) as I noticed they were not practical to implement and use.

I also had less time to work on the project as I realized, so I had to prioritize and compromise features and ideas I originally had for the project. For example, I could not implement the Extras class (extras.py).

Realized scheduled:

I separated the coding into parts:

- coding the structure and basis all classes
- coding the graphical interface windows
- coding the main function
- finalizing classes (relations and methods)
- debugging main function

It was useful to frequently work on different parts whenever I got stuck or did not know what to do. That helped me increase productivity and efficiency.

Assessment of the final result:

Overall, I am proud of what I have accomplished with the project. Even though there was a lot on my plans I could not accomplish (i.e., the graphical interface), I believe I learned a lot from the process and managed to create a useful and almost completely error-free program.

Given that this was my first project, I believe one of the weakest aspects was the planning. Not knowing what I am capable of or any time-frame template, made me have to re-plan and reconsider things constantly. I had to re create the plan at least 4 times throughout the project. However, I believe my self-evaluating abilities are now sharper and I will have an easier time making a realistic and efficient plan on future projects.

References:

3.10.2 Documentation. (n.d.). Python Documentation. Retrieved February 24, 2022, from https://docs.python.org/3/

Qt for Python — Qt for Python. (n.d.). Pqty5 Documentation. Retrieved February 24, 2022, from https://doc.qt.io/qtforpython/