Enrique Munoz

1/29/2023

CS 3331 – Advanced Object-Oriented Programming – Spring 2023

Dr Mejia

PA0 MinerAir

This work was done individually and completely on my own. I did not share, reproduce, or alter any part of this assignment for any purpose. I did not share code, upload this assignment online in any form, or view/received/modified code written from anyone else. All deliverables were produced entirely on my own. This assignment is part of an academic course at The University of Texas at El Paso and a grade will be assigned for the work I produced.

Program Explanation

The objective of this assignment was to create a "Flight" class that represented a flight that included an origin and destination. The "Flight" class needed to include the attributes of a flight as well as methods to set and obtain these attributes. Afterwards, we needed to read a file to set a list of flights that a user could interact with and contain a log file that has every attribute change the user commited. Being able to see any information about the flight attributes or commit any changes to any attribute. I tackled the problem by first creating the attributes of the "Flight" class and then began with the setters. The setters were a little more problematic especially in the destination time and date setters since the arrival date and time also relied on these two. In order to solve this problem I created a method called "timeChange" and "dayChange" to successfully update these methods when changed the destination time and date. I broke down the problem by first solving the general solution of a change in time then adding the edge cases after the solution worked with simple cases. Creating the user and computer interaction menu was mostly syntax-based errors. Finding documents about file reading helped me solve these syntax errors.

What did I learn?

I learned more about the interactions of a class within itself. Especially in the "setDestinationDate" and "setDestinationTime", both of these methods had to interact with one another in order to correctly update both of them. Using the "timeChange" and "dayChange" methods where I needed to use the getter methods of the destination date and time as parameters. Allowing me to see understand how a class's methods can interact with itself. I feel like the other methods aren't very improvable since they are just getters and print methods but the time and date changing methods are able to be more efficient. Finding a way to know the number of days passed without returning an array of strings could improve the "timeChange" method in order to have a space of O(1) instead of O(N) with the creation of the array. To fix this, I could instead

compare the arrival time and destination time and see if the destination time of PM changed to AM showing that a new day has passed. Usually, flights shouldn't last more than 24 hours but an edge case would be a flight lasting more than that. Two days would pass but it would only take into account one day or none if it's at the same time of day. I took me about 6 hours to complete this lab assignment, with the time methods taking up most of the time.

Solution Design

Creating the flight method and its attributes was fairly simple, all I needed to do was create the following attributes given in the class. I set the attributes to private as I would later create getter and setter methods if the user wanted to change or obtain any of the attributes. The only method that was a bit different was the "setDestinationDate" and "setDestinationTime" methods. Unlike the other methods, the "arrivalTime" and "arrivalDate" depended on these methods, meaning that when updating the destination time, the arrival time would also have to be changed. In order to tackle this problem, I had to access the "arrivalTime" and "arrivalDate" in the "setDestinationTime" and "setDestinationDate" methods. I created my own method called "timeChange" and "dayChange" in where the "dayChange" method depended on the "timeChange" method. The "timeChange" method took in the duration as a parameter and the destination time attribute. Converting the duration into hours and minutes and adding that time to the destination time in order to obtain our new arrival time. The method also had to correctly determine if the new time was PM or AM meaning that at 11:58 PM with a duration of 94 minutes, our new time would be 1:32 AM and the destination date would change by one day, which is the reason for the method returning an array of strings. The array would return the new arrival time and a variable called "daysPassed" that would determine the number of new days that are meant to be added to the destination date. After, the destination date would take in the value of the variable at the index of the "daysPassed" in the returned string array and change the date depending on the number of days that were passed. The method "dayChange" would also change the month and year to its corresponding date if the day is over 31 or the month exceeds 12. This was probably the most difficult method in the flight class as time could change the arrival date meaning that I had to know the changed time and the number of days that passed during that time change. These methods contained a variety of edge cases so I took solved the general problem first and then changed the method depending on each edge case. Accounting for each edge case, in the beginning, would have been much harder because of the number of edge cases and with that, the main problem might have not have been solved. Past this point, creating the print method that outputs every attribute was much simpler. With the completion of the "Flight" class, came the interaction between the program and the user. I had to create an array list of flights to contain all the flights within "FlightSchedulePA0.csv" and another 2-dimensional array list in order to use the indices of every row as attributes for a new "Flight" object. Using a buff reader to read the csv file and input each line as a new "Flight" object with each line in the "lines" array list. I included a terminal menu so that the user could interact with each flight, allowing him to search for any flight ID, and interact with the flights attributes. Allowing them to change and look for any of the attributes. If the user changed any of the

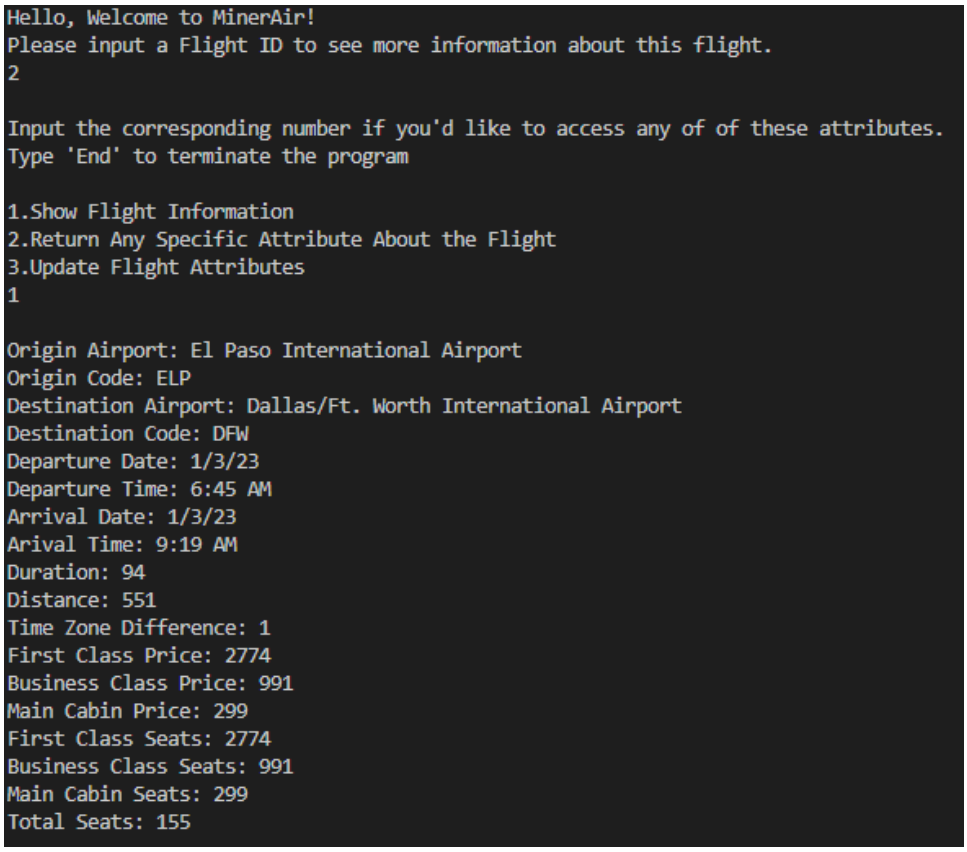
attributes, the program contained a log text file that would be updated with the user changes and reset every time the program is restarted.

Testing

I tested my program by using both black-box and white-box testing. In testing the terminal menu, I mostly used a black-box testing method by trying every given input and observing if the output was correct. Also testing to see if a users input was incorrect, they were given other chances to change their input. Taking only the outputs into account to see if the menu terminal was working correctly. I kept using black-box testing to test out the “Flight” class as well. Initially creating new objects and using the setters and getters to see if the class was working correctly. If the output of the methods were correct then the class was working successfully. The only methods I tested using the white-box method were the “setDestinationTime” and “setDestinationDate” methods since they incorporated the day and time changing methods I added. I had to carefully look at the code of the “timeChange” method especially to account for every edge case using print statements in every step of the algorithm to observe that the change was correct. The “dayChange” method also had the same steps taken but not as much since the “daysPassed” variable already gave the number of days that passed, so I had to check if the conversion of months and years was done correctly. I tested my solutions enough for the program to work optimally but a way to improve my test cases would be to include a different file of test cases to fully document each case that was tested. The test cases I used were wrong input user inputs and changing the departure time and date, as well as duration. These methods contained the most room for errors, especially with times like 11:58 AM changing to a new day and staying to PM at a departure time of 12:00 PM. I broke my program especially in the menu terminal by providing wrong inputs in order to handle these inputs gracefully by giving the user the same response until he chooses an input that works.

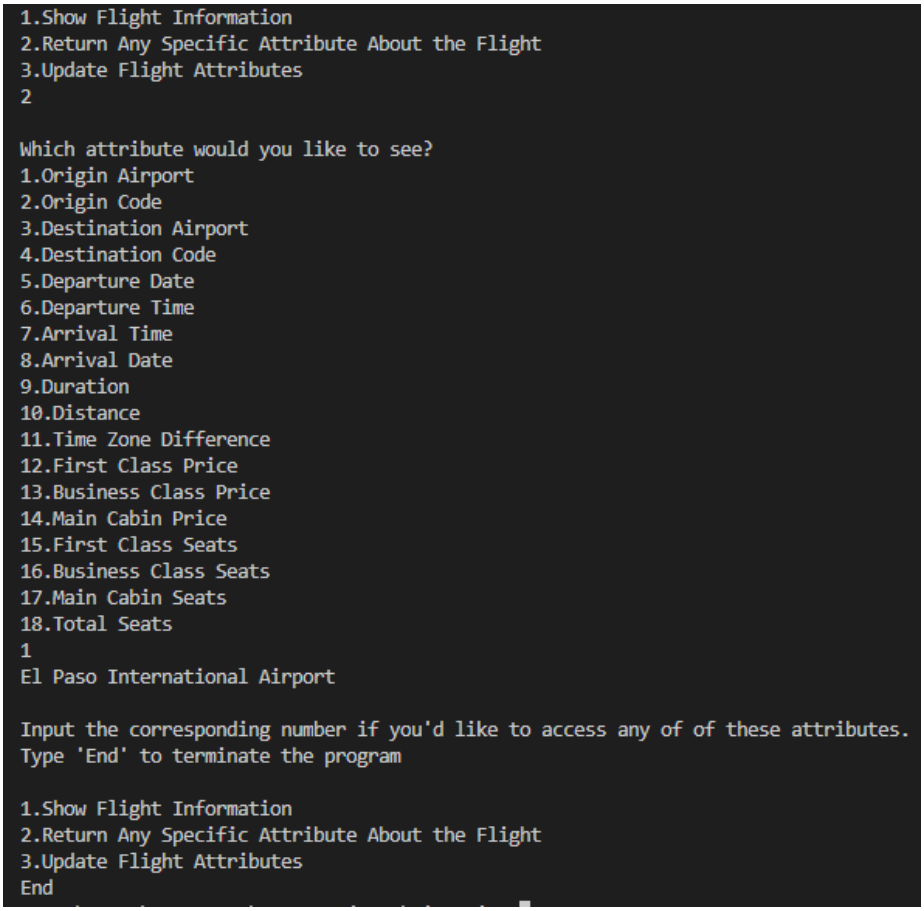
Test Results

Test of User Correct Inputs



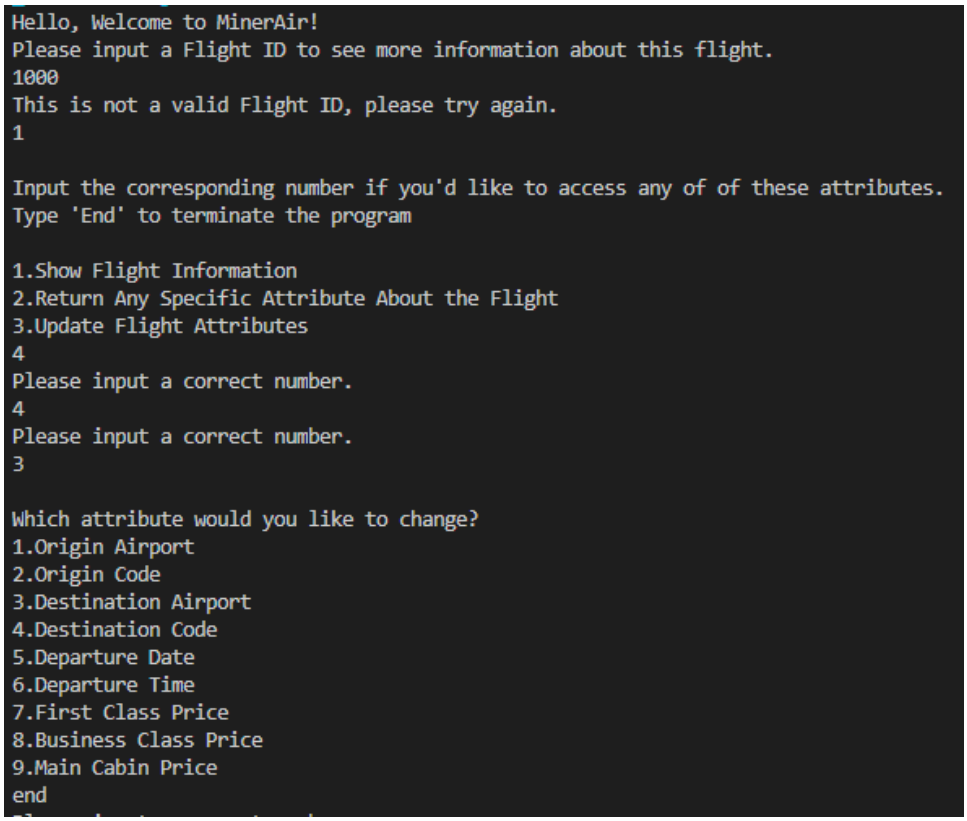
- The user here uses correct inputs, and the program successfully gives out the correct output of each user input.

Test of Inputs with Termination



- In this example, the user also incorporates the right inputs and each input gives a correct output showing that the methods of the “Flight” class are working correctly. The termination of the program is also demonstrated.

Incorrect User Inputs



- This example demonstrates that the program can handle incorrect user inputs, allowing them to try again until they have the correct input or end the program.

Time Changing

```
7
1:09 PM

Input the corresponding number if you'd like to access any of of these attributes.
Type 'End' to terminate the program

1.Show Flight Information
2.Return Any Specific Attribute About the Flight
3.Update Flight Attributes
3

Which attribute would you like to change?
1.Origin Airport
2.Origin Code
3.Destination Airport
4.Destination Code
5.Departure Date
6.Departure Time
7.First Class Price
8.Business Class Price
9.Main Cabin Price
6
What would you like to change the Departure Time to?
2:31 PM

Input the corresponding number if you'd like to access any of of these attributes.
Type 'End' to terminate the program

1.Show Flight Information
2.Return Any Specific Attribute About the Flight
3.Update Flight Attributes
2

Which attribute would you like to see?
1.Origin Airport
2.Origin Code
3.Destination Airport
4.Destination Code
5.Departure Date
6.Departure Time
7.Arrival Time
8.Arrival Date
9.Duration
10.Distance
11.Time Zone Difference
12.First Class Price
13.Business Class Price
14.Main Cabin Price
15.First Class Seats
16.Business Class Seats
17.Main Cabin Seats
18.Total Seats
7
4:05 PM
```

- The departure time is initially set to 1:09 PM but the user decides to change it to 2:31 PM. The time change method is demonstrated and works correctly as the arrival time changes to 4:05 PM. Which is a 94 minute difference between the departure time and the arrival time.

Code Reviews

I conducted a review of my code by checking every instruction individually. Initially creating a flight object and first checked if the setter methods were working correctly and used the getter methods to confirm. With these two-working hand in hand, the print method worked correctly. I tested the CSV File reader by printing the array list named “lines” to see if each line of the csv file was inserted correctly and did the same method in reviewing the Flight array list. I observed the log text file after the program was terminated and checked if every input was appended

correctly. The readability of the code was clear, containing well named variables and methods. I included comments and documentation to provide more readability to my code. The user input and outputs is also handled well as seen in the test results and the program also included every functionality that was meant to be added. Reading the file correctly and storing it in an array list, allowing the user to access all the attributes of a flight. The inputs of the user are also handled gracefully allowing them to give responses with the correct inputs or allowing them to terminate the program. Each change is also documented in the log text file at the end of the program. The uses of object-oriented programming are used correctly.