Gustavo Rubio
Date: April 4, 2023
CS 3331 – Advanced Object-Oriented Programming – Spring 2023
Instructor: Dr. Daniel Mejia Assignment: PA4

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.

NOTE: **Please write in complete sentences (paragraph form).** Check grammar, punctuation, and ensure your writing is clear. Write enough to make sure you clearly explain each part of the question.

1. Program Explanation

During this assignment we were given a scenario where we have been hired by *MinerAir* to create a system that could store and change the information of a flight given a fight ID number. The system must extract information from a given file and let the user change the following: origin airport, origin code, destination airport, destination code, departure date, departure time, first class price, business class price, and main cabin price. It is important to note that whenever the system changes the departure date or time, it must also change the arrival date or time of the specific flight. The system must also keep track of any changes that were made and submit a report after the user has finished updating the information. In addition, we were now tasked to implement a customer purchasing system and allow the employee/manager to make changes into the flights as well.

In this assignment I had to create new methods to support the new purchasing system and keep the original changes from the last assignment. The new changes consisted of adding an auto purchasing system so that the employee can use to purchase “test tickets”. The new feature would allow the employee to view how many customers have bought tickets, as well as the total revenue of the flights. This auto buyer is meant to simulate the backend of the entire system and see how the system would react when a large amount of people start using the system. In addition to the auto buyer, I added a new for the customer to search for their flight, now they can search by flight id, flight number and by origin/destination codes. These new search functions would allow the customer to still find their desired flight with more ways to search for the results.

The way that I broke down the problem was by first creating the new search algorithms for the customer to find their desired flight, to find a flight by the origin/destination codes the system will ask the user to put in both the origin and the destination codes that way the system can show all the flights that fit with those dates. The method that I created to do this is called flightsByCodes() and the way that it works is buy receiving both codes and then it will linearly go through the flight schedule HashMap. Then the method will store all of the flights that match

the codes in a new HashMap and return the new list. If the new HashMap has a length of zero the system will assume that the codes do not exist and the customer will have to put in new codes.

For the auto buyer it was almost that same implementation that I used for my customerPurchaseOptions() method because the auto buyer should programmatically do the same thing just without the prints statements. The auto buyer must will handle all of the exceptions as the customerPurchaseOptions() method, however instead of asking for inputs the system must perform everything programmatically. To do this I created the main auto buyer method along with a helper method. The main method was designed to read the auto files while the helper method was used to do the purchases based on the inputs that were given by the main method. After the method finishes reading the auto purchase file(s), the employee can now view, edit, and cancel flights with now having a large amount of data to use.

The techniques that I used to solve this problem was by first drawing a diagram to see what information was going to be needed for each class and how I wanted my program to look and execute. My design also had to reflect the assignments descriptions, which altered some ideas I had initially, but it did not make the task harder. Instead, I had to improve on those ideas and ask questions about certain imports and assumptions to see my ideas correlated with the assignment. After clarification I started the design process and broke down the problems into helper methods for better access/programming.

2. What did I learn?

During this assignment I learned how to properly manage and store information even when they program would terminate. In the past whenever I would run my program it would perform the tasks as intended however it lacked a way to store information even when the program would terminate. This later lead to complications whenever the customer would try to cancel a ticket because there was no record of the customer buying a ticket in the first place. The way that I was able to fix this was by creating a new csv file for storing customers tickets, and it would store all the information from the ticket object that way the system can refer to it. I did the same thing with the employee role, but for whenever the employee wants to cancel a flight they system would log the flight id and the new status of the flight. After multiple attempts I was able to improve my system by now having a history log for all changes that happen while the system is active.

3. Solution Design

In this assignment the only major changes that we had to do was create a new search option for the customer, create an auto buyer for the employee, and create a ticket summery option to view the purchase history of a customer. The first method that I created was the new search option for the customer, this new function now allows the customer to search for their flight by origin and destination codes. The way that this works is by once the system verifies the given codes, the method will have to linearly search through the flight schedule log and check if the given codes match with the current flight that it is on. If there is a match then it add that flight into a new

HashMap, if there's no match then it will go to the next flight, once the method finish going through the flight log it will return the new HashMap with the flights that match the codes. When this method finishes executing it will print all the flights that match the codes and then the system will ask the customer to enter the flight id that they want and continue as normal for the purchase options.

For the auto buyer it was a similar method the customerPurchaseOptions() method that I have created; however, instead of asking the user for inputs the method programmatically does the inputs. The way that I did this was by creating two methods one of them is responsible for gathering information while the other was responsible for doing the purchases. Based on the information that the first method gathers while reading the file, it will pass on those attributes as inputs for the helper method. The helper method has the exact same implementation as the customerPurchaseOptions(), the only difference between the two is that the auto buyer does not display any messages and that all inputs do not need to be verified since the first method is in charge of handling those exceptions. For the ticket summary to work a customer needs to buy tickets or else there is no summary, for this option to work it has to be after the auto buyer gets executed or when a normal customer buys a ticket. Once any of the two options happened the employee can select the view ticket summary option and the system will ask the employee to enter a name. If the name exists, then the method will go thought every flight and check if the current customer bought any tickets in that flight, if it did then it will write the information in a separate file, if the not purchases were made then it will go to the next flight.

4. Testing

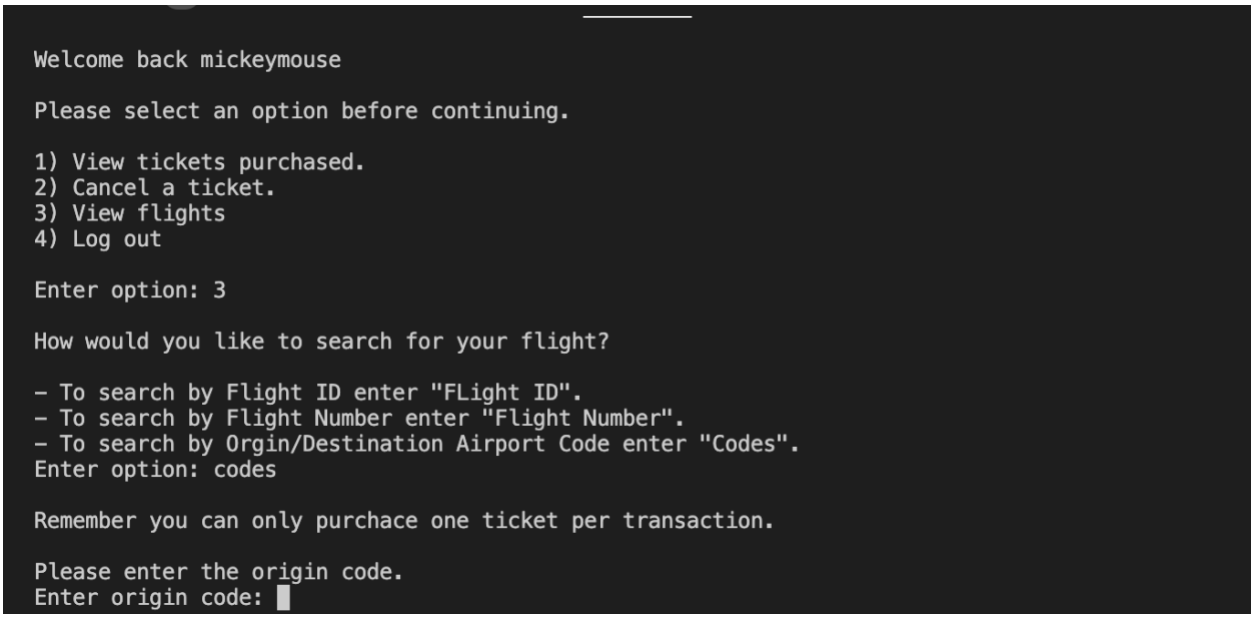
Once I had an almost completed version of my code, I decided to do white box testing to improve upon the system that I had created. The first thing that I did was make sure that all method worked properly, once all methos worked I moved on into my other test cases. The first thing that I did was make sure that all exceptions were handled properly, one of my classmates noticed a couple of exceptions where not handled properly and that they needed to be fixed to prevent the system from crashing. This involved me going through all the errors that my classmate has noticed and ensuring that they were patched, I also had to ensure that all of my other user inputs were handled properly.

After handling all exceptions, a I had to make sure that my auto buyer was performing the right discounts, ticket purchases, and handle customers that did not have enough funds. I was having trouble writing the proper information onto the ticketHistoryPurchase file because I had to reference the amount of money from the customer log and make sure that it gves a reasonable output for the flight. I also had to refactor one of my object methods since it was not handling situations where customers entered zero or more than what is allowed to buy. Then I had to make sure that the information what was being passed on onto the method was correct, for example if the customer wanted to buy a first-class ticket, the system had to pass on the correct option number so that it executes the proper purchase. Once the auto buyer worked properly I successfully manages to go through all three auto purchase files that were provided with no errors.

Another test case that I did was making sure that the user would not get stuck in an infante while loop because there would be times where the system will do unexpected outputs due to the condition. For each while loop that I did I had to make sure that the user could get out of the loop by either entering the correct input or by entering multiple wrong answers and still exiting out. I was able to fix this by tracing each while loops and putting in print statements to ensure that the code was doing what was indented to do. Some of these mistakes where done by not resetting my verifyInput variable. Without the reset the conditions were never meet which either caused the system to skip over certain questions or simply not allowing the system to exit out of the loops. After multiple testing, I was able to ensure that all while loops are working correctly and that the system did not give the customer access to the employ methods.

5. Test results

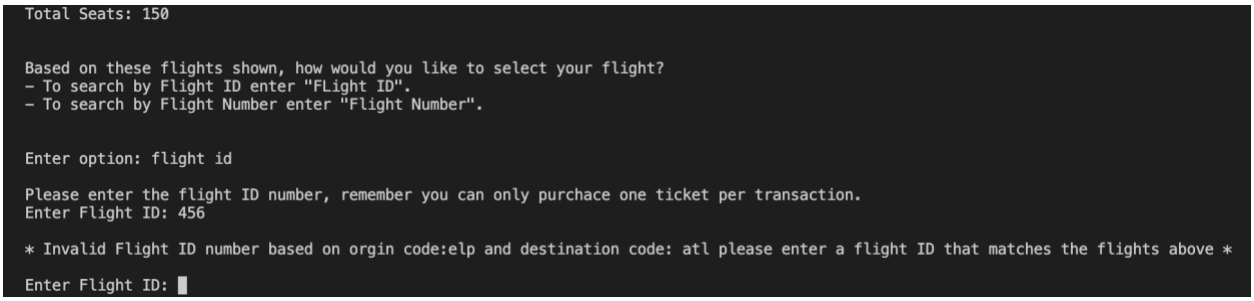
Test case 1: The system now has new functions for the customer to search for their flight, int his test case the customer will buy a ticket based on the origin/destination codes.



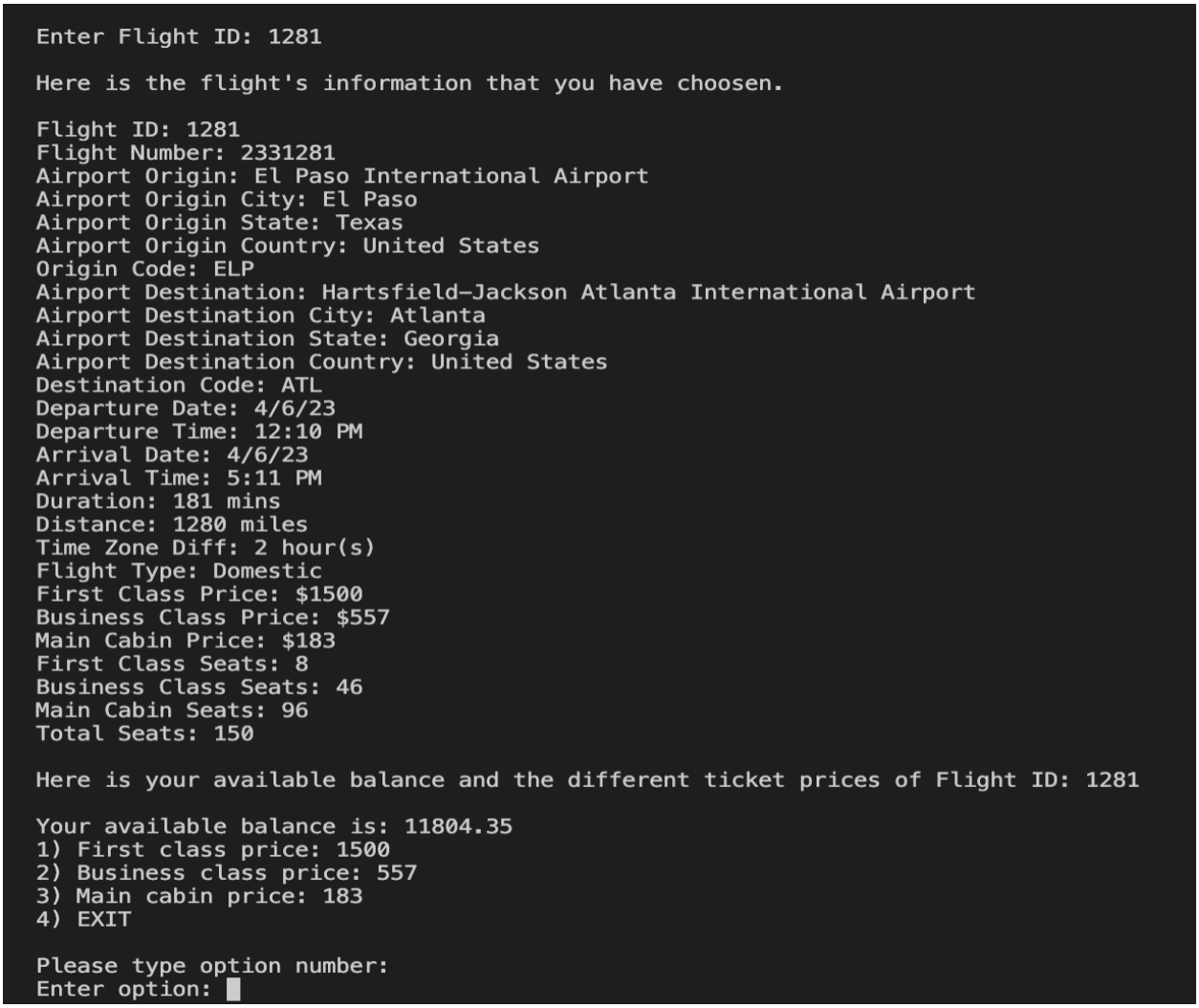
Customer entered ELP for the origin code and ATL for the destination code.

```
Airport Origin State: Texas
Airport Origin Country: United States
Origin Code: ELP
Airport Destination: Hartsfield-Jackson Atlanta International Airport
Airport Destination City: Atlanta
Airport Destination State: Georgia
Airport Destination Country: United States
Destination Code: ATL
Departure Date: 4/6/23
Departure Time: 8:25 AM
Arrival Date: 4/6/23
Arrival Time: 1:26 PM
Duration: 181 mins
Distance: 1280 miles
Time Zone Diff: 2 hour(s)
Flight Type: Domestic
First Class Price: $3310
Business Class Price: $931
Main Cabin Price: $349
First Class Seats: 11
Business Class Seats: 48
Main Cabin Seats: 88
Total Seats: 147

Flight ID: 1281
Flight Number: 2331281
Airport Origin: El Paso International Airport
Airport Origin City: El Paso
Airport Origin State: Texas
Airport Origin Country: United States
Origin Code: ELP
Airport Destination: Hartsfield-Jackson Atlanta International Airport
Airport Destination City: Atlanta
Airport Destination State: Georgia
Airport Destination Country: United States
Destination Code: ATL
Departure Date: 4/6/23
Departure Time: 12:10 PM
Arrival Date: 4/6/23
Arrival Time: 5:11 PM
Duration: 181 mins
Distance: 1280 miles
Time Zone Diff: 2 hour(s)
Flight Type: Domestic
First Class Price: $1500
Business Class Price: $557
Main Cabin Price: $183
First Class Seats: 8
Business Class Seats: 46
Main Cabin Seats: 96
Total Seats: 150


Based on these flights shown, how would you like to select your flight?
- To search by Flight ID enter "FLight ID".
- To search by Flight Number enter "Flight Number".


Enter option:
```

In this example the customer entered a flight ID number that does not exist with codes ELP and ATL for the origin and destination codes.



In this example the customer entered a flight ID number that does exist with codes ELP and ATL for the origin and destination codes.



Once the flight is displayed the customer can continue with the rest of the program like normal.

Test case 2: When an employee logs in to the system this is the new employee main menu that they would see. The new options are the auto-purchase test and ticket summary, in this test case we will look at option 3.

```
Welcome back gustavorubio

Our system noticed that your username is in our employee list.

Please select an option below.

1) Make flight changes by flight ID
2) View all airports' status by code
3) Auto-purchase Test
4) Customer ticket summary
5) Purchase tickets (As customer)
Enter option:
```

After the employee selects option 3 the system will display this message foe the auto purchaser.

```
1) Make flight changes by flight ID
2) View all airports' status by code
3) Auto-purchase Test
4) Customer ticket summary
5) Purchase tickets (As customer)
Enter option: 3

Processing 10k customer file...
Program successfully finished 10k customers!

Would you like to go back to the employee main menu? [Y/N]
Enter option:
```

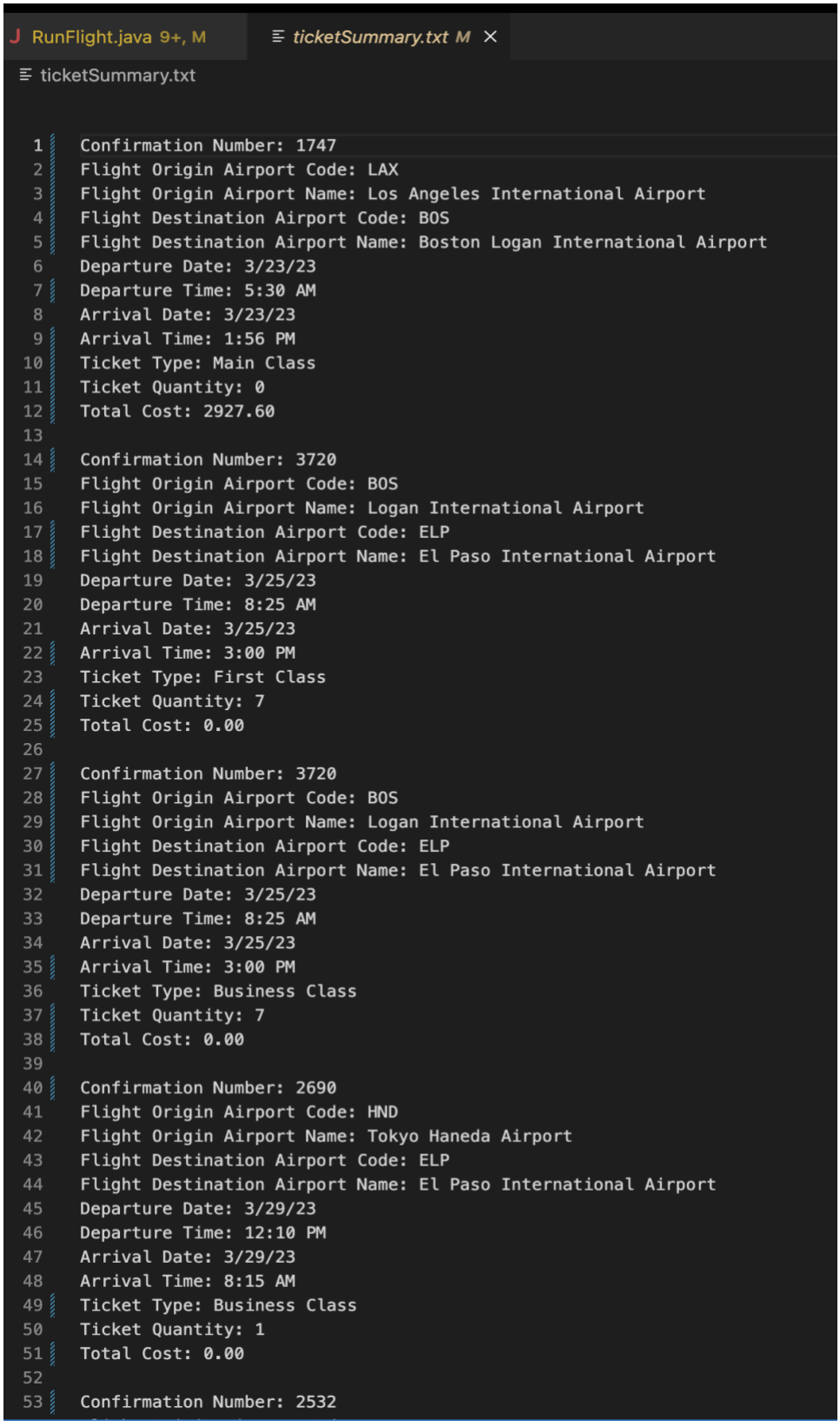
This screen shot shows the result of the auto purchase system since it also logs the purchases.

```
Changes made while purchesing a ticket:
User (mickeymouse) has exited out of the program.
Changes made while in auto purchese mode:
User (laurencoleman) purchased a business class ticket(s) for Flight ID: 36
User (laurencoleman) bought 5 tickets.
User (laurencoleman) was charged $9.15 for the transaction fee.
User (laurencoleman) was charged $28.00 in security fee(s).
User (laurencoleman) saved a total of $3020.00
The ticket cost is $755 per seat.
Flight ID: 36 has a surcharge of $0
Business class now has 5 seats available.
Flight ID: 36 now has 148 total seats.
Comfitmation number is: 4815

Changes made while in auto purchese mode:
User (heatherrobbins) purchased a main cabin class ticket(s) for Flight ID: 10
User (heatherrobbins) bought 7 tickets.
User (heatherrobbins) was charged $9.15 for the transaction fee.
User (heatherrobbins) was charged $39.20 in security fee(s).
User (heatherrobbins) saved a total of $1355.20
The ticket cost is $242 per seat.
Flight ID: 102 has a surcharge of $0
First class now has 97 seats available.
Flight ID: 102 now has 167 total seats.
Comfitmation number is: 3266

Changes made while in auto purchese mode:
User (davidharvey) purchased a business class ticket(s) for Flight ID: 578
User (davidharvey) bought 7 tickets.
User (davidharvey) was charged $9.15 for the transaction fee.
User (davidharvey) was charged $39.20 in security fee(s).
User (davidharvey) saved a total of $5801.25
The ticket cost is $1105 per seat.
Flight ID: 578 has a surcharge of $0
Business class now has 7 seats available.
Flight ID: 578 now has 181 total seats.
Comfitmation number is: 3438

Changes made while in auto purchese mode:
User (douglasmckay) purchased a main cabin class ticket(s) for Flight ID: 1321
User (douglasmckay) bought 6 tickets.
User (douglasmckay) was charged $9.15 for the transaction fee.
User (douglasmckay) was charged $33.60 in security fee(s).
User (douglasmckay) saved a total of $1867.20
The ticket cost is $389 per seat.
Flight ID: 1321 has a surcharge of $0
First class now has 92 seats available.
Flight ID: 1321 now has 161 total seats.
Comfitmation number is: 4768
```

Test case 3: After the employee runs the auto purchase the employee can see a ticket summary by entering the name of the customer and then the system will display a summary of all the tickets they bought.

```
Please select an option below.

1) Make flight changes by flight ID
2) View all airports' status by code
3) Auto-purchase Test
4) Customer ticket summary
5) Purchase tickets (As customer)
Enter option: 4

Please enter a customer's first and last name.
First Name: Mickey
Last Name: Mouse

File has been created.

Would you like to go back to the employee main menu? [Y/N]
Enter option:
```

Proof of Mickey Mouse's ticket summary.



6. Code Review

Explain how you conducted a review of your code. Describe how you checked each part of the code review checklist.
Implementation:
• Does this code do what it is supposed to?

Yes, the program can do the necessary changes based on the given options. • Can it be simplified?

Yes, there was a function that had a lot of procedures to make so a way the solve this is by making multiple helper methods to easy the function. However, based on my knowledge this was my best solution.

• Is the code dynamic or hardcoded?
The program can work dynamically since it is capable of checking if the user-inputs were valid.

• Is the code maintainable?
Yes, they system can run and execute smoothly and proved the necessary results.

Logic:
• Cases where code does not behave as expected/intended?

In the beginning I had multiple issues regarding on how the scanner would take in inputs based on the data type. The way that I manage to change this was by using integer partitions instead of changed from string to integer and vis versa.

• Test cases where it may fail? Readability/Style
For the most part the code was reliable it did not break during the many of the test cases. When there would be an error, I make sure to find out what is causing the error and how I can improve on it.

• Easy to read/understand?
Yes, the code is easy to understand, and I made sure to add comments where it was needed.

• What parts can be modified or adjusted?
My customerPurchaseOptions method is lengthy, although it is capable of performing the right outputs, it can be hard to read since there are many procedures what are being made.

• Is the structure appropriate?
Yes, using an array is appropriate since an array can store multiple objects. Once accessing the information, I can do the necessary changes within that object instead of overwriting all of the other objects.

• Does it follow the appropriate language style?
Yes, the program runs in java and it follows the correct syntax for the language.

• Is the code well documented?
Yes, the program has comments where they describe what the code does.

Performance:
• What is the code complexity?

The overall program runs in O(1) whenever it ask for the user inputs and when it is printing out the inputs. The only time where time complexity changes is when the program is traversing through the *FlightSchedule* and *CustomerListPA4* the time complexity is O(n).

• How does the complexity change with various inputs?
The file writer and all inputs run in O(1) time, the only change is when the program needs to read the file which is in O(n) time.