Team Name: Extreme Software Development Team LLC
Date: Nov. 19, 2023
CS 3331 – Advanced Object-Oriented Programming – Fall 2023
Instructor: Dr. Meija
Assignment: Project 5

This work was done within our team. We did not share, reproduce, or alter any part of this assignment for any purpose. We 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 our 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.

1. Program Explanation

For project 5 we made sure that all of project 4's functionality was correct. We then were tasked with adding a service fee, charity, and convenience fee as well as keeping a running total for each event of these fees. We implemented the discount feature as well. The customer is now able to cancel a ticket and will be reimbursed their money to only include the subtotal no fees will be returned. We also added functionality for the manager/admin to be able to cancel an event, this will reimburse all customers their money to include subtotal and fees. The admin can also view all information for that event, total fees, total taxes, etc.

We tackled the problem by following our UML, it provided a visual way of what needed to get done and what was already done in code. It was very effective when we were doing the lab and anything that was added or changed in the code was updated in the UML to reflect that change. We collaborated via the live share extension in vs code. It allowed us to work in real-time and see what was happening with each other's code changes. Also doing virtual meetings to collaborate boosted the workflow of getting this lab done. We did break the problems into smaller ones. We would all work on the same method and share ideas and collaborate to get the job done. For instance, when calculating the fees and total, we would each calculate the total in our calculator and share the results and see if they match the output results from the program.

2. What did I learn?

We learned how to refactor our code to implement the tasks above. There wasn't a whole lot we had to do. However, after talking with Ali we did decide, based on his suggestion, to use hashmaps instead of a linked list. It was not hard to change since the linked list is just a data structure that allowed us to access the NodeData at a given time. That data structure could be anything, like the hashmap allowing us to retrieve information more efficiently. This created a constant time lookup instead of linear. We also incorporated the ability for the customer to search for either their name, id, or password/username in constant time. Also, we frequently update to a new csv file the event and customer in case something goes wrong in the program any changes prior to that issue will not be lost.

We think that our solutions at the moment were refactored to the best of our abilities. Our solutions have high cohesion and loose coupling ensuring that the code does not have a lot of dependencies. We collaborated to see how we can improve the code and from there ensured the code was functional and could get the job done. If we could not think of something it was left as it is. We will greatly appreciate any feedback obtained for this lab and will incorporate and

refactor if needed to make the code as best as possible. Collaborating via live share allowed us to solve complex problems in a timely manner, like writing code, fixing bugs, sharing ideas and if one idea of solving a problem was better than the other, we incorporated it. Overall, the lab did not take long to complete. It had little new functionality to add and once added it was fixing and refining bugs found during testing. Overall, it was about a total of 10-13 hours spanning a week and a half.

3. Solution Design

After chatting with Ali we decided to switch to hashmaps. This has created a constant time look up for customer and event. As for adding onto fees and cancellation, our classes contained information that made it simple to return a customer's money depending on if the event was canceled or if the customer canceled.

We collaborated together to get the job done. Its problem was carefully analyzed and collaborated on and implemented in code. So, we took Ali's advice by using the hashmap to store the (id, object) and another hashmap to store the (name, id) rather than use the linked list like we did in PA4. It was brought up that the linked list is good, but with retrieving information the time complexity is not the best we could achieve because at worst case it would take O(n). We still used NodeData object since we did not want to change the code. The only thing we were changing is how we access that NodeData object because the linked list was just a data structure to traverse and get that data, it can easily be done with another data structure. So, this data structure hashmap is better since it is inserting, accessing and deleting in constant time. We still used a hashmap for the customer, but also added a hashmap for storing the (full name, id) pair and another for storing the (username/password, id) pair. This allowed for inserting, accessing and deleting in constant time. We used ArrayList to store the string invoices of transactions made by customers. We used an ArrayList to store the Ticket object of a purchase made by customer. This is the only type of data structure we used throughout the program. These data structures are appropriate since the time complexity is constant and the space complexity is not a lot. We did not make any assumptions on user input. We handle input mismatch exception thrown, handle null pointer exceptions thrown. We log all exceptions that are potentially thrown during the program to see how we can fix it and improve. The only thing that would cause the program to fail is if there is no event or customer list to begin with, but overall, we have good code, and we will improve our code with any feedback given.

4. Testing

During this assignment we updated the test cases without the use of the linked list class because we used a Hash Map to replace the class. When testing we utilized the white box testing strategy for our program. This helped us to fix some of the minute bugs within our code base. We also added two new test cases. The first new test case was the test method of a customer purchasing a ticket, and the second newest test case was the customer action of canceling a ticket. Both test cases helped us know if we are returning the correct amount of money based on whether a customer cancels or an administrator cancels the event. If we didn't do the calculations, then we wouldn't have figured out that the discount price was not being calculated to the total. Any other errors or exceptions encountered were tackled and resolved, for instance, the issues while canceling a ticket because we were unsure of where the canceled ticket was being stored. Not only that, but there were issues while using our program normally, like a user input wouldn't do what we programmed it to do, so we went back to the code and fixed it

afterwards. Lastly, we demoed in class with a group and showed each other our terminals, while it was short, they were able to give us feedback on making our program more user friendly. We also gave them feedback as we encountered a few errors, like purchasing a ticket or logging in feature.

5. Test results

In Figure 1 you can see the results of our TestSuite. All green check marks makes an engineer happy.

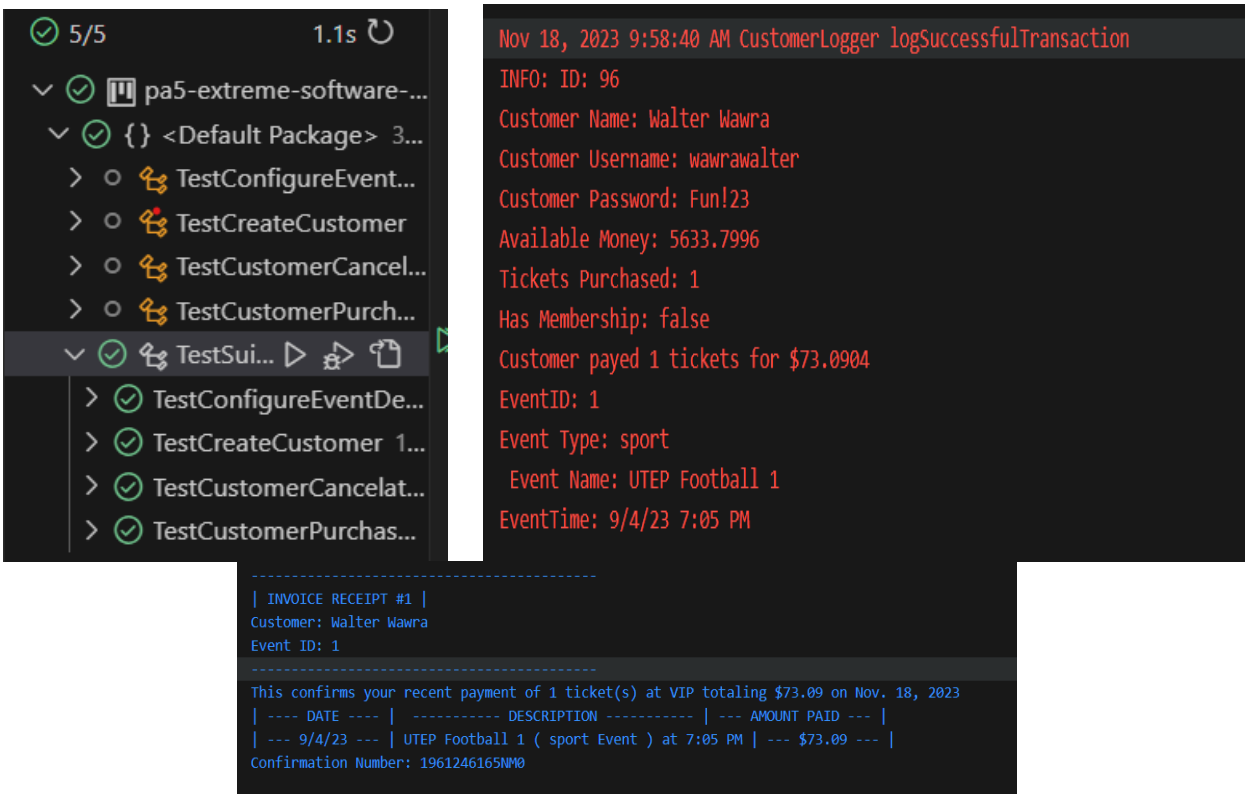

Figure 1. Results of TestSuite.

Below in Figure 2, you will see the customer purchasing a ticket. This ticket costs $82.88. With fees + taxes you can see that the ticket cost is $94.61.

```
Please enter valid event ID between (1-72) to search
>> 45

Event ID: 45
Event Name: UTEP Wind Symphony 1
Event Type: concert
Date: 10/10/23
Time: 6:00 PM
Vip price: $82.88
Gold price: $54.83
Silver price: $48.45
Bronze price: $31.88
General admission price: $12.75
Fireworks planned: No
Total cost: 13536.0
Please enter the ticket type you want to purchase

1. VIP
2. GOLD
3. SILVER
4. BRONZE
5. GA - GENERAL ADMISSION
6: No longer want to buy ticket(s)
>> 1
How many tickets (1 - 6):
>> 1
-----------------------------------------
| INVOICE RECEIPT #1 |
Customer: Walter Wawra
Event ID: 45
-----------------------------------------
This confirms your recent payment of 1 ticket(s) at Vip totaling $94.61 on Nov. 18, 2023
| ---- DATE ---- |  ----------- DESCRIPTION ----------- | --- AMOUNT PAID --- |
| --- 10/10/23 --- | UTEP Wind Symphony 1 ( concert Event ) at 6:00 PM | --- $94.61 --- |
Confirmation Number: 1006552631NM0

Do you want to search again? (Yes/No)
>>
```

Figure 2. Customer purchasing a ticket and receiving the correct total based on discount, fees, and taxes.

Below in Figure 3, you can see the customer canceling that ticket. The customer will then be reimbursed which you can see is correctly implemented by viewing the above TestSuite results in Figure 1.

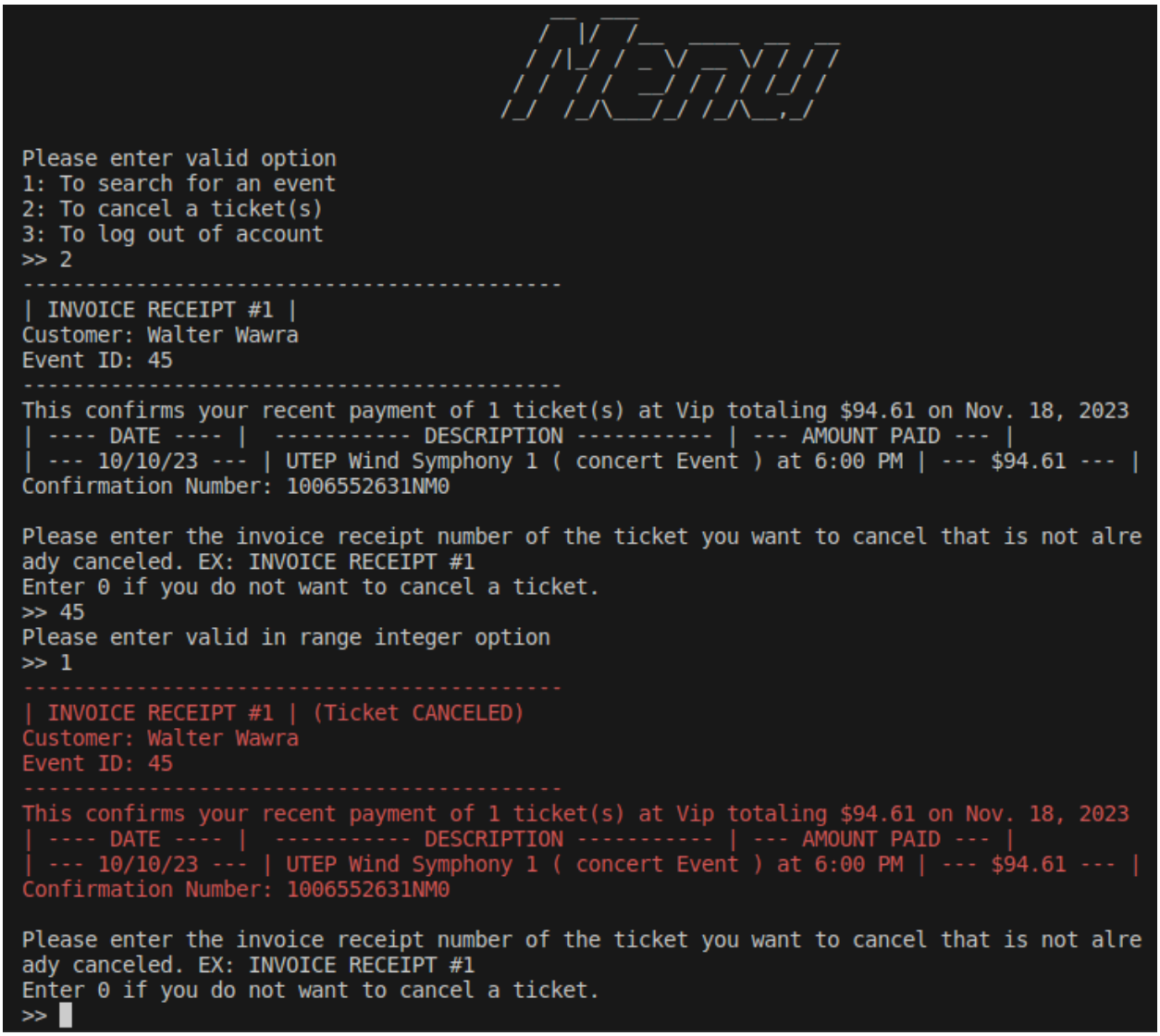

Figure 3. The result of a customer canceling a ticket.

Before we had correctly implemented the TestSuite we also used print statements to debug our code and ensure that the correct reimbursement was being made depending on whether the customer canceled or an administrator canceled the event. Below in Figure 4 & 5, you can see multiple purchases to one event, 13.

```
-----------------------------------------
| INVOICE RECEIPT #1 |
Customer: Raul Pallares
Event ID: 13
-----------------------------------------
This confirms your recent payment of 3 ticket(s) at Vip totaling $193.39
on Nov. 18, 2023
| ---- DATE ---- |  ----------- DESCRIPTION ----------- | --- AMOUNT PAID
--- |
| --- 11/27/23 --- | UTEP Football 13 ( sport Event ) at 7:05 PM | ---
$193.39 --- |
Confirmation Number: 1400229130IM0
```

Figure 4. Raul purchasing a ticket for event ID 13

```
-----------------------------------------
| INVOICE RECEIPT #1 |
Customer: Walter Wawra
Event ID: 13
-----------------------------------------
This confirms your recent payment of 2 ticket(s) at Vip totaling $143.46
on Nov. 18, 2023
| ---- DATE ---- |  ----------- DESCRIPTION ----------- | --- AMOUNT PAID
--- |
| --- 11/27/23 --- | UTEP Football 13 ( sport Event ) at 7:05 PM | ---
$143.46 --- |
Confirmation Number: 534353239NM0
```

Figure 5. Walter purchasing a ticket for event ID 13

Below in Figure 6, you can see the administrator viewing event ID 13 and their ability to view the total fees, individual fees, and other various analytics. In Figure 7, you can see that the administrator has canceled the ticket and can now not view that event as it is canceled.

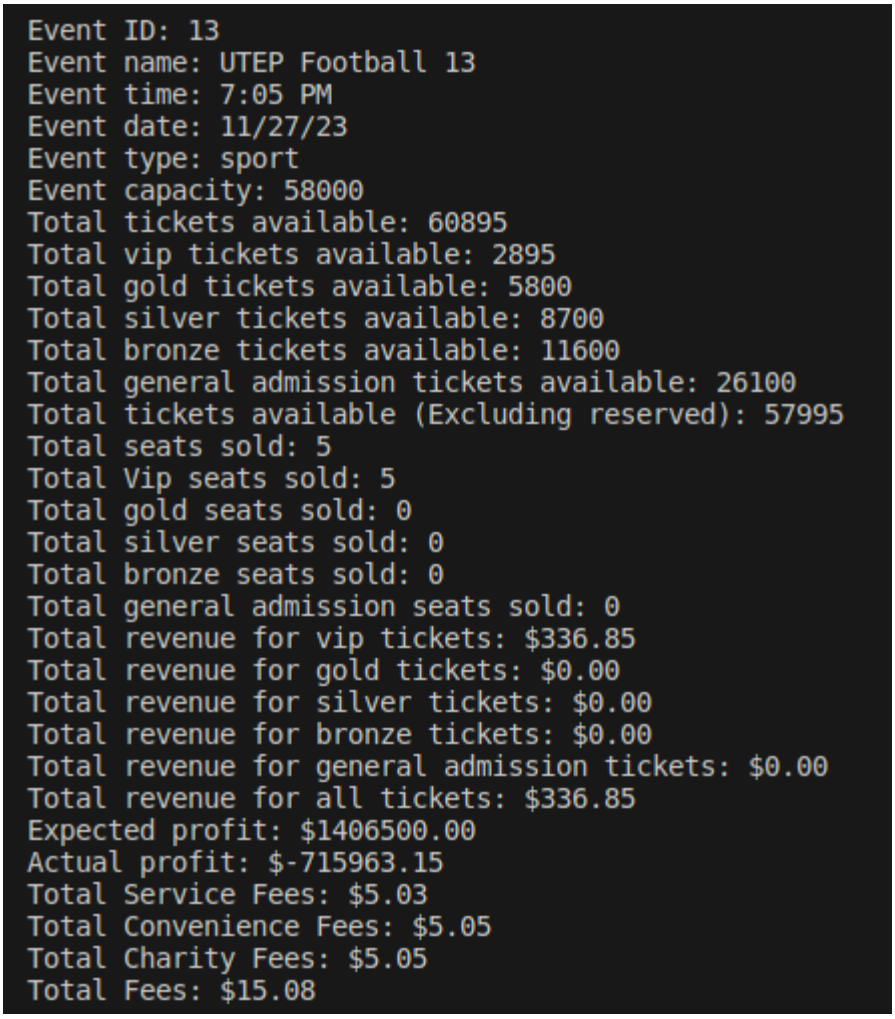

Figure 6. Admin viewing event 13’s analytics.

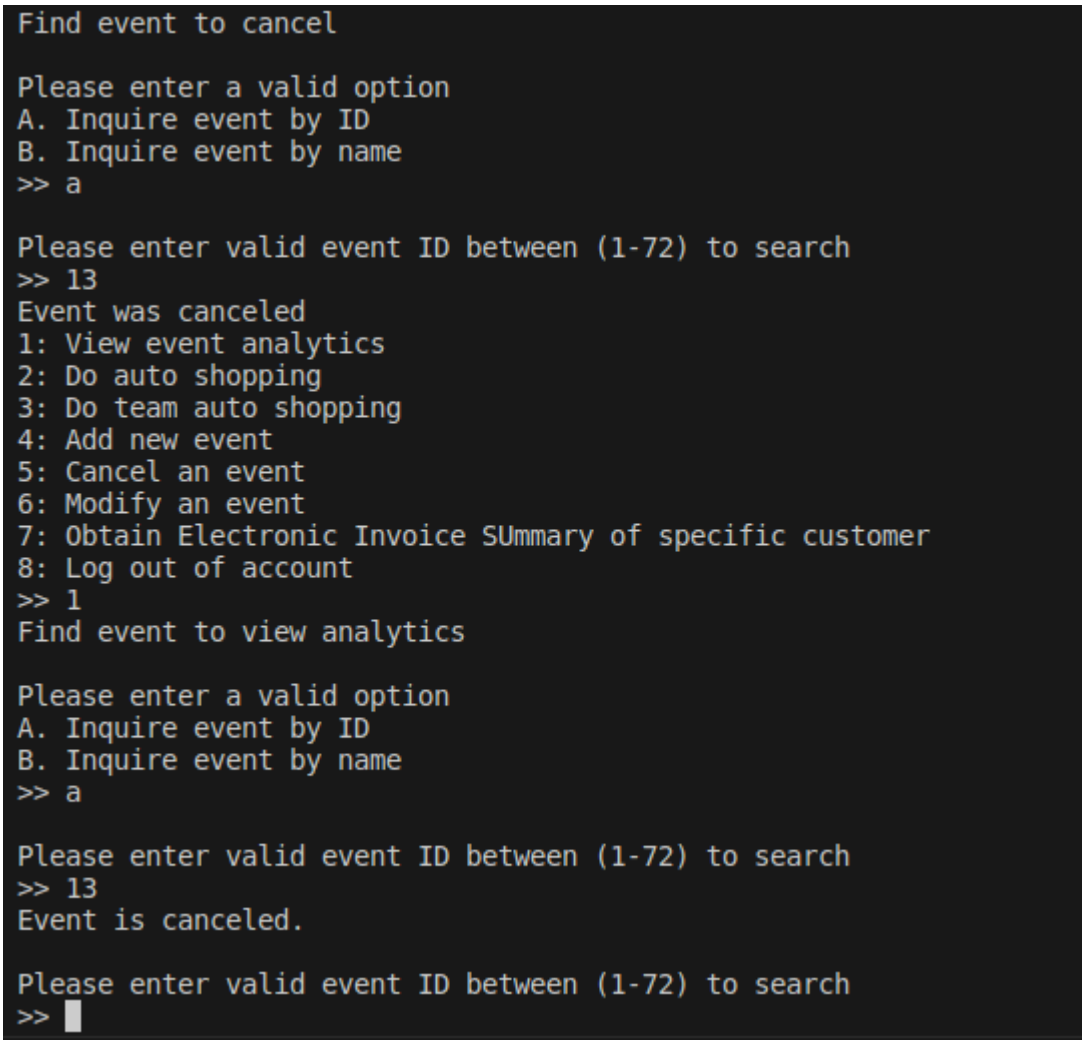

Figure 7. The Administrator can now see that the event was canceled.

6. Code Review

We were given a task to implement a ticket miner system of purchasing tickets/canceling a ticket as a customer and adding, removing, viewing, obtaining, and doing auto purchasing as an admin. All functionality that was specified in the PDF guidelines was included in this lab and all previous functionality works. The code does not do anything unexpected and performs the task. We refactored a lot in this lab to ensure that we adhered to the best code practices. Any feedback obtained in this lab will be greatly appreciated and will be incorporated in the code to make it simplified/modular. We ensured that the code was dynamic and therefore it is able to read in and store the data from excel files as well as the customer and event data appropriately. We updated the data to a new csv file dynamically as well for both customer and event. Any hard coded values are the ones we have control over during the program. For instance, prompting for user input we check for is in range of certain integer values and handle exception appropriately. Yes, the code is maintainable. It was refactored to the best of our combined abilities. It has high cohesion meaning every class and method has a purpose and every method in the class is related to that purpose. It has lost coupling meaning low dependency with each other. The classes and methods follow java conventions and are easy to understand based on the naming conventions and the comments provide. It is easy to change without something breaking from that change. If another person can understand the code, then it is maintainable.

The code behaves as it is expected to behave. We take every exception that could be thrown during the program execution and handle it. The code would not behave unexpectedly from user input and would not crash because of it. The only way the program would fail is if there was not csv files for customers or event within the folder. The test case that failed before was customers canceling a ticket because initially it was not returning the correct amount to the customer. From that we were able to fix that bug and now the test case works correctly and we are certain that the money is being returned. We tested the hashmap functionality for the event for searching by id or name to see if it worked and the customer hashmap for searching by id, full name, username/password to ensure that it worked. Overall, the test cases help improve and find bugs/errors in our code.

The code is easy to read, it follows java naming conventions and is organized. The attributes, classes and methods have appropriate and clear naming that tell what the method is intended to do. The methods and classes have a purpose to do. No method or class is redundant. The formatting and indented of the code make for clear readability. We refactored as best we could to provide good maintainable, modular code. We refactored every method or class that could have been refactored to improve efficiency and readability. Any feedback given will be incorporated into the code to improve the overall quality of the program. The data structures are important. The hashmap allows us to add, get, delete in constant time which is very good considering the linear time it took our old implementation to add, get, and delete. The space complexity is good, we use two hashmaps one for storing the (id, object) part and the other for storing the (name, id) pair, so if we search by name, we get the id and use that id to get the object. Same thing for customers since we also using a hashmap. We store (id, object) pair, another hashmap is implemented for storing (full name, id) pair and another for storing (username/password, id) pair. So, if we search by full name or username/password we get the id and use that id to obtain the customers object data. Yes, the code follows java language styles, it follows java naming and guidelines conventions. It is very well documented. It has good meaningful comments to show what code does and helps with readability. The java documentation is meaningful to provide what the code does to outside sources.

The code complexity for time complexity is good. When reading in the data from a csv file it takes O(n) to store the data in an arraylist and O(n) to store the data to the attributes. This is true for event and customer and auto purchase excel files. Accessing the data either from an event or the customer data is constant time as mentioned above we use hashmaps for storing data for the event and the customer. We use two hashmap one for storing the (id, object) part and the other for storing the (name, id) pair, so if we search by name, we get the id use that id to get the object. Same thing for customer since we also used a hasmap. We store (id, object) pair, another for storing (full name, id) pair and another for storing (username/password, id) pair. So, if we search by full name or username/password we get the id use that id to the customer object data. Since the only input is user input, the time complexity would not change that much. Every search and retrieving are done in constant time, every input is done in constant time.