Anaiah Quinn, Erik LaNeave, Michael Ike, Ian Gutierrez

November 26, 2023

Dr. Daniel Meija

Polish Assignment

This work was done individually and completely on our own. 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 we produced.

1. Program Explanation

This assignment aimed to create a system to facilitate selling tickets to events (sports games, festivals, and concerts). We made a command line interface program that imports event and customer information from CSV files and adds the ability for customers to purchase tickets. The program can also read a CSV file containing purchase information and automate said purchases. It also adds the ability for admins to find administrative information on events, as well as the ability to create new events from the console. Finally, the program was equipped with an electronic invoice summary feature. Information about the customers (money available, tickets purchased, etc.) and events (available seats, revenue, etc.) will be calculated and stored while the program runs; upon termination, all information will be written to new CSV files. This assignment required us to add new features (canceling events, canceling tickets, charging additional fees, and printing new reports). We worked on one piece of the program until completion and moved on to the next. This helped us focus on one problem at a time and led to less and easier debugging.

2. What did We learn?

As a result of this assignment, we learned about the importance of space complexity. Space complexity involves the amount of memory our program uses to perform algorithms and store information. Overall, the space complexity of our solution can be improved. This improvement can be made by storing some of the information in different ways to decrease heap utilization, and the amount of Java heap memory being used. We could also try moving information to a database. We managed our time well and started the assignment immediately. We started by breaking it into tasks, assigning them, and updating the use cases. We then added the new functionalities, updated our diagrams, and finished the use case scenarios.

3. Solution Design

In this program, we finalized the first deliverable of a ticket-purchasing interface designed for TicketMiner. The system operates as a ticket sales company specializing in various events, including sporting events, concerts, and special occasions. The interface facilitates various functionalities for customers, such as browsing events, accessing and saving their invoices, as well as purchasing and canceling tickets. For administrators, it offers capabilities like adding or checking events, running an automated ticket-purchasing tool, canceling events, computing company profits, and storing invoices for customers. Upon program termination, the system updates the Event and Customer list .csv files with the latest information. Additionally, it generates a summary of the program's operations, logging it into a .txt file. Furthermore, invoices are produced for each user who logged in during the session.

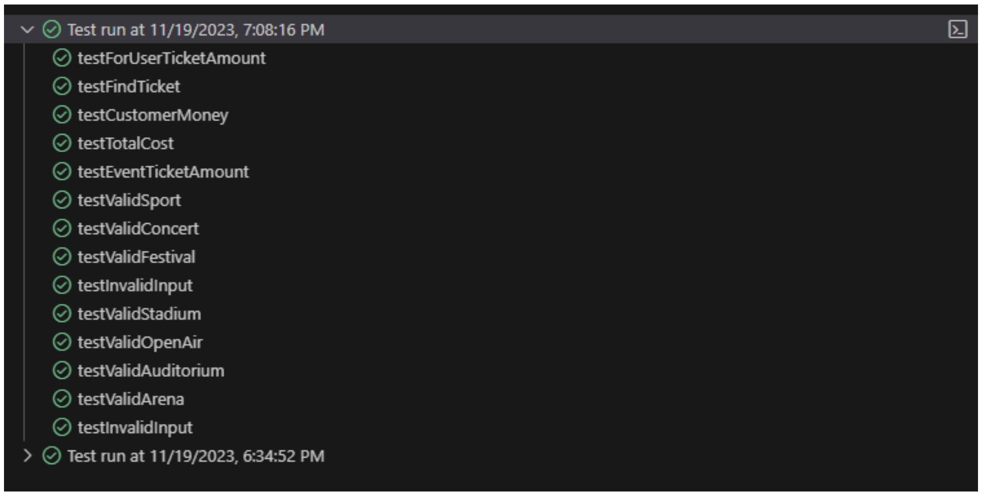
Our approach to addressing this challenge involved breaking down tasks to introduce three new functionalities to the program. Initially, we incorporated convenience, service, and charity fees into every purchase, monitoring the fee collection in both the Event and TicketMiner classes to generate revenue for TicketMiner. The next enhancement allowed users to choose from their list of ticket purchases and select which tickets they wished to cancel. Upon cancellation, customers received a refund for the ticket price, while the company retained the money earned from the fees. The canceled seats were then returned to the Event, becoming available for purchase again. For the final functionality, administrators gained the ability to cancel an entire event, refunding all the money to the customers along with the service fee. Administrators could also generate reports detailing the revenue gained by TicketMiner for any specific event or across all events. To conclude, we expanded the suite of JUnit test cases, refactored the code for improved clarity, added comprehensive Javadoc comments, and ensured the proper writing of data into .csv and .txt files.

We opted to use HashMaps for storing fees to enable swift and efficient retrieval based on unique event and customer IDs, as well as fee types. This decision was deliberate, as HashMaps facilitates rapid and direct access to fees and events in linear time. Their key-value pair structure allowed us to link unique identifiers (IDs) with specific fee types, ensuring efficient search and retrieval operations.

4. Testing

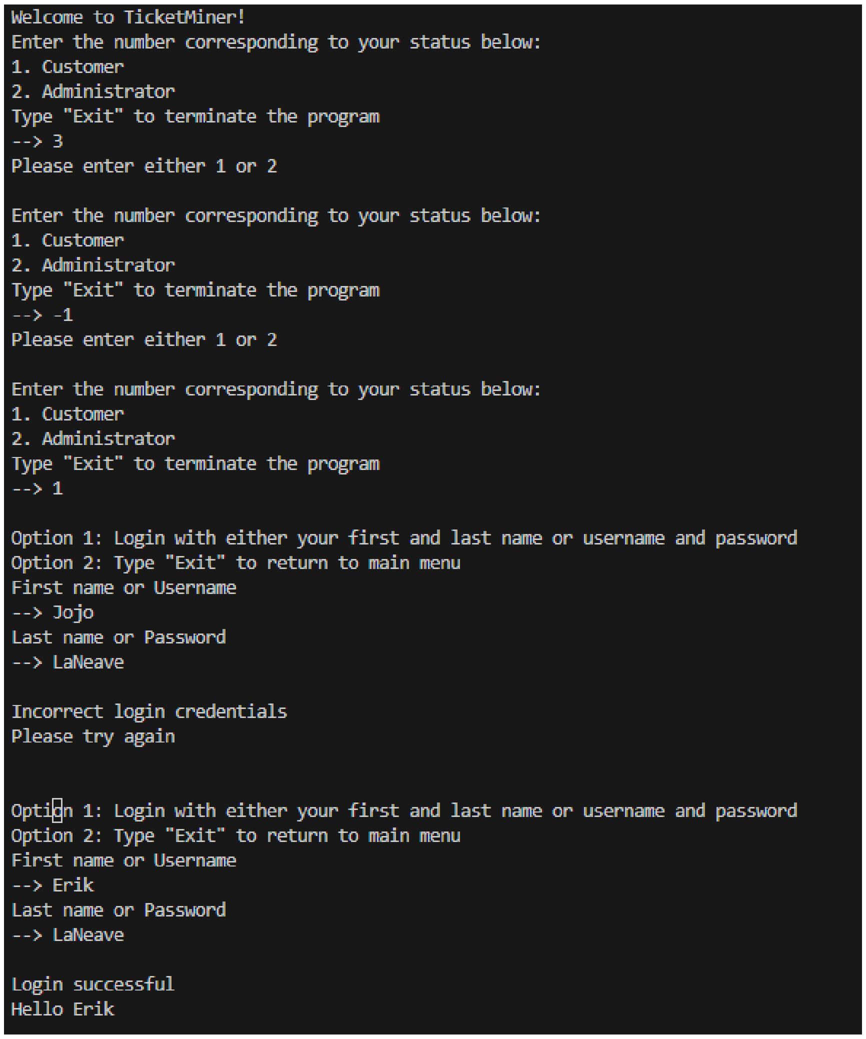
The program was tested using a combination of white-box and black-box testing. Both white and black box testing was used to test methods used to control the auto purchase for the admin, the purchase ticket option for the customer, the cancel a ticket option, the cancel an event option, the creation of a new event, and general checking that the program was updating correctly. These tests consisted of various random and incorrect inputs to prove that the program would keep running under any condition. The tests that were used to ensure that the program was updating correctly purchases were made and then canceled by the user or by the admin. The values were checked either at termination or through menu options to view ticket or event information. The auto purchase required a large amount of testing to get working in a timely and memory-efficient way. The testing required some trial and error to find what was best for the performance of the auto purchase because memory errors were occurring and not occurring depending on the machine that was being used, but the test consisted of running and rerunning the process multiple times to ensure that the code was efficient. This was possible because of previous testing on the purchase method which had been proven to be robust. White-box testing was used on the sub-methods that drive the purchase class to ensure that values are properly updated and the total cost amounts are correct. These tests were conducted with the JUnit framework and proved that the purchase methods can handle incorrect inputs, handle the proper exceptions, and perform the needed checks to ensure that a valid purchase is made. Overall the testing of the program could be improved with the use of more JUnit classes to test all the updating of the various Hashmaps and Arraylist but it seems to working fine at the moment. During all the testing multiple bugs and errors were found in the program and taken care. This was extremely helpful during this lab because of all the different small parts of logic that are needed to update and maintain the large amounts of hashmaps and ArrayList. When demoing with partners a black-box testing method was used because we were not able to see the source code. The various parts of the UI and how the program handles random and incorrect errors were tested and how the program updates with the admin and customer making changes. No errors were found in the testing and their program worked in a correct and normal way.

5. Test results



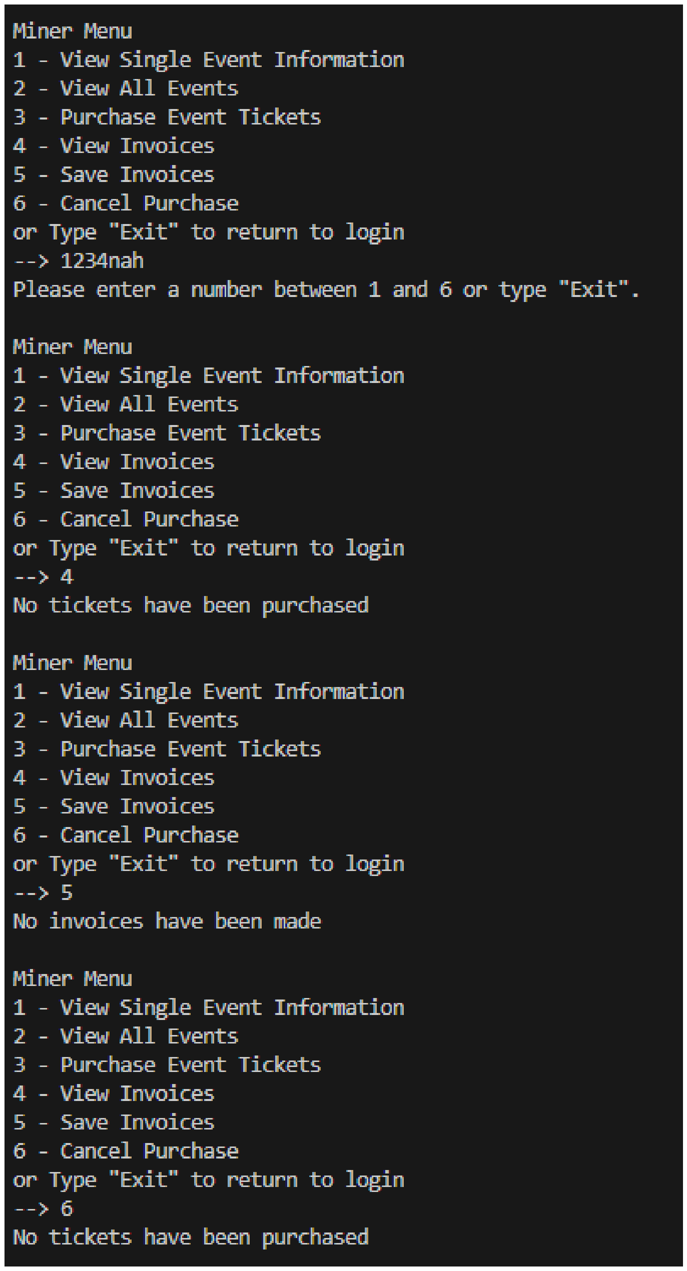
```
Test run at 11/19/2023, 7:08:16 PM
    testForUserTicketAmount
    testFindTicket
    testCustomerMoney
    testTotalCost
    testEventTicketAmount
    testValidSport
    testValidConcert
    testValidFestival
    testInvalidInput
    testValidStadium
    testValidOpenAir
    testValidAuditorium
    testValidArena
    testInvalidInput
Test run at 11/19/2023, 6:34:52 PM
```

The JUnit test above shows that the program does not error out and handles the given test cases.



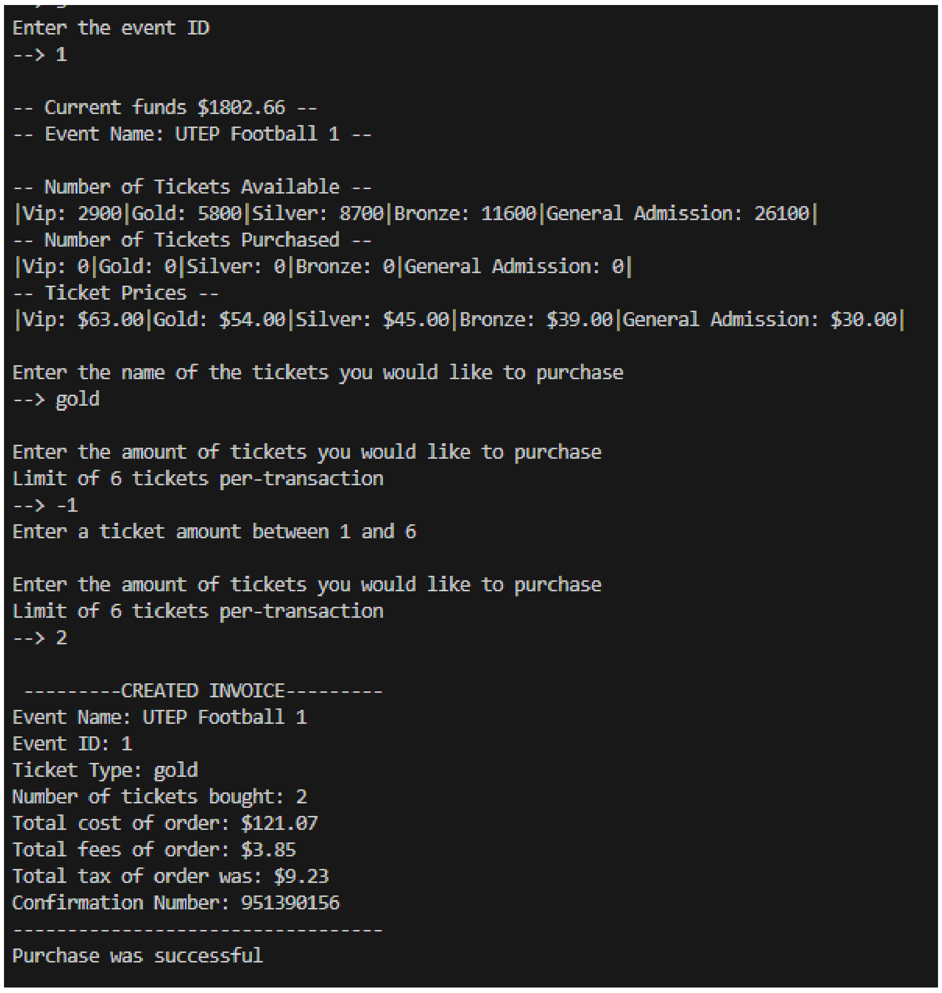
```
Welcome to TicketMiner!
Enter the number corresponding to your status below:
1. Customer
2. Administrator
Type "Exit" to terminate the program
--> 3
Please enter either 1 or 2

Enter the number corresponding to your status below:
1. Customer
2. Administrator
Type "Exit" to terminate the program
--> -1
Please enter either 1 or 2

Enter the number corresponding to your status below:
1. Customer
2. Administrator
Type "Exit" to terminate the program
--> 1

Option 1: Login with either your first and last name or username and password
Option 2: Type "Exit" to return to main menu
First name or Username
--> Jojo
Last name or Password
--> LaNeave

Incorrect login credentials
Please try again


Option 1: Login with either your first and last name or username and password
Option 2: Type "Exit" to return to main menu
First name or Username
--> Erik
Last name or Password
--> LaNeave

Login successful
Hello Erik
```

This test shows that the login can handle incorrect attempts and still let a valid user into the program.



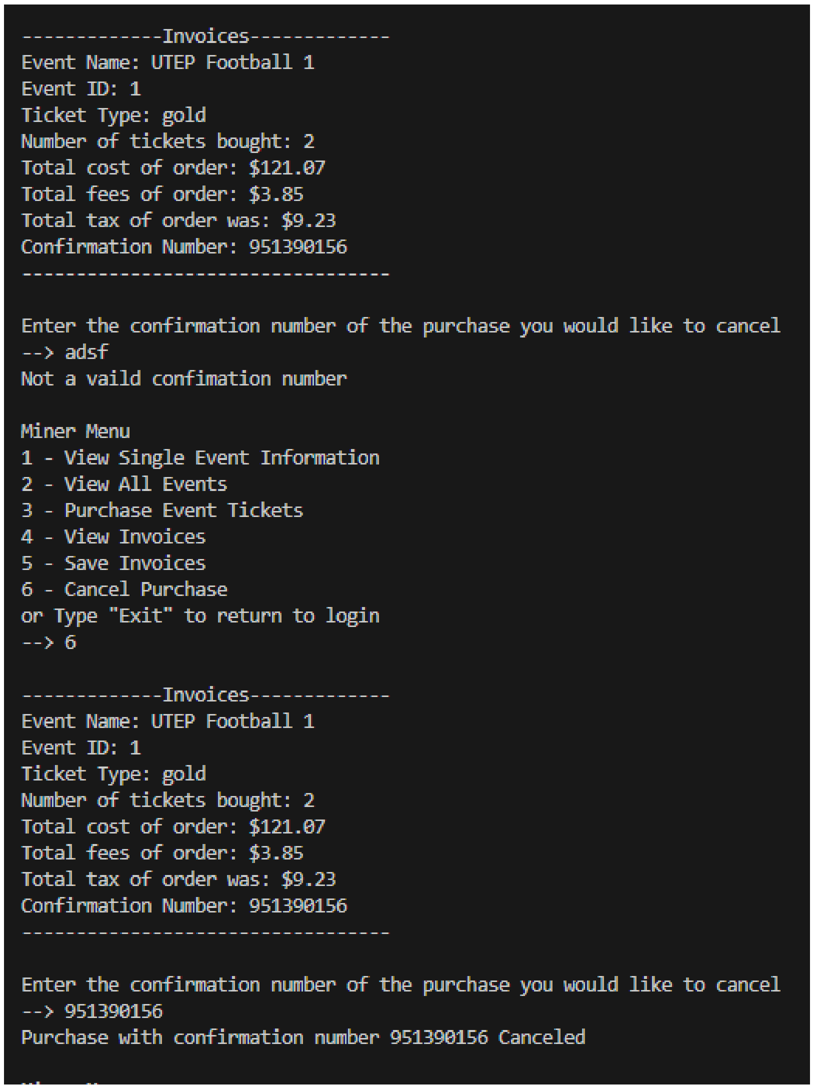
```
Miner Menu
1 - View Single Event Information
2 - View All Events
3 - Purchase Event Tickets
4 - View Invoices
5 - Save Invoices
6 - Cancel Purchase
or Type "Exit" to return to login
--> 1234nah
Please enter a number between 1 and 6 or type "Exit".

Miner Menu
1 - View Single Event Information
2 - View All Events
3 - Purchase Event Tickets
4 - View Invoices
5 - Save Invoices
6 - Cancel Purchase
or Type "Exit" to return to login
--> 4
No tickets have been purchased

Miner Menu
1 - View Single Event Information
2 - View All Events
3 - Purchase Event Tickets
4 - View Invoices
5 - Save Invoices
6 - Cancel Purchase
or Type "Exit" to return to login
--> 5
No invoices have been made

Miner Menu
1 - View Single Event Information
2 - View All Events
3 - Purchase Event Tickets
4 - View Invoices
5 - Save Invoices
6 - Cancel Purchase
or Type "Exit" to return to login
--> 6
No tickets have been purchased
```

This test demonstrates that the customer menu can handle incorrect inputs and invoice menu options won’t activate if the user has no invoices.



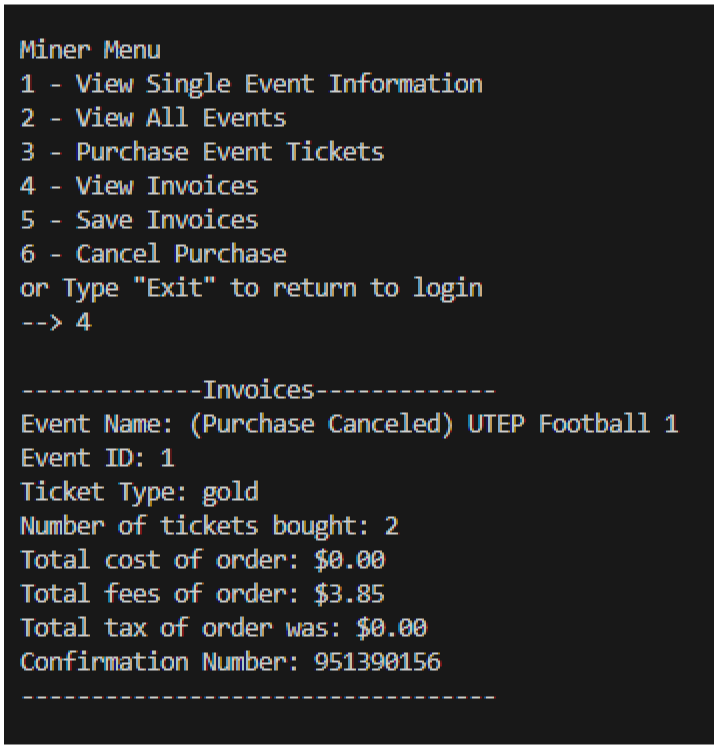
```
Enter the event ID
--> 1

-- Current funds $1802.66 --
-- Event Name: UTEP Football 1 --

-- Number of Tickets Available --
|Vip: 2900|Gold: 5800|Silver: 8700|Bronze: 11600|General Admission: 26100|
-- Number of Tickets Purchased --
|Vip: 0|Gold: 0|Silver: 0|Bronze: 0|General Admission: 0|
-- Ticket Prices --
|Vip: $63.00|Gold: $54.00|Silver: $45.00|Bronze: $39.00|General Admission: $30.00|

Enter the name of the tickets you would like to purchase
--> gold

Enter the amount of tickets you would like to purchase
Limit of 6 tickets per-transaction
--> -1
Enter a ticket amount between 1 and 6

Enter the amount of tickets you would like to purchase
Limit of 6 tickets per-transaction
--> 2

 ---------CREATED INVOICE---------
Event Name: UTEP Football 1
Event ID: 1
Ticket Type: gold
Number of tickets bought: 2
Total cost of order: $121.07
Total fees of order: $3.85
Total tax of order was: $9.23
Confirmation Number: 951390156
---------------------------------
Purchase was successful
```

The test above shows a successful run of the purchase menu option with some incorrect inputs from the user.



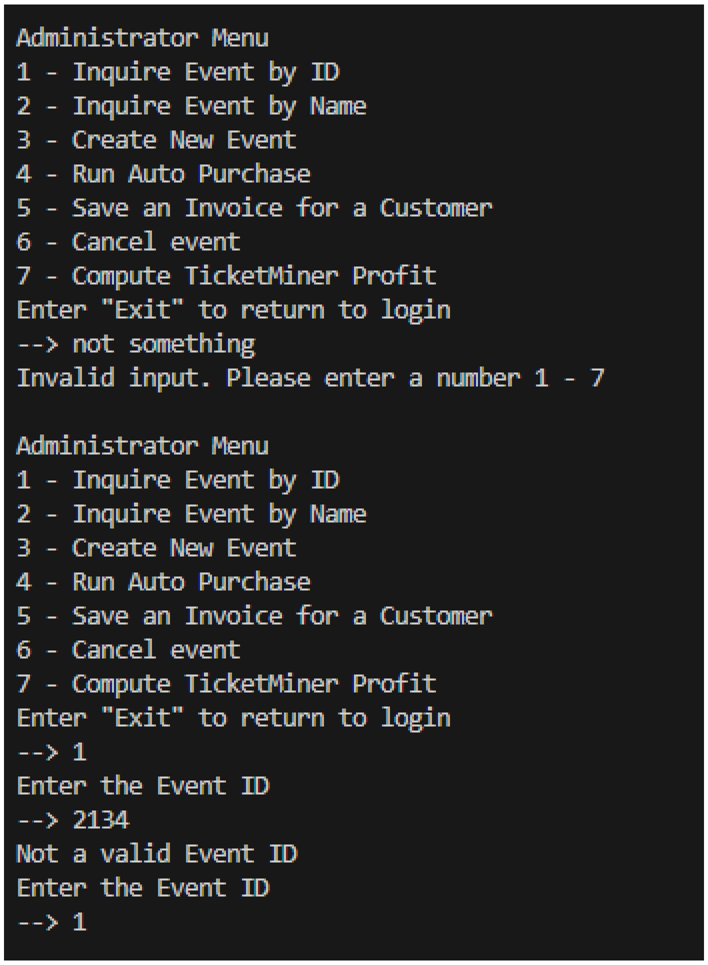
```
-------------Invoices-------------
Event Name: UTEP Football 1
Event ID: 1
Ticket Type: gold
Number of tickets bought: 2
Total cost of order: $121.07
Total fees of order: $3.85
Total tax of order was: $9.23
Confirmation Number: 951390156
----------------------------------

Enter the confirmation number of the purchase you would like to cancel
--> adsf
Not a vaild confimation number

Miner Menu
1 - View Single Event Information
2 - View All Events
3 - Purchase Event Tickets
4 - View Invoices
5 - Save Invoices
6 - Cancel Purchase
or Type "Exit" to return to login
--> 6

-------------Invoices-------------
Event Name: UTEP Football 1
Event ID: 1
Ticket Type: gold
Number of tickets bought: 2
Total cost of order: $121.07
Total fees of order: $3.85
Total tax of order was: $9.23
Confirmation Number: 951390156
----------------------------------

Enter the confirmation number of the purchase you would like to cancel
--> 951390156
Purchase with confirmation number 951390156 Canceled
```

The test above is a successful cancelation of an invoice showing that it can handle incorrect inputs.



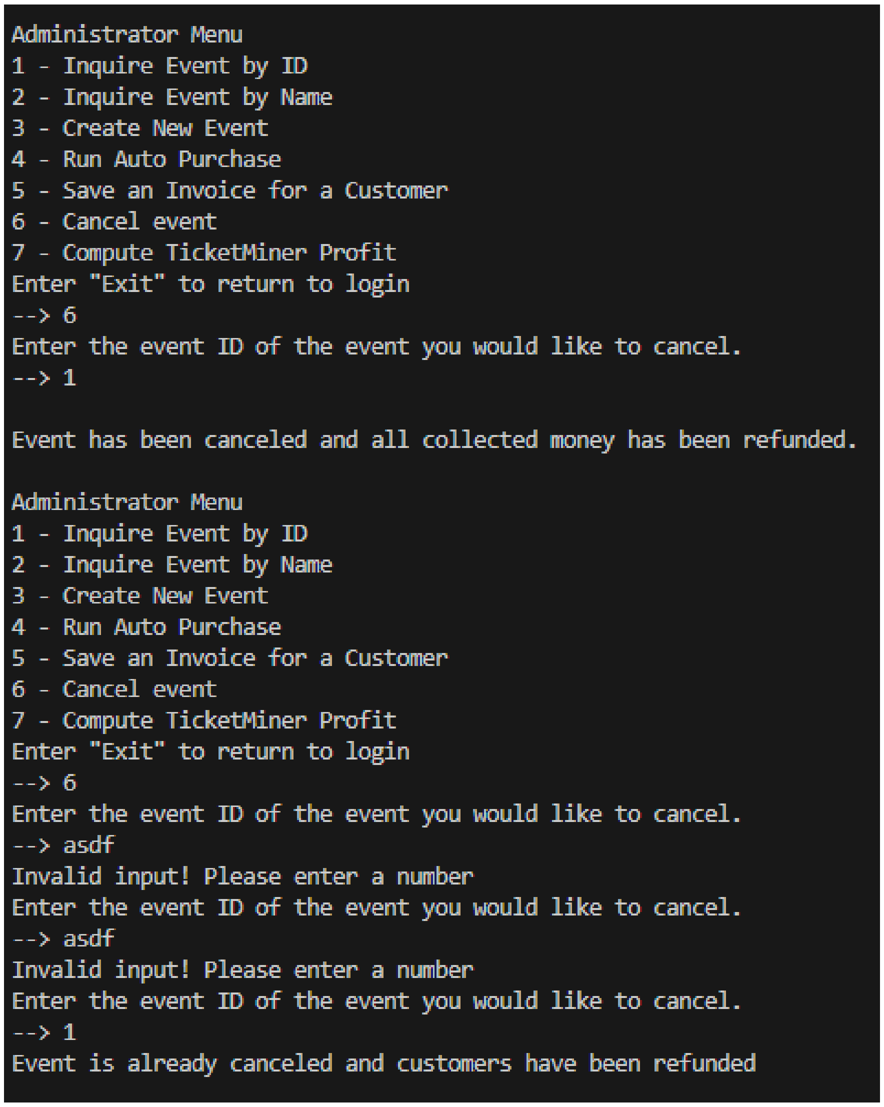
```
Miner Menu
1 - View Single Event Information
2 - View All Events
3 - Purchase Event Tickets
4 - View Invoices
5 - Save Invoices
6 - Cancel Purchase
or Type "Exit" to return to login
--> 4

-------------Invoices-------------
Event Name: (Purchase Canceled) UTEP Football 1
Event ID: 1
Ticket Type: gold
Number of tickets bought: 2
Total cost of order: $0.00
Total fees of order: $3.85
Total tax of order was: $0.00
Confirmation Number: 951390156
----------------------------------
```

Shows how an invoice is denoted as canceled to the customer.



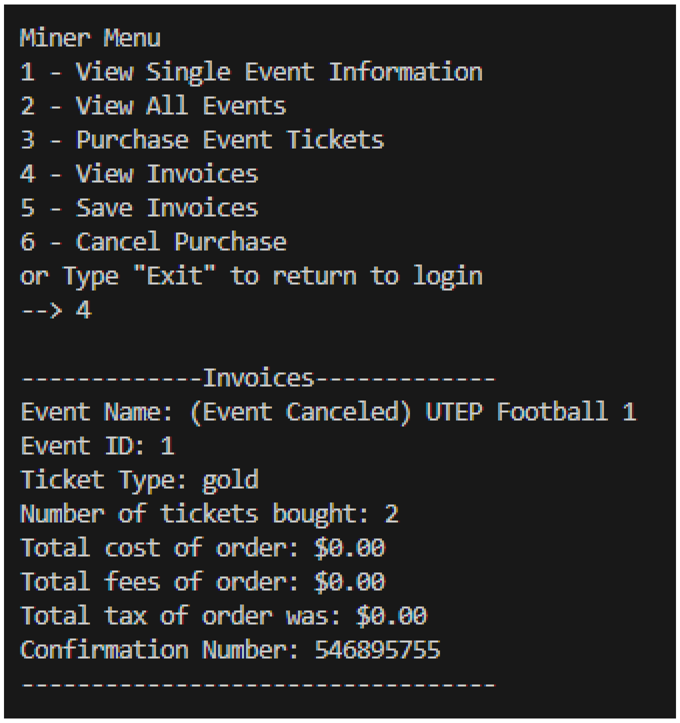
```
Administrator Menu
1 - Inquire Event by ID
2 - Inquire Event by Name
3 - Create New Event
4 - Run Auto Purchase
5 - Save an Invoice for a Customer
6 - Cancel event
7 - Compute TicketMiner Profit
Enter "Exit" to return to login
--> not something
Invalid input. Please enter a number 1 - 7

Administrator Menu
1 - Inquire Event by ID
2 - Inquire Event by Name
3 - Create New Event
4 - Run Auto Purchase
5 - Save an Invoice for a Customer
6 - Cancel event
7 - Compute TicketMiner Profit
Enter "Exit" to return to login
--> 1
Enter the Event ID
--> 2134
Not a valid Event ID
Enter the Event ID
--> 1
```

The test above shows that the admin menu will keep running with incorrect values and the same with all the menu options.



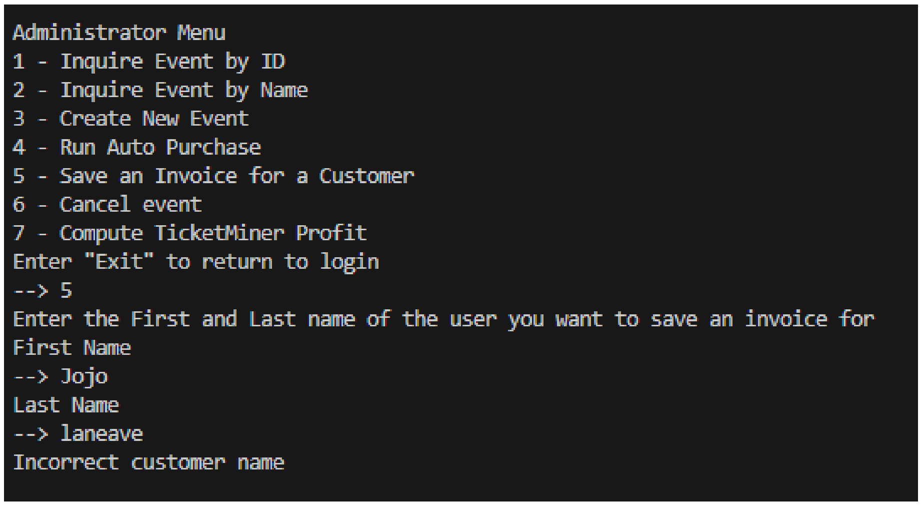
```
Administrator Menu
1 - Inquire Event by ID
2 - Inquire Event by Name
3 - Create New Event
4 - Run Auto Purchase
5 - Save an Invoice for a Customer
6 - Cancel event
7 - Compute TicketMiner Profit
Enter "Exit" to return to login
--> 6
Enter the event ID of the event you would like to cancel.
--> 1

Event has been canceled and all collected money has been refunded.

Administrator Menu
1 - Inquire Event by ID
2 - Inquire Event by Name
3 - Create New Event
4 - Run Auto Purchase
5 - Save an Invoice for a Customer
6 - Cancel event
7 - Compute TicketMiner Profit
Enter "Exit" to return to login
--> 6
Enter the event ID of the event you would like to cancel.
--> asdf
Invalid input! Please enter a number
Enter the event ID of the event you would like to cancel.
--> asdf
Invalid input! Please enter a number
Enter the event ID of the event you would like to cancel.
--> 1
Event is already canceled and customers have been refunded
```

The test above shows how an event is canceled and if the event is already canceled then it can not be canceled again.



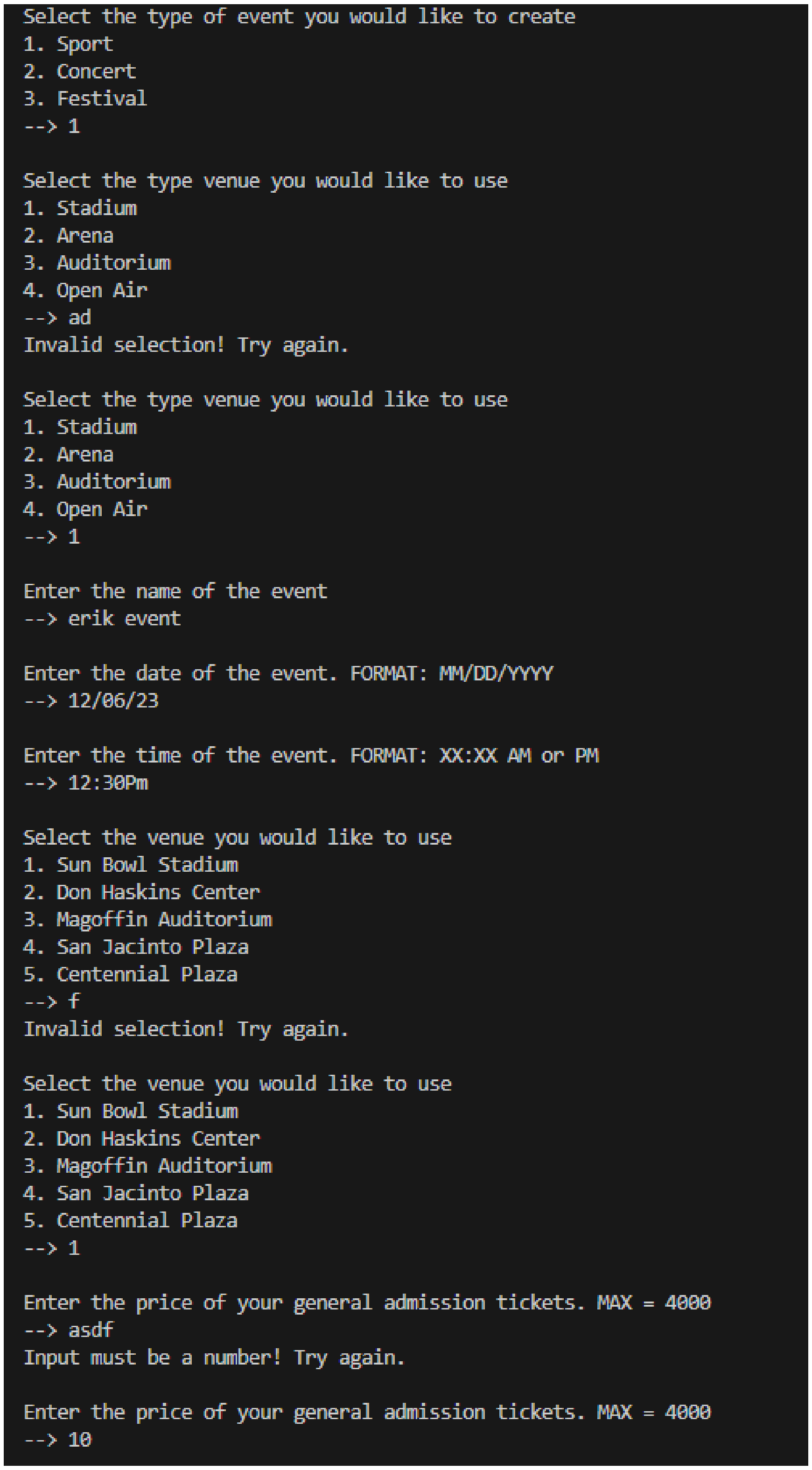
```
Miner Menu
1 - View Single Event Information
2 - View All Events
3 - Purchase Event Tickets
4 - View Invoices
5 - Save Invoices
6 - Cancel Purchase
or Type "Exit" to return to login
--> 4

-------------Invoices-------------
Event Name: (Event Canceled) UTEP Football 1
Event ID: 1
Ticket Type: gold
Number of tickets bought: 2
Total cost of order: $0.00
Total fees of order: $0.00
Total tax of order was: $0.00
Confirmation Number: 546895755
----------------------------------
```

The image above shows how a canceled event is shown to the customer.



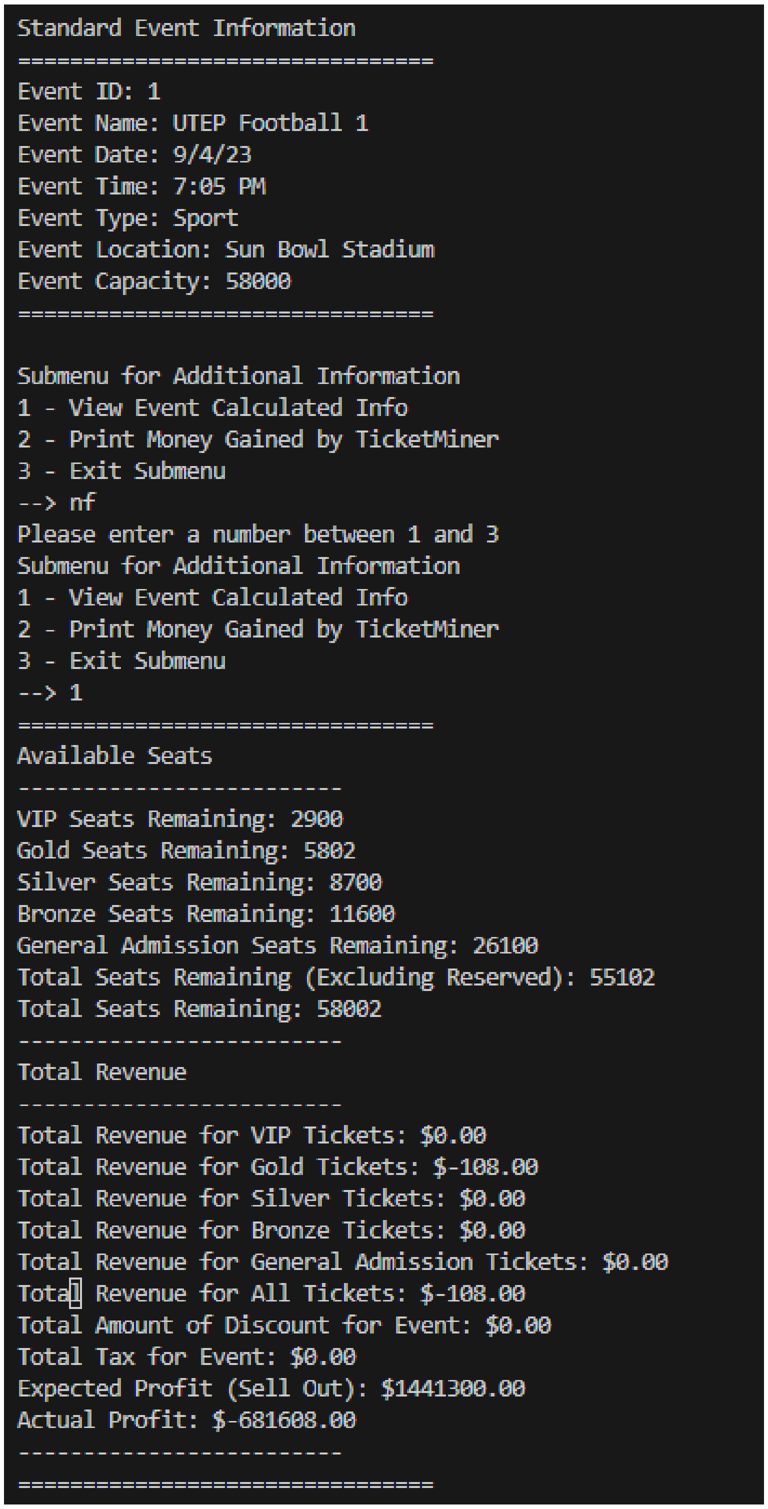
```
Administrator Menu
1 - Inquire Event by ID
2 - Inquire Event by Name
3 - Create New Event
4 - Run Auto Purchase
5 - Save an Invoice for a Customer
6 - Cancel event
7 - Compute TicketMiner Profit
Enter "Exit" to return to login
--> 5
Enter the First and Last name of the user you want to save an invoice for
First Name
--> Jojo
Last Name
--> laneave
Incorrect customer name
```

The test above shows that when looking for a customer the admin menu does not error out in the case of an unknown user.



```
Select the type of event you would like to create
1. Sport
2. Concert
3. Festival
--> 1

Select the type venue you would like to use
1. Stadium
2. Arena
3. Auditorium
4. Open Air
--> ad
Invalid selection! Try again.

Select the type venue you would like to use
1. Stadium
2. Arena
3. Auditorium
4. Open Air
--> 1

Enter the name of the event
--> erik event

Enter the date of the event. FORMAT: MM/DD/YYYY
--> 12/06/23

Enter the time of the event. FORMAT: XX:XX AM or PM
--> 12:30Pm

Select the venue you would like to use
1. Sun Bowl Stadium
2. Don Haskins Center
3. Magoffin Auditorium
4. San Jacinto Plaza
5. Centennial Plaza
--> f
Invalid selection! Try again.

Select the venue you would like to use
1. Sun Bowl Stadium
2. Don Haskins Center
3. Magoffin Auditorium
4. San Jacinto Plaza
5. Centennial Plaza
--> 1

Enter the price of your general admission tickets. MAX = 4000
--> asdf
Input must be a number! Try again.

Enter the price of your general admission tickets. MAX = 4000
--> 10
```

The image above shows how an event is added by the admin along with how it handles incorrect inputs.



```
Standard Event Information
================================
Event ID: 1
Event Name: UTEP Football 1
Event Date: 9/4/23
Event Time: 7:05 PM
Event Type: Sport
Event Location: Sun Bowl Stadium
Event Capacity: 58000
================================

Submenu for Additional Information
1 - View Event Calculated Info
2 - Print Money Gained by TicketMiner
3 - Exit Submenu
--> nf
Please enter a number between 1 and 3
Submenu for Additional Information
1 - View Event Calculated Info
2 - Print Money Gained by TicketMiner
3 - Exit Submenu
--> 1
================================
Available Seats
-------------------------
VIP Seats Remaining: 2900
Gold Seats Remaining: 5802
Silver Seats Remaining: 8700
Bronze Seats Remaining: 11600
General Admission Seats Remaining: 26100
Total Seats Remaining (Excluding Reserved): 55102
Total Seats Remaining: 58002
-------------------------
Total Revenue
-------------------------
Total Revenue for VIP Tickets: $0.00
Total Revenue for Gold Tickets: $-108.00
Total Revenue for Silver Tickets: $0.00
Total Revenue for Bronze Tickets: $0.00
Total Revenue for General Admission Tickets: $0.00
Total Revenue for All Tickets: $-108.00
Total Amount of Discount for Event: $0.00
Total Tax for Event: $0.00
Expected Profit (Sell Out): $1441300.00
Actual Profit: $-681608.00
-------------------------
================================
```

This test shows that the submenu in the admin can handle random and incorrect inputs and continue displaying and asking the user for input.

6. Demo with Classmates

During the demo with the other team in class, we had the opportunity to test each other's code without being able to look at the source code. Testing the program without seeing the inner workings lets us focus on the feel and function of the program. We forgot about data structures, efficiency, and the object-oriented approach and got to experience the program precisely like a customer. This demo helped us discuss things like UI menus and error handling. After the demo, we also talked about our programs' inner workings, which are always interesting to compare and contrast.

7. Code Review

To complete our code review, we went through each section of the code review checklist and made sure we were in good standing. For the implementation section, our code met all of the requirements. Our code does what it is supposed to; it is dynamically coded and easily maintainable through classes and methods. This made the merge go smoothly which gave more time to implement the new functionality. We don't think it can be simplified significantly in a meaningful way. For the logic part, our code handles misinput gracefully and does not stall or break. As for the readability and style part, we believe our code is easy to read and understand. It is structured appropriately and is well documented. It follows Java standards. At worst, our code has a complexity of O(n^2) due to nested while loops to control the flow of the program and the auto purchase but after the first thousand runs the complexity is reduced significantly to O(n) with the addition of the cache. However, most of the program runs in constant time. Changes in input do not have any significant changes in the complexity.