Tunku Abdul Rahman University of Management and Technology

AACS2204

Object-Oriented Programming Techniques

Assignment

2023/2024

Programme : DSFY1S3

Tutorial Group :

Date Submitted to Tutor : 6/5/2024

Team Members:

No	Student Name	Student ID
1.	Elisha Tiong Pei Pei	23SMD05160
2.	Natalie Koa Hao Yee	23SMD01336
3.	Michelle Chin Koh Ying	23SMD00488
4.	Tan Shieh Ling	23SMD05432

No.	Team Member	Task(s) Allocated	Overall Contribution (%)
1.	Elisha	In charge of: - Worked on transaction class. - Class diagram. - admin login, display admin menu, confirm purchase, view all transaction, view bus operation. - Report.	20%
2.	Natalie Koa Hao Yee	In charge of: - Worked on admin, customer , and user class. - Class diagram. - register new customer, customer login, purchase ticket, generate receipt, view bus operation. - Report.	20%
3.	Michelle Chin koh Ying	In charge of: - Worked on payment class. - Class diagram. - Customer menu, admin menu, purchase ticket , calculate total cost, view bus operation. - Report.	20%
4.	Tan Shieh Ling	In charge of: - Worked on ticket class. - Class diagram. - Display customer menu, Display main menu, ,Ticket createticket, display available tickets, view bus operation. - Report.	20%

Coursework Declaration

We confirm that we have read and shall comply with all the terms and conditions of TAR University College's plagiarism policy.

We declare that this assignment is free from all forms of plagiarism and for all intents and purposes is our own properly derived work.

Signature	:				
Name	:	Elisha Tiong Pei Pei	Natalie Koa Hao Yee	Michelle Chin Koh Ying	Tan Shieh Ling
Date	:	2 May 2024	2 May 2024	2 May 2024	2 May 2024

Table Of Contents

rubrics

AACS2204 Object-Oriented Programming Techniques - Assignment Feedback Form

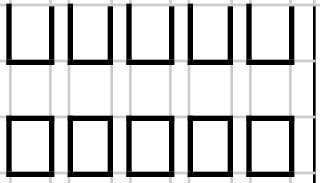
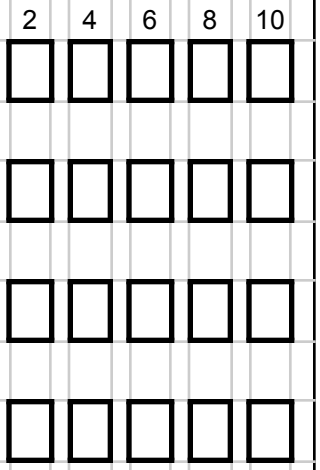
		Name	Total Marks
Students:	1.	Elisha Tiong Pei Pei	
	2.	Natalie Koa Hao Yee	
	3.	Michelle Chin Koh Ying	
	4.	Tan Shieh Ling	

Programme of Study:	DSF1
Tutorial Group:	

CLO 2	**Demonstrate an object-oriented using appropriate programming fundamentals with regards to array, methods and exception handling (P4, PLO3)**

Program (50 marks)

Area	Marks	Very Poor	Rating: Very Poor to Excellent (3 6 9 12 15)	Excellent	Remarks
Completeness (Individual Mark)	15	Program only functions correctly in very limited cases or not at all.	☐ ☐ ☐ ☐ ☐	Completed 100% of the functional requirements. All operations are implemented correctly that ~~includes array,~~	

				methods and exception handling.	
Correct Output (Individual Mark)	10	Output is incorrect and without exception handling.		Program displays correct output with no errors. Handle with proper exception handling.	

Design of Output (Individual Mark)	5	Output is poorly designed.	1 2 3 4 5 ☐ ☐ ☐ ☐ ☐	Nicely formatted output. Program displays more than expected.	
Class (Group Mark)	5	Less/fail to identify the proper classes, methods and attributes to solve a particular problem.	1 2 3 4 5 ☐ ☐ ☐ ☐ ☐	Identified the proper classes, methods and attributes to solve particular problems.	
Object (Group Mark)	5	Poor structure of object collaboration like one object doing everything itself.	1 2 3 4 5 ☐ ☐ ☐ ☐ ☐	Able to create a structure of objects collaborating among themselves to carry out tasks properly.	
Cohesion (Group Mark)	5	Some classes represent more than one entity. Poor class cohesion.	1 2 3 4 5 ☐ ☐ ☐ ☐ ☐	Each class represents a single entity. Each method performs a single well-defined operation that fits the class.	

Coupling (IGroup Mark)	5	High coupling which makes modules harder to reuse or test.	1 2 3 4 5 ☐ ☐ ☐ ☐ ☐	Low coupling in which one module interacts with another module through a simple and stable interface.	
CLO 3	**Illustrate the concepts of encapsulation, inheritance, and polymorphism to solve a given programming problem. (C4, PLO2)**				
Report (50 marks)					
Area	**Marks**	**Very Poor**	**Rating**	**Excellent**	**Remarks**

UML Class Diagram (Group Mark)	10	Most of the entities and relationships in the UML class diagram design are inappropriate.	2 ☐ 4 ☐ 6 ☐ 8 ☐ 10 ☐	The whole UML class diagram design has all the appropriate entities and relationships.	
Encapsulation and Polymorphism (Group Mark)	15	Completely incorrect implementation of encapsulation (private modifier, setter and getter methods) and polymorphism.	3 ☐ 6 ☐ 9 ☐ 12 ☐ 15 ☐	Completely correct implementation of encapsulation (private modifier, setter and getter methods). Completely correct use and implementation of polymorphism. Methods toString() & equals() correctly overridden.	
Association, Inheritance, Aggregation and Composition (Group Mark)	15	Completely incorrect use and implementation of association, aggregation inheritance, and composition relationship.	3 ☐ 6 ☐ 9 ☐ 12 ☐ 15 ☐	Completely correct use and implementation of association, inheritance, aggregation and composition relationship.	
Descriptions of System (Group Mark)	10	Poor description of the system with missing screenshots.	2 ☐ 4 ☐ 6 ☐ 8 ☐ 10 ☐	Completely correct descriptions of the system with proper screen shots.	

	1. Elisha Tiong Pei Pei				
	2. Natalie Koa Hao Yee				
	3. Michelle Chin Koh Ying				
	4. Tan Shieh Ling				

Introduction

Our team has selected the Transportation industry as the industry that our program will be programmed for. The transportation industry involves transporting goods, people , or animals from one place to another. This industry is one of the vital sectors of the economy that enables trade, commerce , tourism , and mobility.

The modern transportation industry we recognize began to take shape back in the late 19th and early 20th centuries with mass production of automobiles , the expansion of railway networks, and aviation. Since then, the transportation industry has continued to evolve rapidly, driven by technological advancements.

The essential requirement for the transportation industry we have selected is ticketing and fare collection. The system that we will develop should be capable of processing ticket sales for passengers, display the transaction history from the beginning to the end ,and display bus operation time and place.

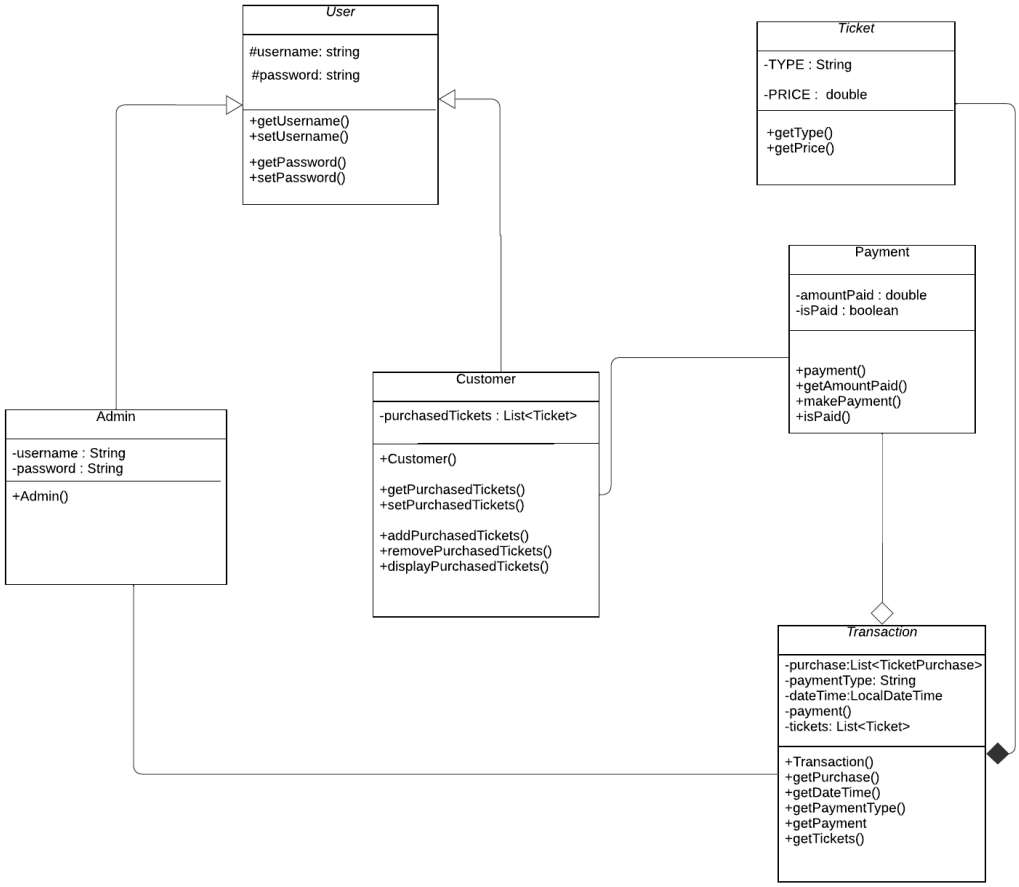
Company Background

The company that my team has decided to create a software for is Metrobus Nationwide Sdn Bhd. It is a privately owned public transport company in Sabah that was formed in 1992. Based in Kota Kinabalu. Metrobus owns a fleet of Nissan Diesel, Hino and Mercedes Benz buses.

Metrobus is strategically located in multiple areas throughout Sabah in residential , industrial , and commercial areas. Metrobus operates everyday on weekdays 6 AM - 11 PM , to accommodate those who are returning home after work. On weekends the bus will operate from 6 AM - 10.30pm.

Metrobus uses a manual way of collecting the fares for transportation. This is inefficient because it slows down boarding time and is difficult in controlling the crowd. Additionally it requires additional resources for handling cash. This issue can be solved by implementing a ticketing system allowing passengers to get various amounts of tickets such as one day pass, weekly pass, and monthly pass. For example, if you purchase a one day pass the customer can use the pass for 24 hours once 24 hours is up the pass is unusable.

Class Diagram



Relationship Used

Inheritance

Inheritance is a relationship where one class (subclass) inherits attributes and behaviors from another class (superclass).

In this system, both ‘Admin’ and ‘Customer’ classes inherit attributes and behaviors from the ‘User’ class, representing an ‘is-a’ relationship. For example, an admin or customer is a type of user

Composition

Composition is a strong form of association where one class contains objects of another class as part of its attribute.

In this system, the ‘Transaction’ class contains a list of ‘Ticket’ objects, indicating that ‘Transaction’ is composed of ‘Ticket’ objects.

Aggregation

Aggregation is a weaker form of association where one class has a reference to another class as part of its attributes.

In this system, the ‘Transaction’ class has a reference to a ‘Payment’ object, representing an aggregation relationship. This means that a ‘Transaction’ is associated with ‘Payment’, but ‘Payment’ object can exist independently of the ‘Transaction’ object.

Association

Association represents a relationship between two classes, indicating that objects of one class are connected to objects of another class.

In this system, ‘Admin’ class and ‘Transaction’ class are associated. This association implies that an ‘Admin’ object can interact with one or more ‘Transaction’ objects.

Fundamentals of Object Oriented Programming

Encapsulation

Encapsulation refers to the bundling data (attributes) and methods (functions) that operate on the data into a single unit, typically known as a class. This concept helps in hiding the internal state of an object from the outside world and only allowing access to it through well-defined interfaces.

There are 3 access modifiers used:

1. ‘private’ members are accessible only within the same class.
2. ‘protected’ members are accessible within the same package and subclasses.
3. ‘public’ members are accessible from any other class.

In the system, there are attributes that are ‘protected’ where inheritance takes place.

(User Class)

```
class User {
    //declare private field
    protected String username;
    protected String password;
```

The ‘protected’ modifier makes the field (‘username’ and ‘password’) accessible within the same package and by subclasses. This means that the fields are accessible to subclasses such as ‘Admin’ and ‘Customer’, which inherit from the ‘User’ class.

‘private’ is used throughout the code.

(Payment Class)

```
public class Payment {
    private double amountPaid;
    private boolean isPaid;
```

The encapsulation is employed to control the access to its internal state and behavior. The ‘amountPaid’ and ‘isPaid’ fields are declared as ‘private’. This means that they are accessible only within the ‘Payment’ class itself and are unable to be accessed directly from outside the class. Which also means that their internal representation is hidden from external classes.

Polymorphism

Polymorphism refers to the ability of different classes to be treated as instances of a common superclass or through a common interface. This allows objects of different types to be handled in a consistent manner, providing flexibility and extensibility to the code.

(Ticket Class)

```java
// Define the Ticket interface to represent a ticket in the sy
interface Ticket {
    String getType();
    double getPrice();
}

// OneDayTicket class
class OneDayTicket implements Ticket {
    private static final String TYPE = "One Day";
    private static final double PRICE = 6.0;

    // Modified constructor to accept no arguments
    public OneDayTicket() {
        // Constructor implementation
    }

    @Override
    public String getType() {
        return TYPE;
    }

    @Override
    public double getPrice() {
        return PRICE;
    }
}

// WeeklyTicket class
class WeeklyTicket implements Ticket {
    private static final String TYPE = "Weekly";
    private static final double PRICE = 18.0;

    // Modified constructor to accept no arguments
    public WeeklyTicket() {
        // Constructor implementation
    }

    @Override
    public String getType() {
        return TYPE;
    }

    @Override
    public double getPrice() {
        return PRICE;
```

```
    @Override
    public double getPrice() {
        return PRICE;
    }
}

// MonthlyTicket class
class MonthlyTicket implements Ticket {
    private static final String TYPE = "Monthly";
    private static final double PRICE = 30.0;

    // Modified constructor to accept no arguments
    public MonthlyTicket() {
        // Constructor implementation
    }

    @Override
    public String getType() {
        return TYPE;
    }

    @Override
    public double getPrice() {
        return PRICE;
    }
}
```

Shown in the diagram above, what is shown is method overrides. Method overriding happens when a subclass provides a specific implementation of a method that is already defined in its superclass.

For this case, Each class (‘OneDayticket’ , ‘WeeklyTicket’ , ‘MonthlyTicket’) implements the ‘Ticket’ interface and provides its own implementation for the methods ‘getType()’ and ‘getPrice()’. Method overriding occurs when these classes override the methods defined in the ‘Ticket’ interface with their own specific implementations.

Inheritance

Inheritance allows a class(subclass) to inherit properties and behaviors from another class (superclass). This means that the subclass can reuse the code defined in the superclass without having to rewrite it.

(User Class)

```
class User {
    //declare private field
    protected String username;
    protected String password;

    // Constructor method for initializing a User object with a username and password
    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }
```

'User' class serves as the superclass for both 'Customer' and 'Admin' classes. It contains common attributes and methods that are shared by both types of users. Such as, 'username' and 'password'. Both 'Customer' and 'Admin' classes inherit these attributes and methods from 'User' class.

(Admin Class)

```
public class Admin extends User {
    public Admin(String username, String password) {
        super(username, password);  // Call the constructor of the superclass (User) with provided username and password
    }

}
```

The 'Admin' class is a subclass of 'User'. It inherits the 'username' and "password" attributes from the 'User' class using the 'extends' keyword.

(Customer Class)

```
import java.util.ArrayList; // Import ArrayList class from java.util package
import java.util.List; // Import List interface from java.util package


public class Customer extends User {
    private List<Ticket> purchasedTickets;

    public Customer(String username, String password) {
        super(username, password);  // Call the constructor of the superclass (User) with provided username and password
        this.purchasedTickets = new ArrayList<>();  // Initialize the purchasedTickets list with an empty ArrayList
    }
```

The 'Customer' class is a subclass of 'User'. It Inherits the 'username' and 'password' attributes from the 'User' class using the 'extends' keyword.

Metrobus Ticketing System

Description

The Metrobus Ticketing System starts with a menu page where users can choose to register, login as customer , and login as admin. By clicking ‘1’ user is directed to the register where you have to enter customer username and password to register. By pressing ‘2’ the user is shown the customer login where the user has to input the registered username and password. Other than that, by pressing ‘3’ user is then shown admin login where a username and password is needed to enter the admin menu. Lastly, by pressing ‘4’ which is to exit the program user will exit from the program.

When logged in as Customer, the user is shown the ‘Customer Menu’ where the user can either press ‘1’ to purchase tickets, press ‘2’ to view bus operation , and press ‘3’ to logout from the customer menu.

When logged in as Admin, the user will be shown the ‘Admin Menu’ where the admin can view the transaction history.

Metrobus Ticketing Program

Scenarios

- Logging in as Admin and viewing transaction history.
- Registering as a customer and entering the system as a customer. Purchasing ticket and view bus operation.

As A Customer

```
--------------------Configuration: <Default>--------------------
----- Metrobus Ticketing System -----
1. Register
2. Login
3. Admin Login
4. Exit
Enter your choice: 1
```

Enter a digit to either register or login. In this case, enter ‘1’ to register a customer account.

Input: 1

```
----- Register New Customer -----
Enter username: customer
Enter password: customer123
Account registered successfully!
```

Once a user enters ‘1’ user is prompted to enter username and password to register new customer account.

Input username: customer
Input password: customer123

```
----- Metrobus Ticketing System -----
1. Register
2. Login
3. Admin Login
4. Exit
Enter your choice:
```

User is led back to the Metrobus Ticketing System main menu. To continue as a customer enter ‘2’ to login as a customer. Then the user is shown the customer login.

Input: 2

```
Enter your choice: 2
----- Customer Login -----
Enter username: customer
Enter password: customer123
Login successful as customer: customer
```

The user is then prompted to enter the username and password of the registered customer account to login. Then the Customer Menu is displayed.

Input username: customer
Input password: customer123

```
----- Customer Menu -----
1. Purchase Ticket
2. View Bus Operation
3. Logout
Enter your choice:
```

Customer Menu shows 3 options which are purchasing a ticket, view bus operation , and logout. The customer will input ‘1’ to purchase a ticket. The user will then be shown the available tickets.

Input: 1

```
Enter your choice: 1
Available Tickets:
1. One Day
2. Weekly
3. Monthly
Enter the number of the ticket you want to purchase:
```

The available tickets are displayed. Users can enter ‘1’ to select a one day ticket.

Input: 1

```
Available Tickets:
1. One Day
2. Weekly
3. Monthly
Enter the number of the ticket you want to purchase: 1
Enter the quantity: 3
Ticket Type: One Day
Quantity: 3
Total Cost: 18.0
Confirm purchase? (yes/no): yes
Ticket(s) added to cart successfully!
Would you like to purchase more tickets? (yes/no): no
Total Cost: 18.0
Please select payment type:
1. Debit
2. Credit
3. Back
Enter your choice:
```

Then the user is prompted to enter the quantity which will be ‘3’. User is then asked to confirm the purchase by entering (yes/no) user shall input ‘yes’ to add to cart. Users are then prompted to continue either enter ‘yes' to purchase more tickets or ‘no’ to not continue. In this case the user enters ‘no’. The user is then shown the total cost and to select the payment type.

Input: 3
Input: yes
Input: no

```
Total Cost: 18.0
Please select payment type:
1. Debit
2. Credit
3. Back
Enter your choice: 1
Enter payment amount: 18
Payment successful!
Change: 0.0
--------------- Metrobus Receipt ---------------
Date and Time: 2024-05-06T11:00:16.669
Customer: customer
------------------------------------------------
For More Information. Please Contact Us At:
Contact Number: +1234567890
Email: MetrobusServices@gmail.com
------------------------------------------------
Payment Type: Debit  **** **** ****
Ticket Type          Price      Quantity   Total
One Day              6.00       3          18.00

Total Cost: 18.0
------------------------------------------------

Thank you for purchasing tickets from Metrobus!
```

The user will then enter ‘1’ to pay using debit. The user is prompted to enter a payment amount which is ‘18’ . Payment will be successful and the receipt will be shown. The customer will then be shown the customer menu.

Input: ‘1’ to pay using debit.
Input: ‘18’ to pay the total cost.

```
----- Customer Menu -----
1. Purchase Ticket
2. View Bus Operation
3. Logout
Enter your choice: 2
```

Users will enter ‘2’ to view bus operation. Bus operation time and place will then be shown.

Input: 2

```
----- Bus Operation -----
Route: Penampang to Alamesra and Back
Departure Times:
Morning: 7:00 AM, 8:30 AM, 10:00 AM
Afternoon: 12:00 PM, 2:00 PM, 4:00 PM
Evening: 6:00 PM, 7:30 PM, 9:00 PM

Route: Kepayan to Alamesra and Back
Departure Times:
Morning: 6:45 AM, 8:15 AM, 9:45 AM
Afternoon: 11:45 AM, 1:45 PM, 3:45 PM
Evening: 5:45 PM, 7:15 PM, 8:45 PM

Route: Inanam to Alamesra and Back
Departure Times:
Morning: 6:30 AM, 8:00 AM, 9:30 AM
Afternoon: 11:30 AM, 1:30 PM, 3:30 PM
Evening: 5:30 PM, 7:00 PM, 8:30 PM

Route: Town to Alamesra and Back
Departure Times:
Morning: 7:15 AM, 8:45 AM, 10:15 AM
Afternoon: 12:15 PM, 2:15 PM, 4:15 PM
Evening: 6:15 PM, 7:45 PM, 9:15 PM

----- Customer Menu -----
1. Purchase Ticket
2. View Bus Operation
3. Logout
Enter your choice:
```

Enter ‘3’ to logout.

Input: 3

As An Admin

```
----------------------Configuration: <Default>--------------------
----- Metrobus Ticketing System -----
1. Register
2. Login
3. Admin Login
4. Exit
Enter your choice:
```

Enter a digit to select a role you want to login or register as. In this case, enter ‘3’ to login as admin.

Input: 3

```
Enter your choice: 3
----- Admin Login -----
Enter username: admin
Enter password: admin123
Login successful as admin: admin
```

The program will require the user to enter the correct username and password to login as admin. Once logged in the admin menu will be displayed.

Input in username: admin
Input in password: admin123

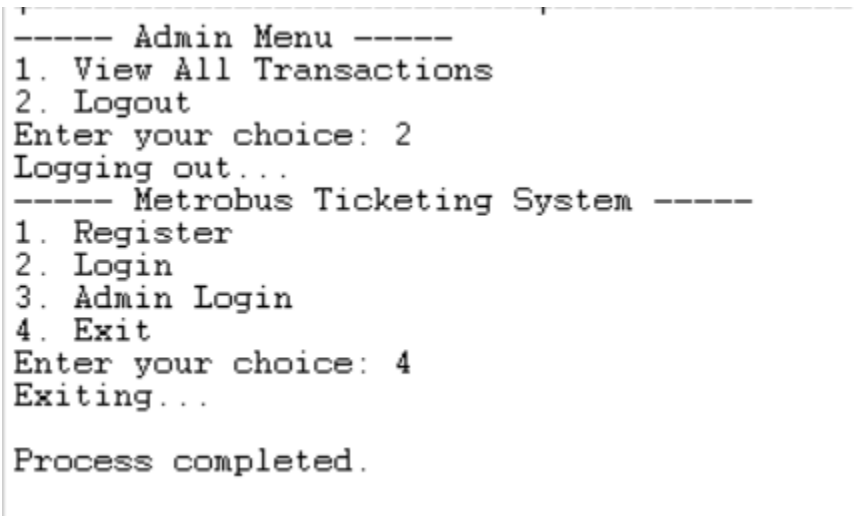
```
----- Admin Menu -----
1. View All Transactions
2. Logout
Enter your choice:
```

Shown above is the Admin Menu where admin can view transactions made through the program.

```
+-------------------------+-------------------------+-------------------------+----------------+----------------+
| Date and Time           | Ticket Type             | Quantity                | Payment Type   | Total Cost     |
+-------------------------+-------------------------+-------------------------+----------------+----------------+
| 2024-05-06T11:00:12.637 | One Day                 | 3                       | Debit          | 18.00          |
+-------------------------+-------------------------+-------------------------+----------------+----------------+
----- Admin Menu -----
1. View All Transactions
2. Logout
Enter your choice:
```

This is the transaction history. It is displaying the purchase we made as a user. Enter ‘1’ to view transaction history and it will be displayed.

Input: 1



The Admin Menu will be displayed after the transaction history is displayed. Users enter ‘2’ to logout and enter ‘4’ to exit the program.

Input: 2
Input: 4