






Name(s): 1. MUHAMMAD AKMAL BIN MARZUKI 2. AFIQ ISKANDAR BIN MANAN 3. IRFAN ISTIQLAL BIN AIDIL AHMAD SHAM		
ID Number(s): 1. AM2211012788 2. AM2211012858 3. AM2211012762		
Lecturer Name: MADAM NOORNAJWA BINTI MD AMIN		Lab group / Tutorial group / Tutor (if applicable):
Course Name and Course Code: SWC3344-DATA STRUCTURE		Submission Date: 6/11/2023
Assignment No. / Title Proposal: GROUP PROJECT (30%)		Extension & Late Submission: *Allowed / Disallowed
Assignment Type: Group	% of Assignment Mark:	Returning Date:
Penalties: 1. 10% of the original mark will be deducted for every one-week period after the submission date. 2. No work will be accepted after two week of the deadline.		

3. If you were unable to submit the coursework on time due to extenuating circumstances, you may be eligible for an extension.
4. Extension will not exceed one week.

Declaration: We the undersigned confirm that we have read and agree to abide by these regulations on plagiarism and cheating. We confirm that this of work is our own. We consent to appropriate storage of our work for checking ton ensure that there is no plagiarism/academic cheating.

Full Name	Signature
MUHAMMAD AKMAL BIN MARZUKI (AM2211012788)	
AFIQ ISKANDAR BIN MANAN AM2211012858	
IRFAN ISTIQLAL BIN AIDIL AHMAD SHAM (AM2211012762)	

This section may be used for feedback or other information

TABLE OF CONTENTS

No.	Contents	Page
1.0	Introduction	4
2.0	Analysis	5 - 6
	Flowchart & Class Definition	6 - 9
3.0	Design of Our ADTs using UML Class Diagram and show their relations Design of the appropriate data structure(s) class	10
4.0	Java code and sample input and output.	11-38
5.0	Program Output	39 - 49
6.0	Lessons learned, reflective experience towards completing the individual/overall tasks and Conclusion .	50 - 52
7.0	References	53

1.0 Introduction

For this project, it required to design and develop a GUI application by applying appropriate data structures and algorithm. The first reason to do this project because it wants us to understand about queues and stacks that handles ticketing system. Our group which consists of Afiq Iskandar, Muhammad Akmal and Irfan Istiqlal are required to prepare a report based on this project which we are required to work together as a team and construct a ticketing system for our coding and system that is used in our program. We are responsible to manage the customer and ticket details into our management system. Each of us will separate into working different types of functions in the JAVA program to simplify our task. This way we can combine our function together to complete the JAVA program and cause the service to run smoothly.

Our ticketing system is called Manafan Ticketing System. This kind of function is to manage all the customer in the list to control the data into each counter to be run smoothly and properly. We will provide three counter to proceed all the customer in list into each counter with a condition. Each process on the counter will take 5 customer each will take from the list and process into counter 1 and 2 as express counter and counter 3 is regular counters. At this process will continue until all the 100 customers in the list are completed.

2.0 Analysis

IPO (INPUT, PROCESS, OUTPUT)

INPUT:

- Read 100 customers data from customerList.txt document.
- To input all the data, we just need to click Load Data button.
- We can see the loaded data by clicking the Show Data button, and it will print out all the 100 data inside customerList.txt

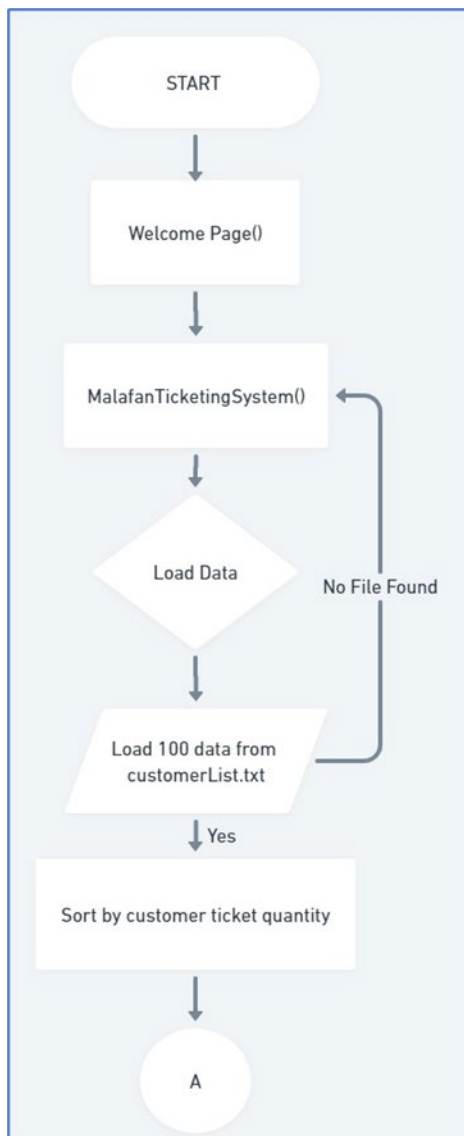
PROCESS:

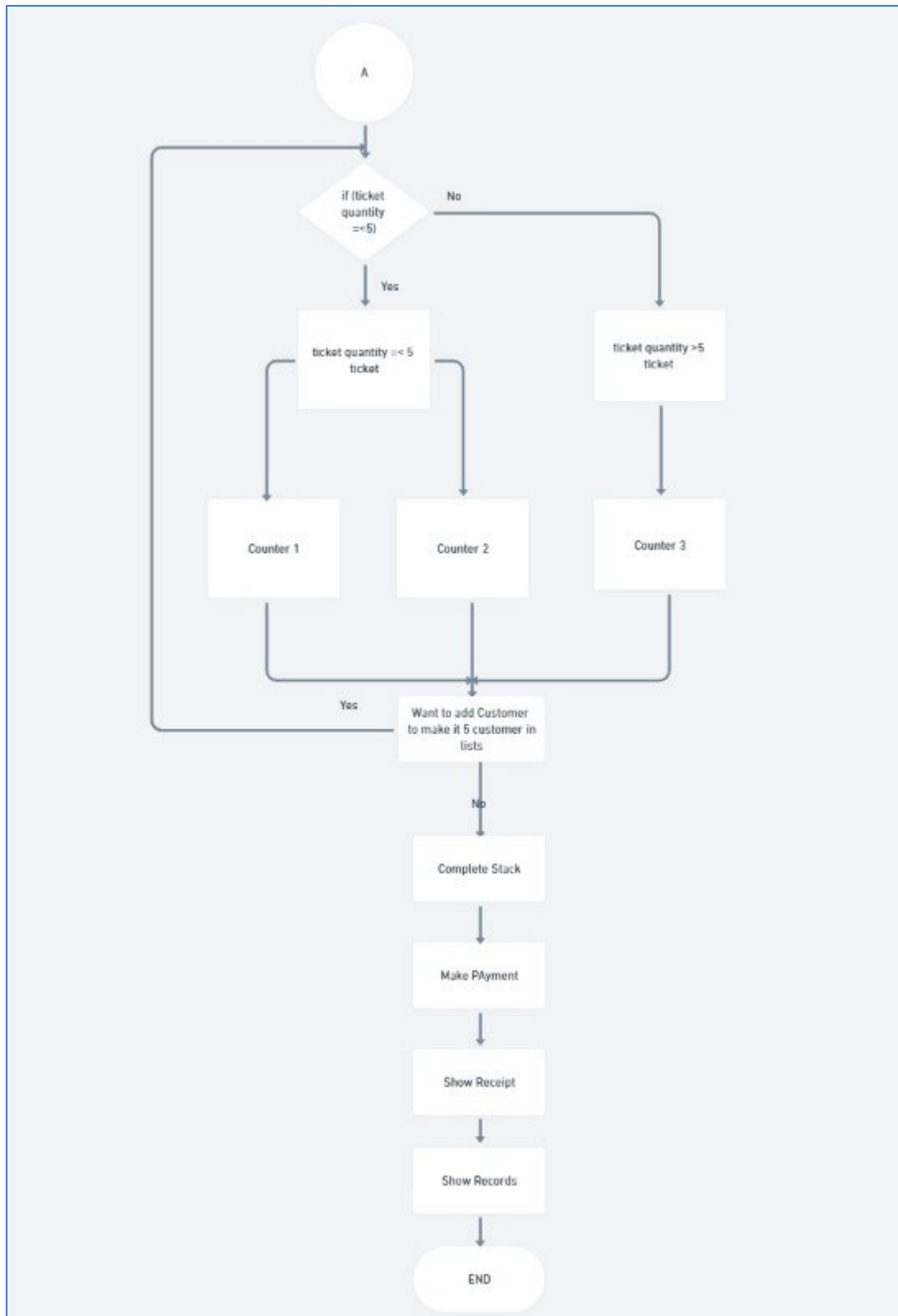
- This program needs to divide all the customer data into three counters.
- This programme will read the data from the customerList.txt document and separate it into three counters based on the assigned customer ticket quantity.
- The third counter is only for customers who have a ticket quantity greater than 5.
- The third counter is only for customers who have a ticket quantity greater than 5.
- The function button for Process Counter 1, Process Counter 2, and Process Counter 3 is the way the customer is being processed. Example: If I click Process Counter 1, it will immediately process customers that have a quantity ticket less than 5 inside the customerList.txt document. The processed customer will be shown inside the stack in the main panel.
- The counter will process five customers at a time.
- The Display Customers button is working by showing us a detailed review of the receipt for the processed customer inside the second panel on the right.

OUTPUT :

- The printout receipt will work, but we need to choose from the options at the bottom right of the frame. After choosing the customer, click the Show Receipt button, and then the chosen customer receipt will appear.
- The printout receipt will generate the total number of customers by multiplying the ticket and quantity for each customer.

DESIGN ALGORITHM (FLOWCHART)





CLASS DEFINITION

Customer Information class:

```
import java.util.*;

// This class represents customer information including their purchased tickets
class CustomerInformation {

    // Unique customer identifier
    private String custId;

    // Name of the customer
    private String custName;

    // Represents the latest ticket ID purchased by this customer
    private int ticketId;

    // Represents the counter number where the customer makes purchases
    private int counterNumber;

    // List of tickets purchased by this customer
    private List<TicketInformation> purchasedTickets;

    // Constructor to initialize customer information with ID and name
    public CustomerInformation(String custId, String custName) {
        this.custId = custId;
        this.custName = custName;
        this.ticketId = 0; // Setting initial ticketId to 0
        this.purchasedTickets = new ArrayList<>();
    }

    // Adds a ticket to the list of tickets purchased by this customer
    public void addItem(TicketInformation ticket) {
        purchasedTickets.add(ticket);
    }

    // Returns the customer ID
    public String getCustId() {
        return custId;
    }

    // Returns the customer name
    public String getCustName() {
        return custName;
    }

    // Returns the latest ticket ID purchased by the customer
    public int getTicketId() {
        return ticketId;
    }

    // Increments the ticket ID when a new ticket is purchased
    public void incrementCounterPaid() {
        ticketId++;
    }

    // Returns the list of tickets purchased by the customer
    public List<TicketInformation> getPurchasedTickets() {
        return purchasedTickets;
    }

    // Returns the counter number associated with the customer
    public int getCounterNumber() {
        return counterNumber;
    }

    // Sets the counter number for the customer
    public void setCounterNumber(int counterNumber) {
        this.counterNumber = counterNumber;
    }

    // Returns a string representation of the customer, displaying their ID and name
    @Override
    public String toString() {
        return custId + " - " + custName;
    }
}
```


Ticket Information class:

```
// This class represents information about a ticket purchase for a ride.
class TicketInformation {

    // Unique identifier for the ticket
    private int ticketId;

    // Name of the ride associated with the ticket
    private String rideName;

    // Price of the individual ticket
    private int ticketPrice;

    // Date when the ticket was purchased
    private String purchaseDate;

    // Number of tickets purchased in this transaction
    private int purchasedQuantity;

    // Constructor to initialize all fields of the ticket
    public TicketInformation(int ticketId, String rideName, int ticketPrice, String purchaseDate, int purchasedQuantity) {
        this.ticketId = ticketId;
        this.rideName = rideName;
        this.ticketPrice = ticketPrice;
        this.purchaseDate = purchaseDate;
        this.purchasedQuantity = purchasedQuantity;
    }

    // Returns the unique ID of the ticket
    public int getTicketId() {
        return ticketId;
    }

    // Returns the name of the ride associated with the ticket
    public String getRideName() {
        return rideName;
    }

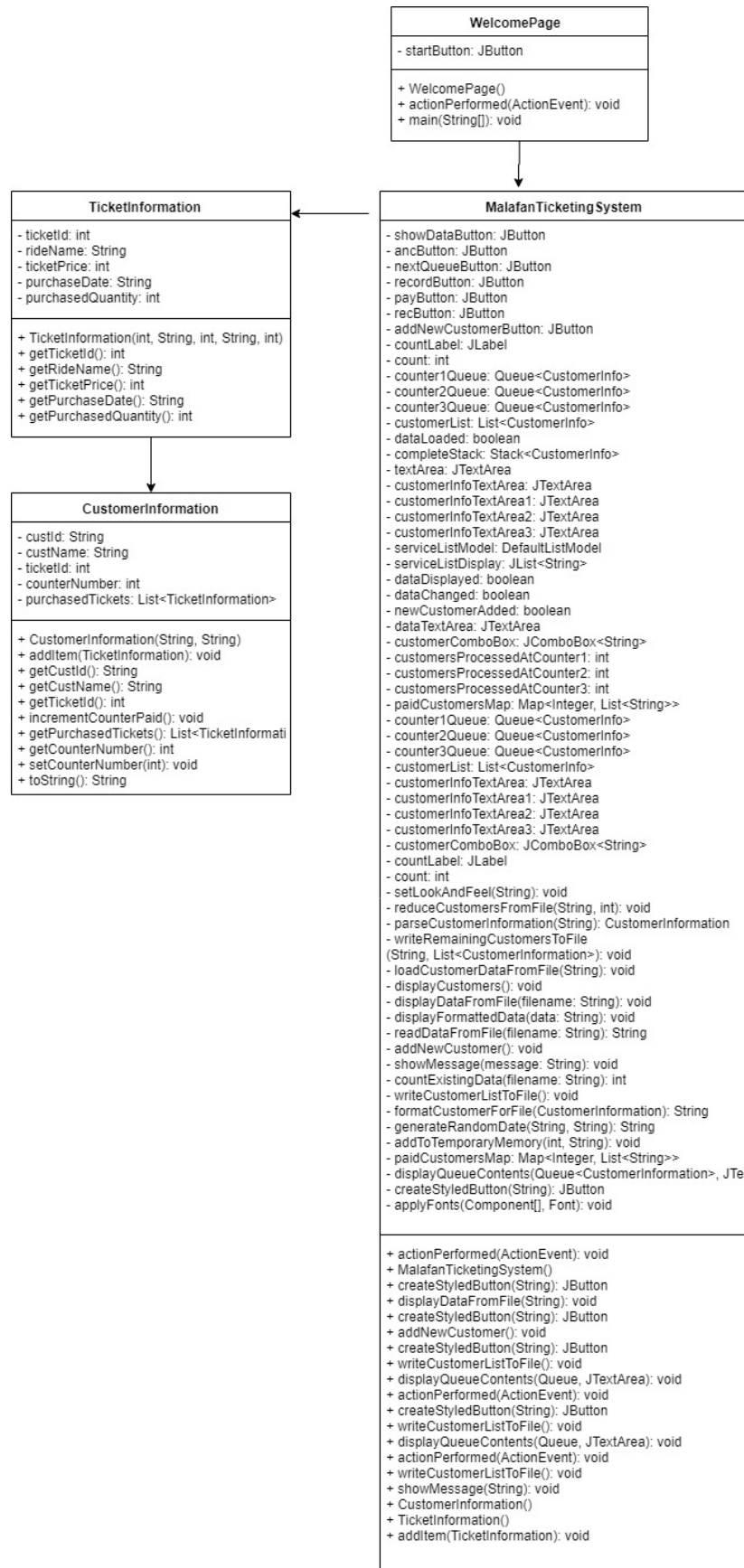
    // Returns the price of the ticket
    public int getTicketPrice() {
        return ticketPrice;
    }

    // Returns the date when the ticket was purchased
    public String getPurchaseDate() {
        return purchaseDate;
    }

    // Returns the number of tickets purchased in this transaction
    public int getPurchasedQuantity() {
        return purchasedQuantity;
    }
}
```

3.0 Design of Our ADTs using UML Class Diagram and show their relations

Design of the appropriate data structure(s) class



4.0 Java code and sample input and output.

CustomerInformation classes

```
import java.util.*;

// This class represents customer information including their
// purchased tickets
class CustomerInformation {

    // Unique customer identifier
    private String custId;

    // Name of the customer
    private String custName;

    // Represents the latest ticket ID purchased by this customer
    private int ticketId;

    // Represents the counter number where the customer makes
    // purchases
    private int counterNumber;

    // List of tickets purchased by this customer
    private List<TicketInformation> purchasedTickets;

    // Constructor to initialize customer information with ID and
    // name
    public CustomerInformation(String custId, String custName) {
        this.custId = custId;
        this.custName = custName;
        this.ticketId = 0; // Setting initial ticketId to 0
        this.purchasedTickets = new ArrayList<>();
    }

    // Adds a ticket to the list of tickets purchased by this
    // customer
    public void addItem(TicketInformation ticket) {
        purchasedTickets.add(ticket);
    }

    // Returns the customer ID
    public String getCustId() {
        return custId;
    }

    // Returns the customer name
    public String getCustName() {
        return custName;
    }

    // Returns the latest ticket ID purchased by the customer
```

```

public int getTicketId() {
    return ticketId;
}

// Increments the ticket ID when a new ticket is purchased
public void incrementCounterPaid() {
    ticketId++;
}

// Returns the list of tickets purchased by the customer
public List<TicketInformation> getPurchasedTickets() {
    return purchasedTickets;
}

// Returns the counter number associated with the customer
public int getCounterNumber() {
    return counterNumber;
}

// Sets the counter number for the customer
public void setCounterNumber(int counterNumber) {
    this.counterNumber = counterNumber;
}

// Returns a string representation of the customer, displaying
their ID and name
@Override
public String toString() {
    return custId + " - " + custName;
}
}

```

TicketInformation classes

// This class represents information about a ticket purchase for a ride.

```
class TicketInformation {

    // Unique identifier for the ticket
    private int ticketId;

    // Name of the ride associated with the ticket
    private String rideName;

    // Price of the individual ticket
    private int ticketPrice;

    // Date when the ticket was purchased
    private String purchaseDate;

    // Number of tickets purchased in this transaction
    private int purchasedQuantity;

    // Constructor to initialize all fields of the ticket
    public TicketInformation(int ticketId, String rideName, int
ticketPrice, String purchaseDate, int purchasedQuantity) {
        this.ticketId = ticketId;
        this.rideName = rideName;
        this.ticketPrice = ticketPrice;
        this.purchaseDate = purchaseDate;
        this.purchasedQuantity = purchasedQuantity;
    }

    // Returns the unique ID of the ticket
    public int getTicketId() {
        return ticketId;
    }

    // Returns the name of the ride associated with the ticket
    public String getRideName() {
        return rideName;
    }

    // Returns the price of the ticket
    public int getTicketPrice() {
        return ticketPrice;
    }

    // Returns the date when the ticket was purchased
    public String getPurchaseDate() {
        return purchaseDate;
    }

    // Returns the number of tickets purchased in this transaction
```

```
public int getPurchasedQuantity() {  
    return purchasedQuantity;  
}  
}
```

WelcomePage classes

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class WelcomePage extends JFrame implements ActionListener {
    JButton startButton;
    public WelcomePage()
    {

        // Create a title for Theme Park
        setTitle("Welcome to CELESTIAL MANAFAN FANTASY");
        setSize(700, 700);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        ImageIcon icon = new ImageIcon("icons.jpg");
        // Set the icon image for the frame
        setIconImage(icon.getImage());
        // Create image icon logo
        ImageIcon logo = null;
        try {
            logo = new ImageIcon("logo1.jpg");
        } catch (Exception e) {
            e.printStackTrace();
        }
        JLabel logoLabel = new JLabel(logo);
        // Create a Panel
        JPanel panel = new JPanel();
        panel.setLayout(new BorderLayout());
        add(panel);

        // Create Start Button
        startButton = new JButton("Start");
        startButton.addActionListener(this);
        panel.add(startButton, BorderLayout.SOUTH);
        startButton.setBackground(new Color(176, 224, 230));
        startButton.setBounds(200, 250, 100, 40);
        panel.add(logoLabel);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == startButton) {
            dispose(); // Close the WelcomePage window
            new MalafanTicketingSystem().setVisible(true); // Create
and show the ThemeParkTicketingSystem window
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
```

```
        new WelcomePage();
    }
});
}
```


MalafanTicketingSystem Classes

```
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.util.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Stack;
import java.util.Queue;
import java.util.LinkedList;
import java.util.Random;
import java.util.concurrent.ThreadLocalRandom;
import java.time.LocalDate;

public class MalafanTicketingSystem extends JFrame {

    private JButton showDataButton, ancButton, nextQueueButton,
    recordButton, payButton, recButton, addNewCustomerButton;
    private JLabel countLabel; // Add a label to display the
    counting number

    private int count = 0; // Initialize the count variable

    private Queue<CustomerInformation> counter1Queue;
    private Queue<CustomerInformation> counter2Queue;
    private Queue<CustomerInformation> counter3Queue;
    private List<CustomerInformation> customerList;
    private boolean dataLoaded = false;

    private Stack<CustomerInformation> completeStack = new
    Stack<>();

    private JTextArea textArea;
    private JTextArea customerInfoTextArea;
    private JTextArea customerInfoTextArea1;
    private JTextArea customerInfoTextArea2;
    private JTextArea customerInfoTextArea3;
    private DefaultListModel<String> serviceListModel = new
    DefaultListModel<>();
    private JList<String> serviceListDisplay;
    private boolean dataDisplayed = false;
    private boolean dataChanged = false;

    private boolean newCustomerAdded = true;

    private JTextArea dataTextArea;
```

```

private JComboBox<String> customerComboBox;

private int customersProcessedAtCounter1 = 0;
private int customersProcessedAtCounter2 = 0;
private int customersProcessedAtCounter3 = 0;

private JButton payButton1, recButton1; // For Counter 1
private JButton payButton2, recButton2; // For Counter 2
private JButton payButton3, recButton3; // For Counter 3

private Map<Integer, List<String>> paidCustomersMap = new
HashMap<>();

public MalafanTicketingSystem() {
    // Initialize the counters and customerList
    counter1Queue = new LinkedList<>();
    counter2Queue = new LinkedList<>();
    counter3Queue = new LinkedList<>();
    customerList = new ArrayList<>();

    // Initialize customerInfoTextArea
    customerInfoTextArea = new JTextArea();
    customerInfoTextArea.setEditable(false);
    customerInfoTextArea.setLineWrap(true);

    // Initialize the JTextAreas for each counter
    customerInfoTextArea1 = new JTextArea();
    customerInfoTextArea2 = new JTextArea();
    customerInfoTextArea3 = new JTextArea();

    // Initialize customerComboBox
    customerComboBox = new JComboBox<>();

    setTitle("Malafan Ticketing System");
    setSize(1300, 650);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLocationRelativeTo(null);
    setLayout(new BorderLayout(10, 10));
    getRootPane().setBorder(BorderFactory.createEmptyBorder(10,
10, 10, 10));

    // Set Nimbus look and feel
    setLookAndFeel("Nimbus");

    // Use the same color scheme as in MalafanTicketingSystem
    Color backgroundColor = new Color(223, 227, 230);
    Color buttonColor = new Color(100, 149, 237);

    // Create a panel for the top buttons and labels
    JPanel topButtonPanel = new JPanel();

```

```

        topButtonPanel.setLayout(new BorderLayout(5, 5)); // Set
        BorderLayout for topButtonPanel
        topButtonPanel.setBackground(backgroundColor);
        add(topButtonPanel, BorderLayout.NORTH);

        // Create a panel for the buttons and add it to the center
        of the topButtonPanel
        JPanel buttonRowPanel = new JPanel();
        buttonRowPanel.setLayout(new FlowLayout(FlowLayout.CENTER,
        5, 5)); // Use FlowLayout for the buttons
        buttonRowPanel.setBackground(backgroundColor); // Set the
        background color to match the top panel

        // Now add the buttonRowPanel to the center of the
        topButtonPanel
        topButtonPanel.add(buttonRowPanel, BorderLayout.CENTER);

        // Create a panel for the count label and add it to the
        topButtonPanel
        countLabel = new JLabel("Count: " + count); // Initialize
        the label with the count
        countLabel.setHorizontalAlignment(SwingConstants.CENTER); //
        Center the label horizontally
        countLabel.setFont(new Font(countLabel.getFont().getName(),
        Font.BOLD, 18)); // Set the font to be larger and bold
        topButtonPanel.add(countLabel, BorderLayout.NORTH); // Add
        the countLabel to the top of the topButtonPanel

        // Initialize and add the existing buttons
        showDataButton = createStyledButton("Show Data");
        showDataButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                displayDataFromFile("customerList.txt");
            }
        });

        // Initialize the addCustomerButton here, assigning it to
        the class-level variable
        ancButton = createStyledButton("Add New Customer");
        ancButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                addNewCustomer();
            }
        });

        nextQueueButton = createStyledButton("Next Queue");
        // nextQueueButton ActionListener
        nextQueueButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Temporary lists to hold customers based on
                their ticket quantity

```

```

        List<CustomerInformation>
customersWithFiveOrLessTickets = new ArrayList<>();
        List<CustomerInformation>
customersWithMoreThanFiveTickets = new ArrayList<>();

        // Step 1: Categorize customers based on ticket
quantity
        for (CustomerInformation customer :
customerList) {
            // Assume each customer only has one
TicketInformation object.
            if
(customer.getPurchasedTickets().get(0).getPurchasedQuantity() <= 5)
{
customersWithFiveOrLessTickets.add(customer);
            } else {

customersWithMoreThanFiveTickets.add(customer);
            }
        }

        // Step 2: Add customers to the queues based on
the logic provided
        // Add to Queue 1 (customers with ticket
quantity <= 5)
        while (counter1Queue.size() < 5 &&
!customersWithFiveOrLessTickets.isEmpty()) {
counter1Queue.add(customersWithFiveOrLessTickets.remove(0));
        }

        // Add to Queue 2 (next set of customers with
ticket quantity <= 5)
        while (counter2Queue.size() < 5 &&
!customersWithFiveOrLessTickets.isEmpty()) {
counter2Queue.add(customersWithFiveOrLessTickets.remove(0));
        }

        // Add to Queue 3 (customers with ticket
quantity > 5)
        while (counter3Queue.size() < 5 &&
!customersWithMoreThanFiveTickets.isEmpty()) {
counter3Queue.add(customersWithMoreThanFiveTickets.remove(0));
        }

        // Step 3: Update the main customer list with
remaining customers
        customerList.clear();

```

```

customerList.addAll(customersWithFiveOrLessTickets);

customerList.addAll(customersWithMoreThanFiveTickets);

// Step 4: Write the updated customer list back
to the file
writeCustomerListToFile();

// Step 5: Update GUI components to reflect the
new state of queues and customer list
displayQueueContents(counter1Queue,
customerInfoTextArea1);
displayQueueContents(counter2Queue,
customerInfoTextArea2);
displayQueueContents(counter3Queue,
customerInfoTextArea3);

// Update the count label with the new size of
the customer list
count = customerList.size();
countLabel.setText("Count: " + count);
    }
});

// Add buttons to the buttonRowPanel
buttonRowPanel.add(showDataButton);
buttonRowPanel.add(ancButton);
buttonRowPanel.add(nextQueueButton);

// Center panels
JPanel centerPanel = new JPanel();
centerPanel.setLayout(new FlowLayout(FlowLayout.CENTER, 5,
5));

centerPanel.setBackground(backgroundColor);
add(centerPanel, BorderLayout.CENTER);

// Now create the counter panels and add them to the main
frame
JPanel counterPanel1 = createCounterPanel(1,
customerInfoTextArea1, buttonColor);
JPanel counterPanel2 = createCounterPanel(2,
customerInfoTextArea2, buttonColor);
JPanel counterPanel3 = createCounterPanel(3,
customerInfoTextArea3, buttonColor);

// Assuming 'centerPanel' is the container for your counter
panels
centerPanel.add(counterPanel1);
centerPanel.add(counterPanel2);
centerPanel.add(counterPanel3);

```

```

        // Add buttons to the buttonRowPanel
        buttonRowPanel.add(showDataButton);
        buttonRowPanel.add(ancButton);
        buttonRowPanel.add(nextQueueButton);

        // Now add the buttonRowPanel to the topButtonPanel so it's
        positioned below the countLabel
        topButtonPanel.add(buttonRowPanel);

        // Record button at the bottom
        JPanel bottomButtonPanel = new JPanel(new
        FlowLayout(FlowLayout.CENTER, 5, 5));
        bottomButtonPanel.setBackground(backgroundColor);
        add(bottomButtonPanel, BorderLayout.SOUTH);

        recordButton = createStyledButton("Record");
        recordButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                displayDataFromFile("paidCustomerList.txt");
            }
        });
        recordButton.setPreferredSize(new Dimension(800, 40));
        bottomButtonPanel.add(recordButton);

        // Add New Customer button
        addNewCustomerButton = createStyledButton("Add New
        Customer");
        // Add it to the appropriate panel or layout as required

        // Set font styles
        Font buttonFont = new Font("SansSerif", Font.BOLD, 14);
        applyFonts(new Component[]{
            showDataButton, ancButton, nextQueueButton,
        recordButton,
        counter 1    payButton1, recButton1, // Assuming these are for
        counter 2    payButton2, recButton2, // Assuming these are for
        counter 3    payButton3, recButton3, // Assuming these are for
            addNewCustomerButton,
        }, buttonFont);

        addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                e.getWindow().dispose();
            }
        });
    });

```

```

        setVisible(true);

        // Load customer data from file on program start
        loadCustomerDataFromFile("customerList.txt");
    }

    // New function to reduce customers from the customerList.txt
    file
    // Modify the reduceCustomersFromFile method
    private void reduceCustomersFromFile(String fileName, int
customersToReduce) {
        // Existing code to reduce customers
        List<CustomerInformation> customersToRemove = new
ArrayList<>();

        // Identify customers to remove
        for (int i = 0; i < customersToReduce; i++) {
            if (!customerList.isEmpty()) {
                customersToRemove.add(customerList.remove(0));
            }
        }

        // Update the countLabel
        count = customerList.size();
        countLabel.setText("Count: " + count);

        // Write the updated customer list to the file
        writeCustomerListToFile();

        // Display a single message after reducing customers
        showMessage(customersToReduce + " customer(s) removed
successfully.");

        // Track the removed customers for each queue counter
        int queueCounter1Count = 0;
        int queueCounter2Count = 0;
        int queueCounter3Count = 0;

        // Distribute removed customers among the queue counters
        (max 5 for each)
        for (CustomerInformation removedCustomer :
customersToRemove) {
            if (queueCounter1Count < 5 && counter1Queue.size() < 5)
{
                counter1Queue.add(removedCustomer);
                queueCounter1Count++;
            } else if (queueCounter2Count < 5 &&
counter2Queue.size() < 5) {
                counter2Queue.add(removedCustomer);
                queueCounter2Count++;
            }
        }
    }
}

```

```

        } else if (queueCounter3Count < 5 &&
counter3Queue.size() < 5) {
            counter3Queue.add(removedCustomer);
            queueCounter3Count++;
        }
        // You can add additional logic for handling cases where
all queue counters are full
    }
}

// Helper function to parse customer information from a line
private CustomerInformation parseCustomerInformation(String
line) {
    String[] parts = line.split(";");
    if (parts.length >= 7) {
        String custId = parts[0];
        String custName = parts[1];
        int ticketId = Integer.parseInt(parts[2]);
        String rideName = parts[3];
        int ticketPrice = Integer.parseInt(parts[4]);
        int purchasedQuantity = Integer.parseInt(parts[5]);
        String purchaseDate = parts[6];

        CustomerInformation customer = new
CustomerInformation(custId, custName);
        TicketInformation ticket = new
TicketInformation(ticketId, rideName, ticketPrice, purchaseDate,
purchasedQuantity);

        for (int i = 0; i < purchasedQuantity; i++) {
            customer.addItem(ticket);
        }

        return customer;
    }
    return null;
}

// Helper function to write the remaining customers back to the
file
private void writeRemainingCustomersToFile(String filename,
List<CustomerInformation> remainingCustomers) {
    try (BufferedWriter writer = new BufferedWriter(new
FileWriter(filename, false))) {
        for (CustomerInformation customer : remainingCustomers)
        {
            String customerData =
formatCustomerForFile(customer);
            writer.write(customerData);
            writer.newLine();
        }
    }
}

```



```

        } catch (IOException e) {
            JOptionPane.showMessageDialog(this, "Error writing
customer list: " + e.getMessage());
        }
    }

    private void setLookAndFeel(String lookAndFeelName) {
        try {
            for (UIManager.LookAndFeelInfo info :
UIManager.getInstalledLookAndFeels()) {
                if (lookAndFeelName.equals(info.getName())) {
                    UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (Exception e) {
            try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }

    // Modify loadCustomerDataFromFile method to update count and
countLabel
    private void loadCustomerDataFromFile(String filename) {
        customerList.clear(); // Clear existing data

        try (BufferedReader reader = new BufferedReader(new
FileReader(filename))) {
            String line;

            while ((line = reader.readLine()) != null) {
                CustomerInformation customer =
parseCustomerInformation(line);
                if (customer != null) {
                    customerList.add(customer);
                }
            }

            // Update the count and countLabel after loading the
data

            count = customerList.size();
            countLabel.setText("Count: " + count);
            JOptionPane.showMessageDialog(this, "Customer data
loaded successfully.");
        } catch (IOException e) {
            JOptionPane.showMessageDialog(this, "Error loading
customer data: " + e.getMessage());
        }
    }

```

```

    }
}

private void displayCustomers() {
    serviceListModel.clear();
    customerInfoTextArea.setText("");
    customerComboBox.removeAllItems();
    customerComboBox.addItem("Select a customer");

    int counter = 0;
    int counterLimit = 5;
    String currentCounter = "";
    String currentCounterMain = "";

    for (CustomerInformation customer : completeStack) {
        serviceListModel.addElement(customer.toString());
        customerInfoTextArea.append("Customer ID: " +
customer.getCustId() + "\n");
        customerInfoTextArea.append("Customer Name: " +
customer.getCustName() + "\n");
        customerInfoTextArea.append("Ticket ID: " +
(customer.getTicketId() - 1)+ "\n");
        customerInfoTextArea.append("Counter Number: " +
customer.getCounterNumber() + "\n\n");
        customerComboBox.addItem(customer.toString());

        counter++;
    }
}

private void displayDataFromFile(String filename) {
    String data = readDataFromFile(filename);
    if (!data.isEmpty()) {
        displayFormattedData(data);
    }
}

private void displayFormattedData(String data) {
    JTextArea dataTextArea = new JTextArea(data);
    dataTextArea.setEditable(false);
    dataTextArea.setLineWrap(true);
    dataTextArea.setWrapStyleWord(true);
    JScrollPane dataScrollPane = new JScrollPane(dataTextArea);
    dataScrollPane.setPreferredSize(new Dimension(400, 600));

    dataTextArea.setFont(new Font("Monospaced", Font.PLAIN,
12));

    dataTextArea.setMargin(new Insets(10, 10, 10, 10));
    dataTextArea.setOpaque(false);
    dataTextArea.setForeground(Color.BLACK);
    dataTextArea.setBackground(Color.WHITE);

```

```

        JOptionPane.showMessageDialog(this, dataScrollPane,
"Customer Data", JOptionPane.INFORMATION_MESSAGE);
    }

    private String readDataFromFile(String filename) {
        try {
            BufferedReader reader = new BufferedReader(new
FileReader(filename));
            StringBuilder data = new StringBuilder();
            String line;

            while ((line = reader.readLine()) != null) {
                String[] parts = line.split(";");
                if (parts.length >= 7) {
                    String custId = parts[0];
                    String custName = parts[1];
                    int ticketId = Integer.parseInt(parts[2]);
                    String rideName = parts[3];
                    int ticketPrice = Integer.parseInt(parts[4]);
                    int purchasedQuantity =
Integer.parseInt(parts[5]);
                    String purchaseDate = parts[6];

                    data.append("Customer ID:
").append(custId).append("\n");
                    data.append("Customer Name:
").append(custName).append("\n");
                    data.append("Ticket ID:
").append(ticketId).append("\n");
                    data.append("Ride Name:
").append(rideName).append("\n");
                    data.append("Ticket Price:
RM").append(ticketPrice).append("\n");
                    data.append("Purchased Quantity:
").append(purchasedQuantity).append("\n");
                    data.append("Purchase Date:
").append(purchaseDate).append("\n");

                    data.append("=====").append("\n");
                }
            }

            reader.close();
            return data.toString();
        } catch (IOException e) {
            JOptionPane.showMessageDialog(this, "Error loading data:
" + e.getMessage());
            return "";
        }
    }
}

```

```

// New function to add a new customer
private void addNewCustomer() {
    // Define the maximum limit for data entries
    int maxDataLimit = 100;

    // Read the existing data from the file and count the
entries
    int existingDataCount =
countExistingData("customerList.txt");

    // Calculate the number of customers needed to reach the max
limit
    int customersNeeded = maxDataLimit - existingDataCount;

    // Check if the data is full
    if (existingDataCount >= maxDataLimit) {
        JOptionPane.showMessageDialog(this, "Data is full.
Cannot add more customers.");
        return; // Exit the function if data is full
    }

    Random random = new Random();
    // Instead of directly writing to the file, you should add
the new customers to the in-memory list first.
    List<CustomerInformation> newCustomers = new ArrayList<>();

    // Find the last customer ID and ticket ID from the
customerList collection instead of reading the file each time
    int lastCustomerId = customerList.isEmpty() ? 0 :
Integer.parseInt(customerList.get(customerList.size() -
1).getCustId().substring(2));
    int lastTicketId = customerList.isEmpty() ? 0 :
customerList.get(customerList.size() - 1).getTicketId();

    // Generate and add new customers in a loop
    for (int i = 0; i < customersNeeded; i++) {

        try (BufferedReader reader = new BufferedReader(new
FileReader("customerList.txt"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] parts = line.split(";");
                if (parts.length >= 2) {
                    int customerId =
Integer.parseInt(parts[0].substring(2));
                    int ticketId = Integer.parseInt(parts[2]);
                    lastCustomerId = Math.max(lastCustomerId,
customerId);
                    lastTicketId = Math.max(lastTicketId,
ticketId);
                }
            }
        }
    }
}

```

```

        }
    }
} catch (IOException e) {
    e.printStackTrace();
    JOptionPane.showMessageDialog(this, "Error reading
customer data: " + e.getMessage());
}

// Increment the last customer and ticket IDs
int newCustomerIdNumber = lastCustomerId + 1;
int newTicketIdNumber = lastTicketId + 1;

// Generate new customer and ticket IDs by formatting
them to your desired format
String newCustomerId = String.format("TP%04d",
newCustomerIdNumber);
String newRidesId = String.format("%03d",
newTicketIdNumber);

// Generate a random customer name
String[] customerNames1 = {"Muhammad", "Nurul", "Dina",
"Siti", "Nor", "Aminah", "Hassan", "Fatimah", "Zain", "Zara",
"Imran", "Aisyah", "Khalid", "Zainab", "Hussein", "Raihana", "Ali",
"Maznah", "Salim", "Zulaikha"};
String[] customerNames2 = {"Abdullah", "Mohd", "Ali",
"Ahmad", "Hakim", "Abu", "Ibrahim", "Syafiq", "Ismail", "Mohd",
"Aisyah", "Hassan", "Izzati", "Amir", "Hakeem", "Nadia", "Farah",
"Dila", "Daniel", "Akmal"};
String randomCustomerName =
customerNames1[random.nextInt(customerNames1.length)] + " " +
customerNames2[random.nextInt(customerNames2.length)];

// Generate a random ride name
String[] rideNames = {"Roller Coaster", "Ferris Wheel",
"Carousel", "Haunted House", "Tea Cups", "Bumper Cars", "Water
Slide", "Swing Ride", "Merry-Go-Round"};
String randomRideName =
rideNames[random.nextInt(rideNames.length)];

// Generate a random price based on the ride name
int randomPrice = 0;
switch (randomRideName) {
    case "Roller Coaster":
        randomPrice = 20;
        break;
    case "Ferris Wheel":
        randomPrice = 15;
        break;
    case "Carousel":
        randomPrice = 10;
        break;
}

```

```

        case "Haunted House":
            randomPrice = 25;
            break;
        case "Tea Cups":
            randomPrice = 12;
            break;
        case "Bumper Cars":
            randomPrice = 18;
            break;
        case "Water Slide":
            randomPrice = 30;
            break;
        case "Swing Ride":
            randomPrice = 15;
            break;
        case "Merry-Go-Round":
            randomPrice = 10;
            break;
    }

    int randomQuantity = random.nextInt(10) + 1;
    String randomPurchaseDate = generateRandomDate("2024-01-
21", "2025-01-01");

    String customerData = newCustomerId + ";" +
randomCustomerName + ";" + newRidesId + ";" + randomRideName + ";" +
randomPrice + ";" + randomQuantity + ";" + randomPurchaseDate;

    try (BufferedWriter writer = new BufferedWriter(new
FileWriter("customerList.txt", true))) {
        writer.write(customerData);
        writer.newLine();
    } catch (IOException e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(this, "Error adding a
new customer: " + e.getMessage());
    }
    // Create a new CustomerInformation object
    CustomerInformation newCustomer = new
CustomerInformation(newCustomerId, randomCustomerName);
    TicketInformation newTicket = new
TicketInformation(newTicketIdNumber, randomRideName, randomPrice,
randomPurchaseDate, randomQuantity);
    newCustomer.addItem(newTicket);

    newCustomer.addItem(newTicket);

    // Add the new customer to the in-memory list
    newCustomers.add(newCustomer);

    // Update the last IDs used

```

```

        lastCustomerId = newCustomerIdNumber;
        lastTicketId = newTicketIdNumber;
    }
    // Add the newly created customers to the main customer list
    customerList.addAll(newCustomers);

    // Now write the entire customer list to the file
    writeCustomerListToFile();

    // Display a single message after adding all the generated
customers
    showMessage(customersNeeded + " New customer added
successfully.");
    newCustomerAdded = true;

    // Update the countLabel
    count = customerList.size(); // Assuming customerList is
your in-memory list
    countLabel.setText("Count: " + count);
}

// Helper method to add receipt to file
private void addReceiptToFile(String filename, String receipt) {
    // No change needed if you're just appending the formatted
string to the file
    try (BufferedWriter writer = new BufferedWriter(new
FileWriter(filename, true))) {
        writer.write(receipt);
        writer.newLine();
    } catch (IOException e) {
        JOptionPane.showMessageDialog(this, "Error writing to
receipt file: " + e.getMessage());
    }
}

private void showMessage(String message) {
    JOptionPane.showMessageDialog(this, message);
}

// Helper function to count existing data entries in the file
private int countExistingData(String filename) {
    try {
        BufferedReader reader = new BufferedReader(new
FileReader(filename));
        int count = 0;

        while (reader.readLine() != null) {
            count++;
        }

        reader.close();
    }
}

```

```

        return count;
    } catch (IOException e) {
        JOptionPane.showMessageDialog(this, "Error counting
existing data: " + e.getMessage());
        return 0;
    }
}

private void writeCustomerListToFile() {
    try (BufferedWriter writer = new BufferedWriter(new
FileWriter("customerList.txt"))) {
        for (CustomerInformation customer : customerList) {
            String customerData =
formatCustomerForFile(customer);
            writer.write(customerData);
            writer.newLine();
        }
    } catch (IOException e) {
        JOptionPane.showMessageDialog(this, "Error writing
customer list: " + e.getMessage());
    }
}

private String formatCustomerForFile(CustomerInformation
customer) {
    // Get the last purchased ticket from the list of purchased
tickets
    List<TicketInformation> purchasedTickets =
customer.getPurchasedTickets();
    TicketInformation lastTicket =
purchasedTickets.get(purchasedTickets.size() - 1);

    return String.format("%s;%s;%d;%s;%d;%d;%s",
        customer.getCustId(),
        customer.getCustName(),
        lastTicket.getTicketId(),
        lastTicket.getRideName(),
        lastTicket.getTicketPrice(),
        lastTicket.getPurchasedQuantity(),
        lastTicket.getPurchaseDate()
    );
}

// Helper function to generate a random date between two dates
private String generateRandomDate(String startDate, String
endDate) {
    LocalDate start = LocalDate.parse(startDate);
    LocalDate end = LocalDate.parse(endDate);
    long randomDate =
ThreadLocalRandom.current().nextLong(start.toEpochDay(),
end.toEpochDay());

```



```

        return LocalDate.ofEpochDay(randomDate).toString();
    }

    private JPanel createCounterPanel(int counterNumber, JTextArea
customerInfoTextArea, Color buttonColor) {
        JPanel counterPanel = new JPanel(new BorderLayout());
        counterPanel.setBackground(new Color(223, 227, 230));

        JLabel counterLabel = new JLabel("<html><div style='text-
align: center; font-size: 16px;'>Counter " + counterNumber +
"</div></html>");
        counterLabel.setHorizontalAlignment(SwingConstants.CENTER);
        counterPanel.add(counterLabel, BorderLayout.NORTH);

        JButton ancButton = createStyledButton("ANC");

        JPanel buttonPanel = new JPanel(new
FlowLayout(FlowLayout.CENTER, 5, 5));
        buttonPanel.setBackground(new Color(223, 227, 230));
        buttonPanel.add(ancButton);

        counterPanel.add(buttonPanel, BorderLayout.CENTER);

        customerInfoTextArea.setEditable(false);
        customerInfoTextArea.setLineWrap(true);
        JScrollPane customerInfoScrollPane = new
JScrollPane(customerInfoTextArea);
        customerInfoScrollPane.setPreferredSize(new Dimension(250,
300));
        counterPanel.add(customerInfoScrollPane,
BorderLayout.CENTER);
        payButton = createStyledButton("Payment");
        payButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Queue<CustomerInformation> queue = null;
                switch (counterNumber) {
                    case 1:
                        payButton1 =
createStyledButton("Payment");
                        recButton1 =
createStyledButton("Receipt");
                        queue = counter1Queue;
                        // Set any additional properties or
listeners specific to counter 1
                        break;
                    case 2:
                        payButton2 =
createStyledButton("Payment");
                        recButton2 =
createStyledButton("Receipt");
                        queue = counter2Queue;

```

```

        // Set any additional properties or
listeners specific to counter 2
        break;
        case 3:
            payButton3 =
createStyledButton("Payment");
            recButton3 =
createStyledButton("Receipt");
            queue = counter3Queue;
            // Set any additional properties or
listeners specific to counter 3
            break;
    }
    if (queue != null && !queue.isEmpty()) {
        CustomerInformation customer = queue.poll();
// Remove the first customer from the queue
        String receipt = generateReceipt(customer);
// Generate the receipt for the customer
        addReceiptToFile("paidCustomerList.txt",
receipt); // Add the receipt to paidCustomerList.txt
        addToTemporaryMemory(counterNumber,
receipt); // Add the receipt to temporary memory
        displayQueueContents(queue,
customerInfoTextArea); // Update the display
    } else {

JOptionPane.showMessageDialog(MalafanTicketingSystem.this, "No
customers in queue.");
    }
    // Now apply the bold font to these buttons
specifically
    Font buttonFont = new Font("SansSerif",
Font.BOLD, 14);
    applyFonts(new Component[]{payButton1,
recButton1, payButton2, recButton2, payButton3, recButton3},
buttonFont);
    }
    });

    // Inside the createCounterPanel method, modify the
recButton's ActionListener
    recButton = createStyledButton("Receipt");
    recButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            List<String> receipts =
paidCustomersMap.getDefault(counterNumber, new ArrayList<>());
            displayReceipts(receipts, counterNumber); //
Pass the counter number if needed for unique formatting
        }
    });

```

```

        JPanel bottomButtonPanel = new JPanel(new
FlowLayout(FlowLayout.CENTER, 5, 5));
        bottomButtonPanel.setBackground(new Color(223, 227, 230));
        bottomButtonPanel.add(payButton);
        bottomButtonPanel.add(recButton);

        counterPanel.add(bottomButtonPanel, BorderLayout.SOUTH);

        // Display the content of the respective counter queue in
the JTextArea
        switch (counterNumber) {
            case 1:
                displayQueueContents(counter1Queue,
customerInfoTextArea);
                break;
            case 2:
                displayQueueContents(counter2Queue,
customerInfoTextArea);
                break;
            case 3:
                displayQueueContents(counter3Queue,
customerInfoTextArea);
                break;
        }

        return counterPanel;
    }

    // Add the new displayReceipts method
    private void displayReceipts(List<String> receipts, int
counterNumber) {
        // Create a JTextArea to display the receipts
        JTextArea receiptTextArea = new JTextArea(20, 40);
        receiptTextArea.setEditable(false);
        receiptTextArea.setMargin(new Insets(5, 5, 5, 5));
        receiptTextArea.setFont(new Font("Segoe UI", Font.PLAIN,
12));

        // Format each receipt and append it to the JTextArea
        for (String receipt : receipts) {
            // Split the receipt data to separate it out from the
tokenizer format
            String[] receiptData = receipt.split(";");
            // Check if the receiptData array has the expected
number of elements
            if (receiptData.length == 7) {
                receiptTextArea.append("Customer ID: " +
receiptData[0] + "\n");
                receiptTextArea.append("Customer Name: " +
receiptData[1] + "\n");
            }
        }
    }

```

```

        receiptTextArea.append("Ticket ID: " +
receiptData[2] + "\n");
        receiptTextArea.append("Ride Name: " +
receiptData[3] + "\n");
        receiptTextArea.append("Ticket Price: " +
receiptData[4] + "\n");
        receiptTextArea.append("Quantity Purchased: " +
receiptData[5] + "\n");
        receiptTextArea.append("Purchase Date: " +
receiptData[6] + "\n");
        receiptTextArea.append("-----
-----\n");
    }
}

// Wrap the JTextArea in a JScrollPane
JScrollPane scrollPane = new JScrollPane(receiptTextArea);

scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR
_ALWAYS);

// Show the dialog
JOptionPane.showMessageDialog(null, scrollPane, "Receipts
for Counter " + counterNumber, JOptionPane.INFORMATION_MESSAGE);
}

// Helper method to add receipt to temporary memory
private void addToTemporaryMemory(int counterNumber, String
receipt) {
    List<String> receipts =
paidCustomersMap.getOrDefault(counterNumber, new ArrayList<>());
    receipts.add(receipt);
    paidCustomersMap.put(counterNumber, receipts);
}

// Helper method to generate a receipt for a customer
private String generateReceipt(CustomerInformation customer) {
    // Assuming customer has a method to get all purchased
tickets
    List<TicketInformation> tickets =
customer.getPurchasedTickets();
    // Retrieve the last purchased ticket
TicketInformation lastTicket = tickets.get(tickets.size() -
1);

    // Format the receipt as per the requirement
return String.format("%s;%s;%d;%s;%d;%d;%s",
        customer.getCustId(),
        customer.getCustName(),
        lastTicket.getTicketId(),
        lastTicket.getRideName(),

```

```

        lastTicket.getTicketPrice(),
        lastTicket.getPurchasedQuantity(),
        lastTicket.getPurchaseDate()
    );
}

// Helper method to display receipts
private void displayReceipts(List<String> receipts) {
    StringBuilder allReceipts = new StringBuilder();
    for (String receipt : receipts) {
        allReceipts.append(receipt).append("\n");
    }
    // You might want to use a JTextArea inside a JScrollPane if
the receipts are many
    JOptionPane.showMessageDialog(this, allReceipts.toString(),
"Receipts", JOptionPane.INFORMATION_MESSAGE);
}

private void displayQueueContents(Queue<CustomerInformation>
queue, JTextArea textArea) {
    textArea.setText("");
    textArea.append("Queue Contents:\n");

textArea.append("=====\n");

    int customerNumber = 1;
    for (CustomerInformation customer : queue) {
        textArea.append(customerNumber + "." + "\t" + "Customer
ID: " + customer.getCustId() + "\n");
        textArea.append("\t" + "Customer Name: " +
customer.getCustName() + "\n");
        textArea.append("\t" + "Ticket Quantity: " +
customer.getPurchasedTickets().get(0).getPurchasedQuantity() +
"\n");

textArea.append("=====\n");

        customerNumber++;
    }
}

private JButton createStyledButton(String text) {
    JButton button = new JButton(text);
    button.setPreferredSize(new Dimension(200, 40));
    button.setBackground(new Color(176, 224, 230));
    button.setForeground(Color.BLACK);
    button.setBorder(BorderFactory.createLineBorder(new
Color(135, 206, 235), 2));
    return button;
}

```

```

    }

    private void applyFonts(Component[] components, Font font) {
        for (Component component : components) {
            if (component != null) {
                component.setFont(font);
            }
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            new MalafanTicketingSystem();
        });
    }
}

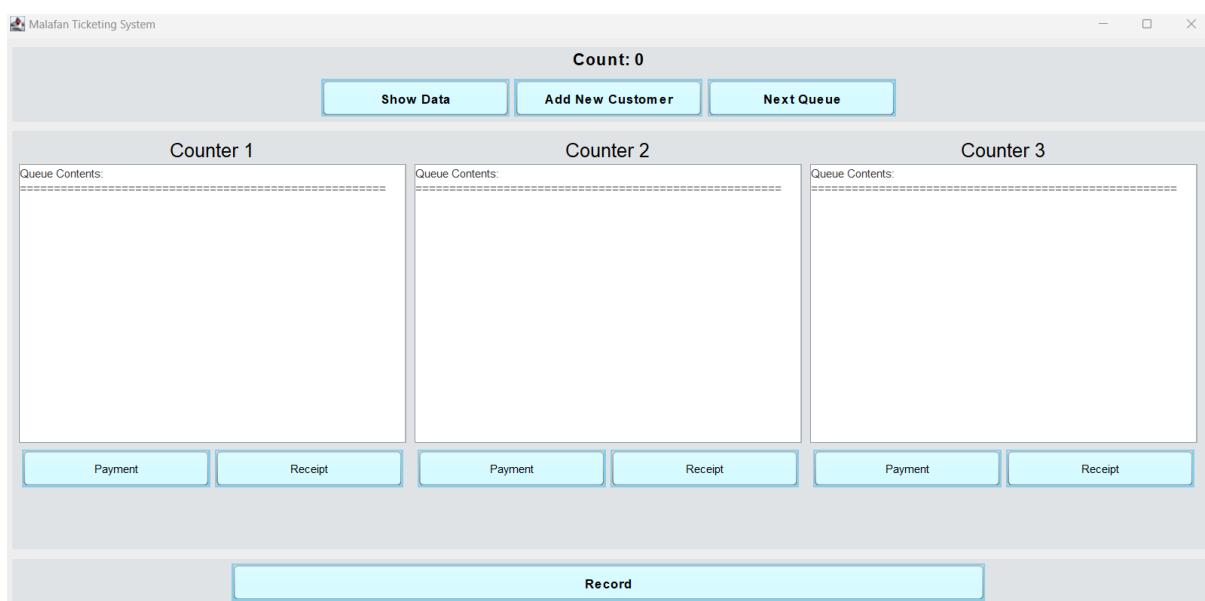
```

5.0 Progarm Output

WelcomePage Input



Must click button "Start" to coutinue to next part.



This frame will be display after the WelcomePage.

GUI INPUT

Malafan Ticketing System

Count: 0

Show Data Add New Customer Next Queue

Counter 1 Counter 2 Counter 3

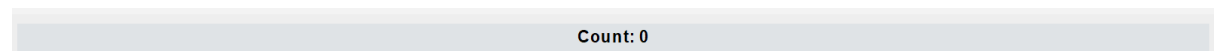
Queue Contents: Queue Contents: Queue Contents:

Payment Receipt Payment Receipt Payment Receipt

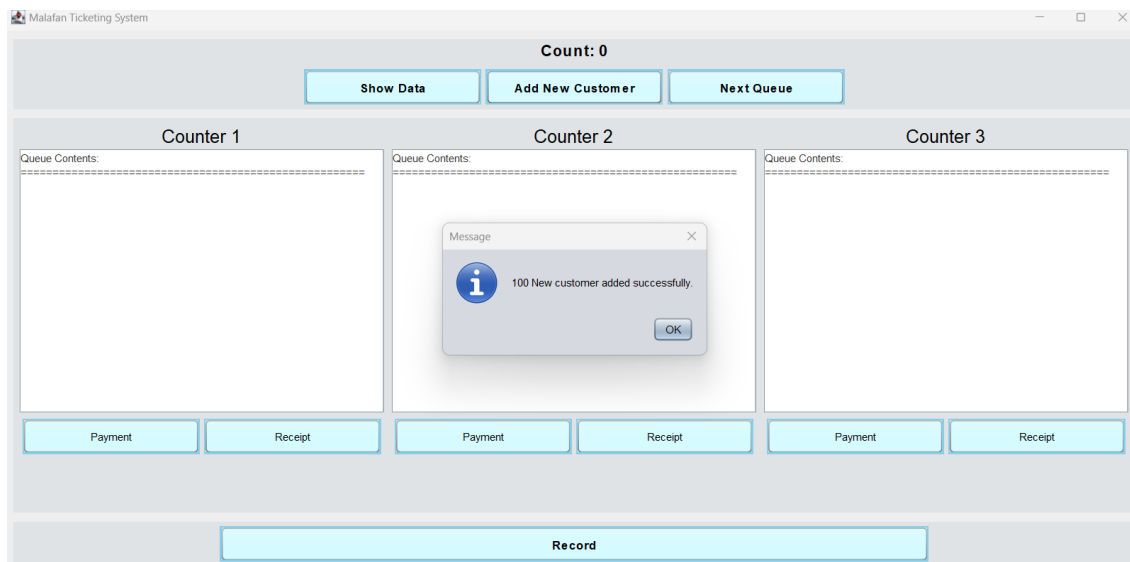
Record

- This is our GUI program for the ticket and counter system
- They have 3 button at the top and have a own function for each button
- The button is “Show Data”, “Add New Customer” and “Next Queue” button
- We have 3 panel that is for our Counter1, Counter2 and Counter3.
- For each Counter below have their own “Payment” and “Receipt” button.
- Last is the Record button

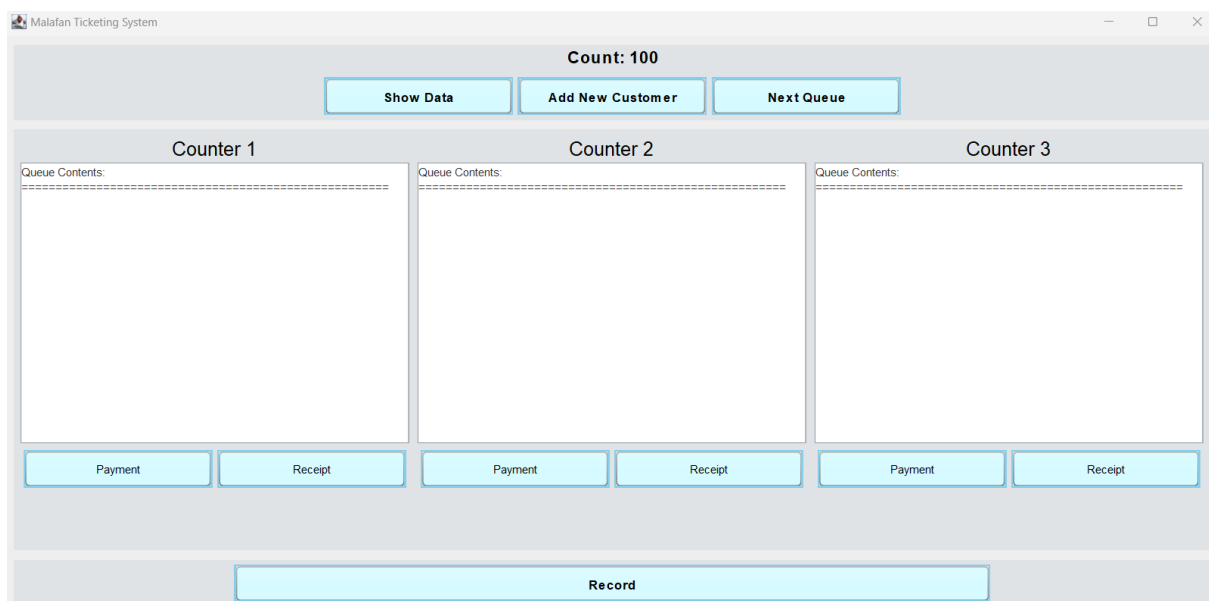
Add New Customer



- This program cannot be run because did not have any data that been add into the queue
- if we click the “Next Queue” button, there will no any data be add at the each Counter and the Count will only display “0”.



- When we click the “Add New Customer” button, the 100 data for the customer will be add to the system and match to our Maxlimit that we already set that is “100” only for the count section
- There will pop up a message said “100 New customer added successfully” so this show the button work greatly
- The data we get it from customerList.txt



- The count show the Count change from 0 to 100.
- So right now we can continue and proceed to use other buttons.

Show Data



- When we click the "Show Data" button, It will show the 100 customers that have been add just now and this data also get from customerList.txt.
- We can scroll to see all the 100 customers details that have been add.

Next Queue

The screenshot displays the Malafan Ticketing System interface. At the top, a status bar shows "Count: 85". Below this are three buttons: "Show Data", "Add New Customer", and "Next Queue". The main area is divided into three columns, each representing a counter. Each counter has a "Queue Contents" list and two buttons: "Payment" and "Receipt".

Counter 1	Counter 2	Counter 3
1. Customer ID: TP0001 Customer Name: Nor Ali Ticket Quantity: 2	1. Customer ID: TP0013 Customer Name: Zain Abu Ticket Quantity: 3	1. Customer ID: TP0002 Customer Name: Aminah Abdullah Ticket Quantity: 8
2. Customer ID: TP0004 Customer Name: Khalid Farah Ticket Quantity: 5	2. Customer ID: TP0018 Customer Name: Zainab Ali Ticket Quantity: 3	2. Customer ID: TP0003 Customer Name: Maznah Aisyah Ticket Quantity: 9
3. Customer ID: TP0005 Customer Name: Zulaikha Syafiq Ticket Quantity: 2	3. Customer ID: TP0020 Customer Name: Maznah Hakeem Ticket Quantity: 5	3. Customer ID: TP0006 Customer Name: Salim Dila Ticket Quantity: 6
4. Customer ID: TP0008 Customer Name: Imran Ismail Ticket Quantity: 5	4. Customer ID: TP0022 Customer Name: Nurul Mohd Ticket Quantity: 5	4. Customer ID: TP0007 Customer Name: Aminah Dila Ticket Quantity: 7

At the bottom of the interface is a large "Record" button.

- "nextQueue" button is working by taking data from customerList.txt and separate it into three counter queue, each counter have their own queue that called counter1Queue, counter2Queue, counter3Queue.
- Each queue counter can only hold 5 customer at a time, if all the queue counter is full of customer, then it cannot add another new customer inside queue counter by click "nextQueue" button.

Counter 1,2 and 3

- Basically this is a process where the counter handle the ticket and separate customer by assigned ticket quantity.

- Each counter have their own “Pay” and “Receipt” button that can proses like a real Counter situation at market

The screenshot displays the 'Malafan Ticketing System' window. At the top, a status bar shows 'Count: 85'. Below this are three buttons: 'Show Data', 'Add New Customer', and 'Next Queue'. The main area is divided into three columns for 'Counter 1', 'Counter 2', and 'Counter 3'. Each counter has a 'Queue Contents' list with four entries, each showing a customer ID, name, and ticket quantity. Below each queue list are 'Payment' and 'Receipt' buttons. At the bottom of the window is a large 'Record' button.

Counter	Queue Contents
Counter 1	<ul style="list-style-type: none">1. Customer ID: TP0001, Customer Name: Nor Ali, Ticket Quantity: 22. Customer ID: TP0004, Customer Name: Khalid Farah, Ticket Quantity: 53. Customer ID: TP0005, Customer Name: Zulaikha Syafiq, Ticket Quantity: 24. Customer ID: TP0008, Customer Name: Imran Ismail, Ticket Quantity: 5
Counter 2	<ul style="list-style-type: none">1. Customer ID: TP0013, Customer Name: Zain Abu, Ticket Quantity: 32. Customer ID: TP0018, Customer Name: Zainab Ali, Ticket Quantity: 33. Customer ID: TP0020, Customer Name: Maznah Hakeem, Ticket Quantity: 54. Customer ID: TP0022, Customer Name: Nurul Mohd, Ticket Quantity: 5
Counter 3	<ul style="list-style-type: none">1. Customer ID: TP0002, Customer Name: Aminah Abdullah, Ticket Quantity: 82. Customer ID: TP0003, Customer Name: Maznah Aisyah, Ticket Quantity: 93. Customer ID: TP0006, Customer Name: Salim Dila, Ticket Quantity: 64. Customer ID: TP0007, Customer Name: Aminah Dila, Ticket Quantity: 7

Counter 1

- only process customer that have the ticket quantity same or less then 5
- when we click “Payment” button,it will take the data from Queue Contents number 1 first in lists for **Counter 1** because we used The FIFO (First-In, First-Out) method in this coding

OUTPUT:

Counter 1

Queue Contents:

1.	Customer ID: TP0001 Customer Name: Nor Ali Ticket Quantity: 2
2.	Customer ID: TP0004 Customer Name: Khalid Farah Ticket Quantity: 5
3.	Customer ID: TP0005 Customer Name: Zulaikha Syafiq Ticket Quantity: 2
4.	Customer ID: TP0008 Customer Name: Imran Ismail Ticket Quantity: 5

Payment Receipt

Counter 1

Queue Contents:

1.	Customer ID: TP0004 Customer Name: Khalid Farah Ticket Quantity: 5
2.	Customer ID: TP0005 Customer Name: Zulaikha Syafiq Ticket Quantity: 2
3.	Customer ID: TP0008 Customer Name: Imran Ismail Ticket Quantity: 5
4.	Customer ID: TP0009 Customer Name: Khalid Hakim Ticket Quantity: 5

Payment Receipt

- if we want to see the Customer that already paid for the **Counter 1** ,we must click “Receipt” button
- and it will display the output from **Counter 1**
- the data is temporary and also it will display the data from before this **Counter 1**

OUTPUT:

Receipts for Counter 1

Ticket Price: 10
Quantity Purchased: 5
Purchase Date: 2024-10-11

Customer ID: TP0099
Customer Name: Aisyah Syafiq
Ticket ID: 99
Ride Name: Ferris Wheel
Ticket Price: 15
Quantity Purchased: 4
Purchase Date: 2024-04-22

Customer ID: TP0001
Customer Name: Nor Ali
Ticket ID: 1
Ride Name: Bumper Cars
Ticket Price: 18
Quantity Purchased: 2
Purchase Date: 2024-04-18

OK

Counter 2

- only process customer that have the ticket quantity same or less then 5
- when we click “Payment” button,it will take the data from Queue Contents number 1 first in lists for **Counter 2** because we used The FIFO (First-In, First-Out) method in this coding

OUTPUT:

Counter 2

Queue Contents:

1.	Customer ID: TP0013 Customer Name: Zain Abu Ticket Quantity: 3
2.	Customer ID: TP0018 Customer Name: Zainab Ali Ticket Quantity: 3
3.	Customer ID: TP0020 Customer Name: Maznah Hakeem Ticket Quantity: 5
4.	Customer ID: TP0022 Customer Name: Nurul Mohd Ticket Quantity: 5

Payment Receipt

Counter 2

Queue Contents:

1.	Customer ID: TP0018 Customer Name: Zainab Ali Ticket Quantity: 3
2.	Customer ID: TP0020 Customer Name: Maznah Hakeem Ticket Quantity: 5
3.	Customer ID: TP0022 Customer Name: Nurul Mohd Ticket Quantity: 5
4.	Customer ID: TP0023 Customer Name: Zulaikha Ali Ticket Quantity: 2

Payment Receipt

- if we want to see the Customer that already paid for the **Counter 2** ,we must click “Receipt” button
- and it will display the output from **Counter 2**
- the data is temporary and also it will display the data from before this **Counter 2**

OUTPUT:

Receipts for Counter 2

Info

Ride Name: Haunted House
Ticket Price: 25
Quantity Purchased: 4
Purchase Date: 2024-09-15

Customer ID: TP0098
Customer Name: Muhammad Hakeem
Ticket ID: 98
Ride Name: Haunted House
Ticket Price: 25
Quantity Purchased: 5
Purchase Date: 2024-06-07

Customer ID: TP0013
Customer Name: Zain Abu
Ticket ID: 13
Ride Name: Roller Coaster
Ticket Price: 20
Quantity Purchased: 3
Purchase Date: 2024-10-02

OK

Counter 3

However for **Counter 3** little bit different because only for customer that have the ticket quantity more than 5.

- only for customer that have the ticket quantity more than 5.
- when we click “Payment” button, it will take the data from Queue Contents number 1 first in lists for **Counter 3** because we used The FIFO (First-In, First-Out) method in this coding

OUTPUT:

Counter 3

Queue Contents:

1.	Customer ID: TP0002 Customer Name: Aminah Abdullah Ticket Quantity: 8
2.	Customer ID: TP0003 Customer Name: Maznah Aisyah Ticket Quantity: 9
3.	Customer ID: TP0006 Customer Name: Salim Dila Ticket Quantity: 6
4.	Customer ID: TP0007 Customer Name: Aminah Dila Ticket Quantity: 7

Payment Receipt

Counter 3

Queue Contents:


1.	Customer ID: TP0003 Customer Name: Maznah Aisyah Ticket Quantity: 9
2.	Customer ID: TP0006 Customer Name: Salim Dila Ticket Quantity: 6
3.	Customer ID: TP0007 Customer Name: Aminah Dila Ticket Quantity: 7
4.	Customer ID: TP0010 Customer Name: Hassan Syafiq Ticket Quantity: 7

Payment Receipt

- if we want to see the Customer that already paid for the **Counter 3**, we must click “Receipt” button
- and it will display the output from **Counter 3**
- the data is temporary and also it will display the data from before this **Counter 3**

OUTPUT:

Receipts for Counter 3



Ride Name: Bumper Cars
Ticket Price: 18
Quantity Purchased: 6
Purchase Date: 2024-12-26

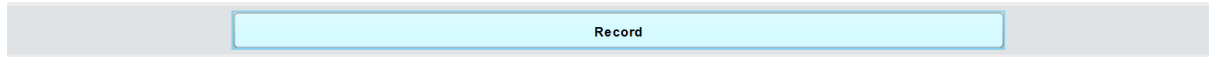
Customer ID: TP0100
Customer Name: Nor Ali
Ticket ID: 100
Ride Name: Merry-Go-Round
Ticket Price: 10
Quantity Purchased: 10
Purchase Date: 2024-08-31

Customer ID: TP0002
Customer Name: Aminah Abdullah
Ticket ID: 2
Ride Name: Bumper Cars
Ticket Price: 18
Quantity Purchased: 8
Purchase Date: 2024-01-27

OK

Record

-This button is function to focus to display all Receipt from all Counter(Counter1,Counter2 and Counter3)



- click at the “Record” button

OUTPUT:



6.0 Lessons learned, reflective experience towards completing the individual/overall tasks and Conclusion

(Afiq)

After finishing this group project there were a lot lesson that I learn. The first one was I learn about queue and stack. I know how to manage and control the order of processing data. This knowledge are very important because this FIFO(First In First Out) and LIFO(Last In First Out) principles are essential in many real-world scenarios. As example, queue can be likened to lines at grocery store or them park ticketing system as we do in this project. In my part in this project, I create a method that called "readDataFile" to read all the data from the "customerList" and display it to the user using StringBuilder. The details that will be displayed is customer id, customer name, ticket id, ride name, ticket price, purchased quantity and purchased date. For this task, there is a problem or error occurred in the coding. My code does not read the data properly and there was a time that the data is not displayed at all. The solution for this problem, I look back at my previous task lesson to refer how to read a data properly. Then I found a solution where my coding doesn't read on the correct order. Also my primitive data type is wrong that cause an error. Finally after fixing my code, all the data can be read and display correctly. After that I do in the next part which is to create formatted text for the display from "readDataFile". For this code I set the format for the display by putting data text area, scroll button, font, background colour and else. Some of the difficulty when proceed this code are the format does not apply to the display. But luckily I got help from my senior that teach me how to set up properly.

For a reflection on this summary, many things that I've been experienced when finishing this project with my group. The first one I experience a great teamwork when working with them. The way we kept each other in touch and helping each other when runs into a problem. My team also keep updating what need to be changed or fix for the project. This helps us to prevent misinformation and wasting time to fix a coding. As a conclusion, this project give a lesson that to be more structured when creating a program. Also we need to be careful and do the project step by step to run the project smoothly.

(Akmal)

We are reaching the end of our project. It was all fun going through all the tasks that we were assigned by Miss Noornajwa. Although we have some fun doing this project, there is also a terrible part that we have to go through as a team to complete it. At first, we were asked to form a group of 3 to 4 to do the project. After about a week, I actually found a group that was suitable for me to do this project together. After the group was formed, I was assigned by my team members to be a group leader. At first, I thought that was a problem, but after reaching the end of our project, it was very fun to be a group leader, right? Yes it is.

While doing this project, we faced a lot of difficulties and challenges. One of the difficulties while doing the project is that I need to face the inner problems of the coding, such as compiling errors, logic errors, syntax errors, etc. There are always errors that pop up every time I execute the project. An example of a compilation error is that every time we want to compile our project, we always need to deal with the errors that have been made inside the code and need to fix them. After that, we also need to face logic, which is the hardest challenge we need to face. A logic error is when our program compiles and executes but does the wrong thing, returns an incorrect result, or returns no output when it should be returning an output.

Furthermore, while doing this project, I was assigned to do the most difficult task, like the tokenizer class, where I need to separate every customer's information using a delimiter or tokenizer. Well, it's not that difficult for intermediate programmers, but for me, this is a new thing, and I'm still adapting the coding well, following my understanding. The class was called `(private void loadCustomerDataFromFile(String filename) {)`, where this class was assigned to separate each line inside `customerList.txt`. What's more that makes this class harder is where it implements using queues and stacks, which is the latest topic that we are learning, and we need to apply it inside the coding. At first, when we were assigned to do the project, I didn't even understand what the question was asking. When I encounter this problem, I always ask Miss for some advice and opinions regarding the project. It was very challenging, but I can still adapt and handle this well. It's all thanks to Miss Noornajwa for assisting me with all the things I want, and it's been so helpful going through all those challenges and tasks.

(Irfan)

Working on this project involved developing multiple buttons, including the `nextQueueButton`, `addNewCustomerButton`, and `showDataButton`. Each button presented unique challenges and learning opportunities. The `addNewCustomerButton` required generating and storing new customer data efficiently, while the `showDataButton` necessitated retrieving and displaying data accurately. These tasks deepened my problem-solving skills and enriched my programming experience. Working with Afiq and Akmal on the team not only helped me solve problems with the buttons, but it also made me better at fixing problems as a group. Throughout the project, Madam Noor Najwa's advice and mentoring were very helpful, giving us important insights into how to deal with difficult problems

In conclusion, this project has been a valuable learning journey that has significantly enhanced my proficiency in Java programming, Object Oriented Programming principles, and practical data structure implementation. It encompassed the development of multiple buttons, including the `nextQueueButton`, `addNewCustomerButton`, and `showDataButton`, showcasing my ability to create functional and efficient solutions for real-world problems. The effective deployment of these buttons showed that I can handle an array of programming challenges, from handling data efficiently to making the user experience smooth. This project has not only helped me improve my technical skills, but it has also shown me how important it is to work in a helpful and collaborative setting if you want to get things done. Again, I want to thank Madam Najwa from the bottom of my heart for letting us present and for her incredible help in helping us understand the problems in the questions. We promise that we will take what we've learned and try not to do it again.

7.0 References

- Stack Overflow. (n.d.). Java GUI - How to append text to JTextArea from a static method? [online]
Available at: <https://stackoverflow.com/questions/31248081/java-gui-how-to-append-text-to-jtextarea-from-a-static-method> [Accessed 6 Nov. 2023].
- Chaudhary, N. 2022. Difference Between Stack and Queue Data Structures - Scaler Topics.
Available at: <https://www.scaler.com/topics/difference-between-stack-and-queue/>
- Difference between Stack and Queue Data Structures. 2023.
Available at: <https://www.geeksforgeeks.org/difference-between-stack-and-queue-data-structures/>
- javahungry.blogspot.com. (n.d.). Stack implementation : Using Swing for GUI Java program Source Code | Java Hungry. [online]
Available at: <https://javahungry.blogspot.com/2013/05/stack-implementation-using-swing-for-gui.html?m=1> [Accessed 6 Nov. 2023].
- Stack (Java Platform SE 8). 2023.
Available at: <https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html>
- Queue Interface In Java. 2023.
Available at: <https://www.geeksforgeeks.org/queue-interface-java/>
- Educative: Interactive Courses for Software Developers. (n.d.). Data Structures 101: how to use Stacks and Queues in Java. [online]
Available at: <https://www.educative.io/blog/data-structures-stack-queue-java-tutorial>
- Queue (Java Platform SE 8). 2023.
Available at: <https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html>.