**A purple and black background with text

Description automatically generated**

**CSC186 : OBJECT ORIENTED PROGRAMMING**

**MINI PROJECT**

**“RESTAURANT MANAGEMENT”**

|  |  |  |
| --- | --- | --- |
| **TEAM MEMBERS** | **SHAHMIR HAZIQ BIN SUHAIMI** | **2022893296** |
| **CHE KHAIRUL AZRI BIN CHE ARIZAN** | **2022465974** |
| **MUHAMMAD HAZEEQ HAIKAL BIN ROSLAN** | **2022676488** |
| **GROUP** | **RCDCS1103B** |  |
| **LECTURER NAME** | **MOHD NIZAM BIN OSMAN** |  |

Table Of Content

[**1.0  ORGANIZATIONAL STRUCTURE** 3](#_Toc155302326)

[**2.0 INTRODUCTION** 4](#_Toc155302327)

[**3.0 OBJECTIVES** 5](#_Toc155302328)

[**4.0 SCOPE** 6](#_Toc155302329)

[**6.0 INPUT FILE** 7](#_Toc155302330)

[**7.0 CLASS DEFINITION OF INHERITANCE, POLYMORPHISM AND RELATED CLASSES** 8](#_Toc155302331)

[**8.0 CLASS APPLICATION** 32](#_Toc155302332)

[**9.0 OUTPUT FILE OR/AND SAMPLE INTERFACES** 48](#_Toc155302333)

[**10.0 REFERENCES** 63](#_Toc155302334)

**1.0  ORGANIZATIONAL STRUCTURE**

**2.0 INTRODUCTION**

Atlantis Theme Park is a brand-new theme park that is working to create a new, user-

friendly system for their park. The system would also enable workers to save consumer data.

LinkedList and Stack will both be features of the new system. by means of LinkedList. The order

number for the customer will be kept in stack.

They will be taken out of line once they receive their order. There are two calculations

applied in this project which are, totalPrice and discount. The 10% discount will be given to

those whose total prices are over RM150. The system will display client testimonials and

payment receipts to show proof of purchase.

Atlantis Theme Park is a brand-new theme park that is working to create a new, user-

friendly system for their park. The system would also enable workers to save consumer data.

LinkedList and Stack will both be features of the new system. by means of LinkedList. The order

number for the customer will be kept in stack.

They will be taken out of line once they receive their order. There are two calculations

applied in this project which are, totalPrice and discount. The 10% discount will be given to

those whose total prices are over RM150. The system will display client testimonials and

payment receipts to show proof of purchase.

Food Inventory System is an innovative project aimed at revolutionizing the way food inventory is managed. The system is designed to be user-friendly, allowing both customers and staff to interact with it seamlessly. The system leverages the power of data structures like LinkedList and Stack.

The LinkedList is used to store and manipulate the information of food items, such as food name, quantity, price, expiry date, net weight, order ID, and user ID. This allows for efficient addition, removal, and retrieval of food items in the inventory.

The Stack is used to manage the order numbers for the customers. Once an order is placed, it is pushed onto the stack. When the order is ready and delivered to the customer, it is popped from the stack. This ensures that the orders are managed in a Last-In-First-Out (LIFO) manner, maintaining the integrity and efficiency of the order processing system.

Two key calculations are applied in this project which are totalPrice and discount. The totalPrice is calculated by summing up the prices of all the food items in an order. A discount of 10% is applied to orders where the totalPrice exceeds RM150, incentivizing customers to order more.

The system also features a robust file handling mechanism for reading and writing data from and to text files. This ensures that all customer data and order information are securely stored and easily retrievable.

Finally, the system provides features for displaying customer testimonials and payment receipts, offering proof of purchase and enhancing the transparency of the system. This Food Inventory System is not just a project, but a step towards making food inventory management more efficient and user-friendly.

**3.0 OBJECTIVES**

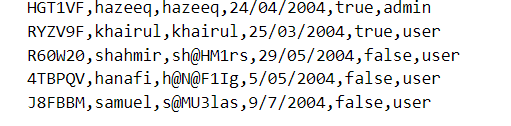
* To streamline the food inventory management process
* To facilitate customers in placing food orders
* To reduce labour and operational costs in managing the inventory
* To develop a system to efficiently handle food orders and inventory.

**4.0 SCOPE**

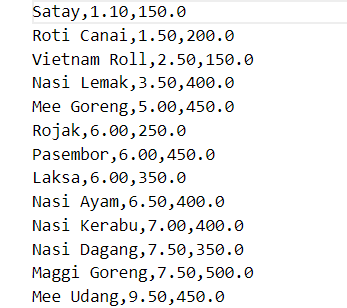
1. **Food Inventory Management**: The system will manage a comprehensive inventory of food items, including details such as food name, quantity, price, expiry date, net weight, order ID, and user ID.
2. **Order Processing**: The system will handle the entire process of order placement, tracking, and delivery. It will use a Stack data structure to manage the order numbers in a Last-In-First-Out (LIFO) manner.
3. **Discount Calculation**: The system will automatically calculate the total price of an order and apply a 10% discount if the total price exceeds RM150.
4. **File Handling**: The system will read and write data from and to text files, ensuring secure storage and easy retrieval of customer data and order information.
5. **User Interaction**: The system will display customer testimonials and payment receipts, enhancing transparency and user trust in the system.
6. **Cost Efficiency**: By automating various tasks related to food inventory management and order processing, the system aims to reduce labour and operational costs.

**6.0 INPUT FILE**

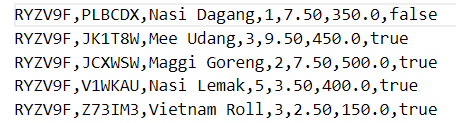
data.txt



foodMenu.txt



foodOrder.txt



**7.0 CLASS DEFINITION**

**Account.java**

import java.io.\*;

import java.util.Scanner;

public class Account extends FileHandling {

    String username;

    String password;

    String birthdate;

    boolean isMember;

    static FileHandling data = new FileHandling();

    // default constructor

    public Account() {

        super();

        this.username = "";

        this.password = "";

        this.birthdate = "";

        this.isMember = false;

    }

    // this is for registration

    public Account(String username, String password, String birthdate, boolean isMember) throws IOException {

        super("data.txt");

        this.username = username;

        this.password = password;

        this.birthdate = birthdate;

        this.isMember = isMember;

    }

    // this is for login

    public Account(String username, String password) throws IOException {

        super("data.txt");

        this.username = username;

        this.password = password;

        this.birthdate = "";

        this.isMember = false;

    }

    // getter and setter

    public String getUsername() {

        return this.username;

    }

    public void setUsername(String username) {

        this.username = username;

    }

    public String getPassword() {

        return this.password;

    }

    public void setPassword(String password) {

        this.password = password;

    }

    public String getBirthdate() {

        return this.birthdate;

    }

    public void setBirthdate(String birthdate) {

        this.birthdate = birthdate;

    }

    public boolean isMember() {

        return this.isMember;

    }

    public void setMember(boolean isMember) {

        this.isMember = isMember;

    }

    public boolean verifying() throws IOException {

        return verify(this.username, this.password);

    }

    public boolean checkingStrength() {

        return checkStrength(this.password);

    }

    public String generatingUserID() {

        return generateUserID();

    }

    public String getUserID() throws IOException {

        FileHandling data = new FileHandling("data.txt");

        String[] lines = data.read().split("\n");

        for (String line : lines) {

            // split the line into an array

            String[] arr = line.split(",");

            String user = arr[1];

            if (user.equals(this.username)) {

                return arr[0];

            }

        }

        return "";

    }

    // method to register a new account

    public void registers(String accountType) throws IOException {

        Scanner strInput = new Scanner(System.in);

        Scanner intInput = new Scanner(System.in);

        System.out.print("Please enter your username: ");

        String username = strInput.nextLine();

        System.out.print("Please enter your password: ");

        String password = strInput.nextLine();

        System.out.print("Confirm your password: ");

        String confirmPassword = strInput.nextLine();

        // check if the password matches

        while (!password.equals(confirmPassword) || !checkStrength(password)) {

            if (!checkStrength(password)) {

                System.out.println("Your password is not strong enough. Please try again.");

            } else {

                System.out.println("Your password does not match. Please try again.");

            }

            System.out.print("Please enter your password: ");

            password = strInput.nextLine();

            System.out.print("Confirm your password: ");

            confirmPassword = strInput.nextLine();

        }

        System.out.print("Please enter your birthdate (dd/mm/yyyy): ");

        String birthdate = strInput.nextLine();

        System.out.print("Are you a member? (Y/N): ");

        char member = intInput.next().charAt(0);

        boolean isMember = false;

        member = Character.toUpperCase(member);

        if (member == 'Y') {

            isMember = true;

        }

        // check if the username already exists

        data = new FileHandling("data.txt");

        String[] lines = data.read().split("\n");

        // check if null

        if (lines[0].equals("")) {

            System.out.println("You have successfully registered.");

        }

        for (String line : lines) {

            // split the line into an array

            String[] arr = line.split(",");

            String user = arr[1];

            // check if the username already exists

            if (user.equals(username)) {

                // close the input stream

                data.close();

                System.out.println("The username already exists. Please try again.");

                return;

            }

        }

        String userID = generateUserID();

        // write to file

        if (accountType.equals("admin")) {

            data.write(userID + "," + username + "," + password + "," + birthdate + "," + isMember + ",staff");

        } else if (accountType.equals("user")) {

            data.write(userID + "," + username + "," + password + "," + birthdate + "," + isMember + ",user");

        }

        data.close();

        System.out.println("You have successfully registered.");

    }

    // method to verify the username and password exists

    public static boolean verify(String username, String password) throws IOException {

        // read the file line by line

        data = new FileHandling("data.txt");

        String[] lines = data.read().split("\n");

        for (String line : lines) {

            // split the line into an array

            String[] arr = line.split(",");

            String user = arr[1];

            String pass = arr[2];

            // check if the username and password matches

            if (user.equals(username) && pass.equals(password)) {

                // close the input stream

                data.close();

                return true;

            }

        }

        return false;

    }

    // method to check strength of password

    public static boolean checkStrength(String password) {

        // check if the password is at least 8 characters long

        if (password.length() < 8) {

            return false;

        }

        // check if the password contains at least one uppercase letter

        boolean hasUppercase = !password.equals(password.toLowerCase());

        if (!hasUppercase) {

            return false;

        }

        // check if the password contains at least one lowercase letter

        boolean hasLowercase = !password.equals(password.toUpperCase());

        if (!hasLowercase) {

            return false;

        }

        // check if the password contains at least one number

        boolean hasNumber = password.matches(".\*\\d.\*");

        if (!hasNumber) {

            return false;

        }

        // check if the password contains at least one special character

        boolean hasSpecial = !password.matches("[A-Za-z0-9 ]\*");

        if (!hasSpecial) {

            return false;

        }

        return true;

    }

    // generate random string for userID

    public static String generateUserID() {

        String chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890";

        StringBuilder sb = new StringBuilder();

        for (int i = 0; i < 6; i++) {

            int index = (int) (Math.random() \* chars.length());

            sb.append(chars.charAt(index));

        }

        return sb.toString();

    }

    public void viewOrder(String userID, boolean finished) throws IOException {

        FileHandling foodOrder = new FileHandling("foodOrder.txt");

        String[] linesFoodOrder = foodOrder.readLines();

        // use circular linked list

        Circular foodOrderLL = new Circular();

        for (String line : linesFoodOrder) {

            String[] arr = line.split(",");

            String userIDFoodOrder = arr[0];

            String foodName = arr[2];

            int quantity = Integer.parseInt(arr[3]);

            double price = Double.parseDouble(arr[4]);

            double netWeight = Double.parseDouble(arr[5]);

            boolean isFinished = Boolean.parseBoolean(arr[6]);

            if (userIDFoodOrder.equals(userID) && isFinished == finished) {

                Food food = new Food(foodName, price, netWeight);

                food.setQuantity(quantity);

                foodOrderLL.add(food);

            }

        }

        // check if the list is empty

        if (foodOrderLL.isEmpty() && finished) {

            System.out.println("There is no finished order.");

            return;

        } else if (foodOrderLL.isEmpty() && !finished) {

            System.out.println("There is no unfinished order.");

            return;

        }

        for (int i = 0; i < 106; i++) {

            System.out.print("-");

        }

        System.out.println();

        System.out.printf("|%-20s|%-20s|%-20s|%-20s|%-20s|\n", "Food Name", "Quantity", "Price", "Net Weight",

                "Total Price");

        for (int i = 0; i < 106; i++) {

            System.out.print("-");

        }

        System.out.println();

        while (!foodOrderLL.isEmpty()) {

            Food food = (Food) foodOrderLL.removeFromFront();

            System.out.printf("|%-20s|%-20d|%-20.2f|%-20.2f|%-20.2f|\n", food.getFoodName(), food.getQuantity(),

                    food.getPrice(), food.getNetWeight(), food.getPrice() \* food.getQuantity());

        }

        for (int i = 0; i < 106; i++) {

            System.out.print("-");

        }

        System.out.println("\n");

    }

    // calculate total price of finished order

    public double calculateTotalPrice(String userID) throws IOException {

        FileHandling foodOrder = new FileHandling("foodOrder.txt");

        String[] linesFoodOrder = foodOrder.readLines();

        double totalPrice = 0;

        for (String line : linesFoodOrder) {

            String[] arr = line.split(",");

            String userIDFoodOrder = arr[0];

            String foodName = arr[2];

            int quantity = Integer.parseInt(arr[3]);

            double price = Double.parseDouble(arr[4]);

            double netWeight = Double.parseDouble(arr[5]);

            boolean isFinished = Boolean.parseBoolean(arr[6]);

            if (userIDFoodOrder.equals(userID) && isFinished) {

                Food food = new Food(foodName, price, netWeight);

                food.setQuantity(quantity);

                totalPrice += food.getPrice() \* food.getQuantity();

            }

        }

        return totalPrice;

    }

    // toString

    public String toString() {

        return "Username: " + username + "\nPassword: " + password + "\nBirthdate: " + birthdate + "\nIs Member: "

                + isMember + "\n";

    }

}

**Circular.java**

// circular linked list class

// the head and tail of the list are connected

// the tail points to the head

public class Circular {

    private Node head;

    private Node tail;

    private int size;

    public Circular() {

        head = null;

        tail = null;

        size = 0;

    }

    public void add(Object data) {

        Node newNode = new Node(data);

        if (head == null) {

            head = newNode;

            tail = newNode;

            tail.setNext(head);

        } else {

            tail.setNext(newNode);

            tail = newNode;

            tail.setNext(head);

        }

        size++;

    }

    public void add(Object data, int index) {

        if (index < 0 || index > size) {

            throw new IndexOutOfBoundsException();

        }

        Node newNode = new Node(data);

        // if the list is empty

        if (head == null) {

            head = newNode;

            tail = newNode;

            tail.setNext(head);

        } else if (index == 0) {

            newNode.setNext(head);

            head = newNode;

            tail.setNext(head);

        } else if (index == size) {

            tail.setNext(newNode);

            tail = newNode;

            tail.setNext(head);

        } else {

            Node current = head;

            for (int i = 0; i < index - 1; i++) {

                current = current.getNext();

            }

            newNode.setNext(current.getNext());

            current.setNext(newNode);

        }

        size++;

    }

    // insert a node at the end of the list

    public void insertAtBack(Object data) {

        add(data, size);

    }

    // insert a node at the front of the list

    public void insertAtFront(Object data) {

        add(data, 0);

    }

    // insert a node at middle of the list

    public void insertAtMiddle(Object data) {

        add(data, size / 2);

    }

    public Object remove(int index) {

        if (index < 0 || index >= size) {

            throw new IndexOutOfBoundsException();

        }

        Object removedData = null;

        if (index == 0) {

            removedData = head.getData();

            head = head.getNext();

            tail.setNext(head);

        } else {

            Node current = head;

            for (int i = 0; i < index - 1; i++) {

                current = current.getNext();

            }

            removedData = current.getNext().getData();

            current.setNext(current.getNext().getNext());

        }

        size--;

        return removedData;

    }

    // remove from front of the list

    public Object removeFromFront() {

        return remove(0);

    }

    // remove from back of the list

    public Object removeFromBack() {

        return remove(size - 1);

    }

    // remove from middle of the list

    public Object removeFromMiddle() {

        return remove(size / 2);

    }

    // remove all nodes from the list

    public void removeAll() {

        while (!isEmpty()) {

            removeFromFront();

        }

    }

    public Object get(int index) {

        if (index < 0 || index >= size) {

            throw new IndexOutOfBoundsException();

        }

        Node current = head;

        for (int i = 0; i < index; i++) {

            current = current.getNext();

        }

        return current.getData();

    }

    // get next node and start from start

    public Object getNext() {

        Node current = head;

        head = head.getNext();

        return current.getData();

    }

    public int size() {

        return size;

    }

    // getSize() is the same as size()

    public int getSize() {

        return size;

    }

    // check if the list is empty

    public boolean isEmpty() {

        return size == 0;

    }

    // swap the data of two nodes

    public void swap(int index1, int index2) {

        if (index1 < 0 || index1 >= size || index2 < 0 || index2 >= size) {

            throw new IndexOutOfBoundsException();

        }

        Node current1 = head;

        for (int i = 0; i < index1; i++) {

            current1 = current1.getNext();

        }

        Node current2 = head;

        for (int i = 0; i < index2; i++) {

            current2 = current2.getNext();

        }

        Object temp = current1.getData();

        current1.setData(current2.getData());

        current2.setData(temp);

    }

    public String toString() {

        String output = "";

        Node current = head;

        for (int i = 0; i < size; i++) {

            output += "[" + current.getData() + "] -> ";

            current = current.getNext();

        }

        return output;

    }

}

**FileHandling.java**

import java.io.\*;

public class FileHandling {

private BufferedReader reader;

private PrintWriter writer;

public FileHandling() {

}

public FileHandling(String fileName) throws IOException {

reader = new BufferedReader(new FileReader(fileName));

writer = new PrintWriter(new FileWriter(fileName, true));

}

public void setFile(String fileName) throws IOException {

reader = new BufferedReader(new FileReader(fileName));

writer = new PrintWriter(new FileWriter(fileName, true));

}

public void write(String text) {

writer.println(text);

}

public void close() throws IOException {

reader.close();

writer.close();

}

// read all lines

public String read() throws IOException {

String text = "";

String line = reader.readLine();

while (line != null) {

text += line + "\n";

line = reader.readLine();

}

return text;

}

public String[] readLines() throws IOException {

String[] lines = read().split("\n");

return lines;

}

public void emptyFiles() throws IOException {

String text = read();

if (text.equals("")) {

System.out.println("The file is empty.");

}

}

// clear the content of the file

public void clear(String fileName) throws IOException {

writer = new PrintWriter(new FileWriter(fileName));

writer.print("");

}

}

**Food.java**

import java.util.\*;

import java.io.\*;

public class Food {

    private String foodName;

    private int quantity;

    private double price;

    private Date expiryDate;

    private double netWeight;

    private String orderID;

    private String userID;

    private boolean isFinished;

    public Food() {

        this.foodName = "";

        this.quantity = 0;

        this.price = 0.0;

        this.expiryDate = new Date();

        this.netWeight = 0.0;

        this.orderID = "";

        this.userID = "";

    }

    public Food(String foodName, int quantity, double price, Date expiryDate, double netWeight) {

        this.foodName = foodName;

        this.quantity = quantity;

        this.price = price;

        this.expiryDate = expiryDate;

        this.netWeight = netWeight;

    }

    // for adding food to foodMenu.txt

    public Food(String foodName, double price, double netWeight) {

        this.foodName = foodName;

        this.price = price;

        this.netWeight = netWeight;

    }

    // for adding food to foodOrder.txt

    public Food(String userID, String orderID, String foodName, int quantity, double price, double netWeight,

            boolean isFinished) {

        this.userID = userID;

        this.orderID = orderID;

        this.foodName = foodName;

        this.quantity = quantity;

        this.price = price;

        this.netWeight = netWeight;

        this.isFinished = isFinished;

    }

    // copy constructor

    public Food(Food food) {

        this.foodName = food.foodName;

        this.quantity = food.quantity;

        this.price = food.price;

        this.expiryDate = food.expiryDate;

        this.netWeight = food.netWeight;

        this.orderID = food.orderID;

        this.userID = food.userID;

        this.isFinished = food.isFinished;

    }

    // setter

    public void setFoodName(String foodName) {

        this.foodName = foodName;

    }

    public void setQuantity(int quantity) {

        this.quantity = quantity;

    }

    public void setPrice(double price) {

        this.price = price;

    }

    public void setExpiryDate(Date expiryDate) {

        this.expiryDate = expiryDate;

    }

    public void setNetWeight(double netWeight) {

        this.netWeight = netWeight;

    }

    public void setOrderID(String orderID) {

        this.orderID = orderID;

    }

    public void setUserID(String userID) {

        this.userID = userID;

    }

    public void setIsFinished(boolean isFinished) {

        this.isFinished = isFinished;

    }

    // getter

    public String getFoodName() {

        return foodName;

    }

    public int getQuantity() {

        return quantity;

    }

    public double getPrice() {

        return price;

    }

    public Date getExpiryDate() {

        return expiryDate;

    }

    public double getNetWeight() {

        return netWeight;

    }

    public String getOrderID() {

        return orderID;

    }

    public String getUserID() {

        return userID;

    }

    public boolean getIsFinished() {

        return isFinished;

    }

    // calculate the total weight

    public double calculateTotalWeight() {

        return netWeight \* quantity;

    }

    // calculate price after SST

    public double afterSST() {

        return price \* 1.06;

    }

    // calculate the total price

    public double calculateTotalPrice() {

        return afterSST() \* quantity;

    }

    // member or non-member

    public boolean isMember() throws IOException {

        // read from file

        FileHandling data = new FileHandling("data.txt");

        String linesData = data.read();

        String[] dataPerLine = linesData.split("\n");

        if (dataPerLine.length == 0) {

            System.out.println("There is no data.");

            return false;

        }

        for (int i = 0; i < dataPerLine.length; i++) {

            String[] dataDetails = dataPerLine[i].split(",");

            boolean isMember = Boolean.parseBoolean(dataDetails[2]);

            if (isMember) {

                return true;

            }

        }

        return false;

    }

    // check if today is their birthday

    public boolean isBirthday() throws IOException {

        FileHandling data = new FileHandling("data.txt");

        String linesData = data.read();

        String[] dataPerLine = linesData.split("\n");

        if (dataPerLine.length == 0) {

            System.out.println("There is no data.");

            return false;

        }

        for (int i = 0; i < dataPerLine.length; i++) {

            String[] dataDetails = dataPerLine[i].split(",");

            String birthday = dataDetails[3];

            // get current date

            Date currentDate = new Date();

            if (birthday.equals(String.format("%td/%tm", currentDate, currentDate))) {

                return true;

            }

        }

        return false;

    }

    // calculate the total price after discount

    // if member and birthday then the discount will be stacked

    public double discountedPrice() throws IOException {

        if (isMember()) {

            return calculateTotalPrice() \* 0.9;

        }

        if (isBirthday()) {

            return calculateTotalPrice() \* 0.8;

        }

        return calculateTotalPrice();

    }

    // generate random foodID with a length of 6 combination string and letter

    public String generateFoodID() {

        String foodID = "";

        String characters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

        int length = 6;

        for (int i = 0; i < length; i++) {

            foodID += characters.charAt((int) (Math.random() \* characters.length()));

        }

        return foodID;

    }

    // determine the food

    public void determineFood(int foodChoice, int quantity) throws IOException {

        // read the price and net weight from the text file

        FileHandling foodMenu = new FileHandling("foodMenu.txt");

        String linesFoodMenu = foodMenu.read();

        String[] foodMenuPerLine = linesFoodMenu.split("\n");

        String[] foodDetails = foodMenuPerLine[foodChoice - 1].split(",");

        double price = Double.parseDouble(foodDetails[1]);

        double netWeight = Double.parseDouble(foodDetails[2]);

        // set the quantity

        setQuantity(quantity);

        // set the food name

        setFoodName(foodDetails[0]);

        // set the price and net weight

        setPrice(price);

        setNetWeight(netWeight);

        setOrderID(generateFoodID());

        // return the food name

    }

    // display food menu from foodMenu.txt

    public int displayFoodMenu() throws IOException {

        FileHandling foodMenu = new FileHandling("foodMenu.txt");

        String linesFoodMenu = foodMenu.read();

        if (linesFoodMenu.equals("")) {

            System.out.println("There is no food in the menu.");

            return 0;

        }

        String[] foodMenuPerLine = linesFoodMenu.split("\n");

        int i = 0;

        // format into table

        for (int j = 0; j < 70; j++) {

            System.out.print("-");

        }

        System.out.println();

        System.out.printf("|%-5s|%-20s|%-20s|%-20s|\n", "No.", "Food Name", "Price (RM)", "Net Weight (gram)");

        for (int j = 0; j < 70; j++) {

            System.out.print("-");

        }

        System.out.println();

        for (i = 0; i < foodMenuPerLine.length; i++) {

            String[] foodDetails = foodMenuPerLine[i].split(",");

            String foodName = foodDetails[0];

            double price = Double.parseDouble(foodDetails[1]);

            double netWeight = Double.parseDouble(foodDetails[2]);

            System.out.printf("|%-5d|%-20s|%-20s|%-20s|\n", (i + 1), foodName, String.format("%,.2f", price),

                    String.format("%,.2f", netWeight));

        }

        for (int j = 0; j < 70; j++) {

            System.out.print("-"); // print 80 dashes

        }

        System.out.println();

        return i;

    }

    // method to count the number of menu in the food menu

    public int countMenu() throws IOException {

        FileHandling foodMenu = new FileHandling("foodMenu.txt");

        String linesFoodMenu = foodMenu.read();

        if (linesFoodMenu.equals("")) {

            return 0;

        }

        String[] foodMenuPerLine = linesFoodMenu.split("\n");

        return foodMenuPerLine.length;

    }

    // printer

    public String toString() {

        // format the date to dd/mm/yyyy

        return "Food Name: " + foodName + "\nQuantity: "

                + quantity + "\nPrice: RM " + String.format("%,.2f", price)

                + "\nNet Weight: " + String.format("%,.2f", netWeight) + " gram\n";

    }

}

**Node.java**

public class Node {

private Object data;

private Node next;

// default constructor

public Node() {

data = null;

next = null;

}

// normal constructor

public Node(Object data) {

this.data = data;

next = null;

}

// getter

public Object getData() {

return data;

}

public Node getNext() {

return next;

}

// setter

public void setData(Object data) {

this.data = data;

}

public void setNext(Node next) {

this.next = next;

}

public String toString() {

return data + "";

}

}

**QueueCustom.java**

public class QueueCustom extends Circular {

public QueueCustom() {

super();

}

// i. Add data at the start of the list (enqueue).

public void enqueue(Object data) {

insertAtFront(data);

}

// ii. Removes data at the end of a list (dequeue) and return the removed data.

public Object dequeue() {

return removeFromBack();

}

// iii. Determine whether the list is empty.

public boolean isEmpty() {

return super.isEmpty();

}

// iv. Determine the size of the list.

public int getSize() {

return super.size();

}

}

**Staff.java**

import java.io.\*;

import java.util.Scanner;

public class Staff extends FileHandling {

    private String username;

    private String password;

    // default constructor

    public Staff() {

        this.username = "";

        this.password = "";

    }

    // normal constructor

    public Staff(String username, String password) {

        this.username = username;

        this.password = password;

    }

    // getter

    public String getUsername() {

        return this.username;

    }

    public String getPassword() {

        return this.password;

    }

    // setter

    public void setUsername(String username) {

        this.username = username;

    }

    public void setPassword(String password) {

        this.password = password;

    }

    // set food price

    public void setFoodPrice(Food food, double price) {

        food.setPrice(price);

    }

    // set food quantity

    public void setFoodQuantity(Food food, int quantity) {

        food.setQuantity(quantity);

    }

    // put food into linked list custom

    // change the last food to be finished

    // write back to foodOrder.txt

    public void updateFood() throws IOException {

        FileHandling foodOrder = new FileHandling("foodOrder.txt");

        String linesFoodOrder = foodOrder.read();

        if (linesFoodOrder.equals("")) {

            System.out.println("There is no food order.");

            return;

        }

        String[] foodOrderPerLine = linesFoodOrder.split("\n");

        Circular foodOrderLL = new Circular();

        // check if all food is finished

        boolean isAllFinished = true;

        for (int i = 0; i < foodOrderPerLine.length; i++) {

            String[] foodOrderDetails = foodOrderPerLine[i].split(",");

            String userID = foodOrderDetails[0];

            String orderID = foodOrderDetails[1];

            String foodName = foodOrderDetails[2];

            int quantity = Integer.parseInt(foodOrderDetails[3]);

            double price = Double.parseDouble(foodOrderDetails[4]);

            double netWeight = Double.parseDouble(foodOrderDetails[5]);

            boolean isFinished = Boolean.parseBoolean(foodOrderDetails[6]);

            if (!isFinished) {

                isAllFinished = false;

            }

            Food food = new Food(userID, orderID, foodName, quantity, price, netWeight, isFinished);

            foodOrderLL.insertAtBack(food);

        }

        if (isAllFinished) {

            System.out.println("All food is finished.");

            return;

        }

        // change the last food to be finished that is not finished

        // traverse from the back

        for (int i = foodOrderLL.getSize() - 1; i >= 0; i--) {

            Food food = (Food) foodOrderLL.get(i);

            if (!food.getIsFinished()) {

                food.setIsFinished(true);

                break;

            }

        }

        // write back to foodOrder.txt

        foodOrder.setFile("foodOrder.txt");

        // clear the file

        foodOrder.clear("foodOrder.txt");

        // sort the food by finished and unfinished

        // the order of the unfinished must stay the same

        // push down the finished food to the bottom

        for (int i = 0; i < foodOrderLL.getSize(); i++) {

            for (int j = i + 1; j < foodOrderLL.getSize(); j++) {

                Food food1 = (Food) foodOrderLL.get(i);

                Food food2 = (Food) foodOrderLL.get(j);

                if (food1.getIsFinished() && !food2.getIsFinished()) {

                    foodOrderLL.swap(i, j);

                }

            }

        }

        while (!foodOrderLL.isEmpty()) {

            Food food2 = (Food) foodOrderLL.removeFromFront();

            foodOrder.write(food2.getUserID() + "," + food2.getOrderID() + "," + food2.getFoodName() + "," +

                    food2.getQuantity() + "," + String.format("%,.2f", food2.getPrice()) + "," +

                    food2.getNetWeight() + "," + food2.getIsFinished());

        }

        foodOrder.close();

    }

    // view finished order in finishedOrder.txt

    public void viewFinishedOrder() throws Exception {

        FileHandling finishedOrder = new FileHandling("foodOrder.txt");

        String linesFinishedOrder = finishedOrder.read();

        // use circular linked list to store finished order

        Circular finishedOrderCLL = new Circular();

        String[] finishedOrderPerLine = linesFinishedOrder.split("\n");

        for (int i = 0; i < finishedOrderPerLine.length; i++) {

            String[] finishedOrderDetails = finishedOrderPerLine[i].split(",");

            String userID = finishedOrderDetails[0];

            String orderID = finishedOrderDetails[1];

            String foodName = finishedOrderDetails[2];

            int quantity = Integer.parseInt(finishedOrderDetails[3]);

            double price = Double.parseDouble(finishedOrderDetails[4]);

            double netWeight = Double.parseDouble(finishedOrderDetails[5]);

            boolean isFinished = Boolean.parseBoolean(finishedOrderDetails[6]);

            if (isFinished) {

                Food food = new Food(userID, orderID, foodName, quantity, price, netWeight, isFinished);

                finishedOrderCLL.insertAtBack(food);

            }

        }

        // if empty

        if (finishedOrderCLL.isEmpty()) {

            System.out.println("There is no finished order.");

            return;

        }

        // print out the finished order

        while (!finishedOrderCLL.isEmpty()) {

            Food food = (Food) finishedOrderCLL.removeFromFront();

            System.out.println(food);

        }

    }

    // add new food

    public void addFood(Food food) throws Exception {

        FileHandling foodMenu = new FileHandling("foodMenu.txt");

        // String linesFoodMenu = foodMenu.read();

        String foodName = food.getFoodName();

        double price = food.getPrice();

        double netWeight = food.getNetWeight();

        foodMenu.write(foodName + "," + String.format("%,.2f", price) + "," + netWeight);

        foodMenu.close();

    }

    // remove food

    // 1 - remove food from front

    // 2 - remove food at the end

    // 3 - remove food from middle

    // 4 - remove all food

    public void removeFood(int choice) throws Exception {

        FileHandling foodMenu = new FileHandling("foodMenu.txt");

        String linesFoodMenu = foodMenu.read();

        Circular foodMenuLL = new Circular();

        if (linesFoodMenu.equals("")) {

            System.out.println("There is no food in the menu.");

            return;

        }

        String[] foodMenuPerLine = linesFoodMenu.split("\n");

        for (int i = 0; i < foodMenuPerLine.length; i++) {

            String[] foodDetails = foodMenuPerLine[i].split(",");

            String foodName = foodDetails[0];

            double price = Double.parseDouble(foodDetails[1]);

            double netWeight = Double.parseDouble(foodDetails[2]);

            Food food = new Food(foodName, price, netWeight);

            foodMenuLL.insertAtBack(food);

        }

        Scanner intInput = new Scanner(System.in);

        // System.out.println("Removed food:\n");

        Food removedObject = null;

        switch (choice) {

            case 1:

                removedObject = (Food) foodMenuLL.removeFromFront();

                break;

            case 2:

                removedObject = (Food) foodMenuLL.removeFromBack();

                break;

            case 3:

                int sizeBeforeRemove = foodMenuLL.getSize();

                removedObject = (Food) foodMenuLL.removeFromMiddle();

                System.out.println("Removed food at index " + (foodMenuLL.getSize() / 2) + " out of " +

                        sizeBeforeRemove + " food.\n");

                break;

            case 4:

                foodMenuLL.removeAll();

                System.out.println("All food is removed.");

                break;

            case 5:

                System.out.print("Enter the index (1 - " + foodMenuLL.getSize() + "): ");

                choice = intInput.nextInt();

                foodMenuLL.remove(choice - 1);

                break;

            default:

                System.out.println("Invalid choice.");

                break;

        }

        if (removedObject != null) {

            System.out.println(removedObject);

            // tell how many food left

            System.out.println("There are " + foodMenuLL.getSize() + " food left.\n");

        }

        // write back to foodMenu.txt

        foodMenu.setFile("foodMenu.txt");

        // clear the file

        foodMenu.clear("foodMenu.txt");

        while (!foodMenuLL.isEmpty()) {

            Food food = (Food) foodMenuLL.removeFromFront();

            foodMenu.write(

                    food.getFoodName() + "," + String.format("%,.2f", food.getPrice()) + "," +

                            food.getNetWeight());

        }

        foodMenu.close();

    }

    // sort food by price

    public void sortFoodByPrice() throws Exception {

        FileHandling foodMenu = new FileHandling("foodMenu.txt");

        String linesFoodMenu = foodMenu.read();

        Circular foodMenuLL = new Circular();

        if (linesFoodMenu.equals("")) {

            System.out.println("There is no food in the menu.");

            return;

        }

        String[] foodMenuPerLine = linesFoodMenu.split("\n");

        for (int i = 0; i < foodMenuPerLine.length; i++) {

            String[] foodDetails = foodMenuPerLine[i].split(",");

            String foodName = foodDetails[0];

            double price = Double.parseDouble(foodDetails[1]);

            double netWeight = Double.parseDouble(foodDetails[2]);

            Food food = new Food(foodName, price, netWeight);

            foodMenuLL.insertAtBack(food);

        }

        // sort by price using bubble sort from LinkedListCustom and sort lowest to

        // highest

        for (int i = 0; i < foodMenuLL.getSize(); i++) {

            for (int j = i + 1; j < foodMenuLL.getSize(); j++) {

                Food food1 = (Food) foodMenuLL.get(i);

                Food food2 = (Food) foodMenuLL.get(j);

                if (food1.getPrice() > food2.getPrice()) {

                    foodMenuLL.swap(i, j);

                }

            }

        }

        // write back to foodMenu.txt

        foodMenu.setFile("foodMenu.txt");

        // clear the file

        foodMenu.clear("foodMenu.txt");

        while (!foodMenuLL.isEmpty()) {

            Food food = (Food) foodMenuLL.removeFromFront();

            foodMenu.write(

                    food.getFoodName() + "," + String.format("%,.2f", food.getPrice()) + "," +

                            food.getNetWeight());

        }

        foodMenu.close();

    }

    // view unfinished order in foodOrder.txt by one by one by using circular linked

    // get next

    public void viewOrder() throws Exception {

        Scanner input = new Scanner(System.in);

        // use circular linked list to store unfinished order and then get next

        Circular orderCLL = new Circular();

        FileHandling foodOrder = new FileHandling("foodOrder.txt");

        String linesFoodOrder = foodOrder.read();

        if (linesFoodOrder.equals("")) {

            System.out.println("There is no order.");

            return;

        }

        String[] foodOrderPerLine = linesFoodOrder.split("\n");

        for (int j = 0; j < foodOrderPerLine.length; j++) {

            String[] foodOrderDetails = foodOrderPerLine[j].split(",");

            String userID = foodOrderDetails[0];

            String orderID = foodOrderDetails[1];

            String foodName = foodOrderDetails[2];

            int quantity = Integer.parseInt(foodOrderDetails[3]);

            double price = Double.parseDouble(foodOrderDetails[4]);

            double netWeight = Double.parseDouble(foodOrderDetails[5]);

            boolean isFinished = Boolean.parseBoolean(foodOrderDetails[6]);

            Food food = new Food(userID, orderID, foodName, quantity, price, netWeight, isFinished);

            orderCLL.insertAtBack(food);

        }

        System.out.println("Press enter to view next order. Press 0 to exit.\n");

        while (true) {

            Food food = (Food) orderCLL.getNext();

            System.out.println(food);

            String choice = input.nextLine();

            if (choice.equals("0")) {

                break;

            }

        }

        foodOrder.close();

    }

    // toString

    public String toString() {

        return "Username: " + username + "\nPassword: " + password;

    }

}

**8.0 CLASS APPLICATION**

import java.util.\*;

import java.io.\*;

import java.text.SimpleDateFormat;

// linkedlist must have removal, searching, updating and traversal

// optional: sorting, insertion, merging, reversing

public class Main {

    public static final int ms = 1;

    public static final int sec = ms \* 1000;

    public static final int min = sec \* 60;

    public static final int hour = min \* 60;

    public static void main(String[] args) throws Exception {

        // 2 Scanners for String and Integer

        Scanner strInput = new Scanner(System.in);

        Scanner intInput = new Scanner(System.in);

        // LinkedList to store Food objects

        LinkedList<Food> foodList = new LinkedList<Food>();

        // LinkedListCustom foodListCustom = new LinkedListCustom();

        QueueCustom foodQueueCustom = new QueueCustom();

        Food food = new Food();

        // welcome message

        System.out.println("Welcome to the Food Inventory System");

        System.out.println("====================================");

        int choice = 0;

        // login or register

        System.out.print("1. Login\n2. Register\n3. Exit\n\nEnter your choice: ");

        choice = intInput.nextInt();

        System.out.println();

        // login

        if (choice == 1) {

            System.out.print("Enter your username: ");

            String username = strInput.nextLine();

            System.out.print("Enter your password: ");

            String password = strInput.nextLine();

            Account login = new Account(username, password);

            // validate the username and password from the text file

            if (!login.verifying()) {

                System.out.println("\nLogin failed.");

                return;

            }

            System.out.println("\nLogin successful.\n");

            // check if the user is admin or not

            // get the user type from the text file

            String userType = "";

            String name = "";

            FileHandling data = new FileHandling("data.txt");

            String lines = data.read();

            String[] dataPerLine = lines.split("\n");

            for (int i = 0; i < dataPerLine.length; i++) {

                String[] dataPerComma = dataPerLine[i].split(",");

                name = dataPerComma[1];

                String passwordData = dataPerComma[2];

                userType = dataPerComma[5];

                if (username.equals(name) && password.equals(passwordData)) {

                    break;

                }

            }

            System.out.println("Welcome, " + name + ".\n");

            while (true) {

                if (userType.equalsIgnoreCase("admin")) {

                    Staff staff = new Staff();

                    System.out.print("================================================\n");

                    System.out.printf("|%-5s|%-40s|\n", "No.", "Menu");

                    System.out.print("================================================\n");

                    System.out.printf("|%-5s|%-40s|\n", "1.", "Update Food Order List");

                    System.out.printf("|%-5s|%-40s|\n", "2.", "View Finished Food Order List");

                    System.out.printf("|%-5s|%-40s|\n", "3.", "View All Order One By One");

                    System.out.printf("|%-5s|%-40s|\n", "4.", "Add Food");

                    System.out.printf("|%-5s|%-40s|\n", "5.", "View Food Menu");

                    System.out.printf("|%-5s|%-40s|\n", "6.", "Delete Food Menu");

                    System.out.printf("|%-5s|%-40s|\n", "7.", "Sort Food Menu By Price");

                    System.out.printf("|%-5s|%-40s|\n", "8.", "Register New Admin");

                    System.out.printf("|%-5s|%-40s|\n", "9.", "Exit");

                    System.out.print("================================================\n");

                    System.out.print("\nEnter your choice: ");

                    int adminChoice = intInput.nextInt();

                    System.out.println();

                    // update food order list

                    if (adminChoice == 1) {

                        staff.updateFood();

                    }

                    else if (adminChoice == 2){

                        staff.viewFinishedOrder();

                    }

                    else if (adminChoice == 3) {

                        staff.viewOrder();

                    }

                    else if (adminChoice == 4) {

                        System.out.print("Enter the food name: ");

                        String foodName = strInput.nextLine();

                        System.out.print("Enter the price (RM): ");

                        double price = intInput.nextDouble();

                        System.out.print("Enter the net weight (gram): ");

                        double netWeight = intInput.nextDouble();

                        food = new Food(foodName, price, netWeight);

                        staff.addFood(food);

                        System.out.println("Food added successfully.\n");

                    }

                    else if (adminChoice == 5) {

                        food.displayFoodMenu();

                    }

                    else if (adminChoice == 6) {

                        System.out.print(

                                "1. Delete from front\n2. Delete from back\n3. Delete from middle\n4. Delete all\n5. Delete by index\n\nEnter your choice: ");

                        int deleteChoice = intInput.nextInt();

                        System.out.println();

                        if (deleteChoice >= 1 && deleteChoice <= 5)

                            staff.removeFood(deleteChoice);

                    }

                    else if (adminChoice == 7) {

                        staff.sortFoodByPrice();

                        System.out.println("Food menu sorted successfully.\n");

                    }

                    else if (adminChoice == 8) {

                        Account register = new Account();

                        register.registers("admin");

                    }

                    else if (adminChoice == 9) {

                        break;

                    } else {

                        System.out.println("Invalid input. Please try again.");

                    }

                } else {

                    System.out.print("================================================\n");

                    System.out.printf("|%-5s|%-40s|\n", "No.", "Menu");

                    System.out.print("================================================\n");

                    System.out.printf("|%-5s|%-40s|\n", "1.", "Order Food");

                    System.out.printf("|%-5s|%-40s|\n", "2.", "View Finished Order");

                    System.out.printf("|%-5s|%-40s|\n", "3.", "View Unfinished Order");

                    System.out.printf("|%-5s|%-40s|\n", "4.", "View Total Price");

                    System.out.printf("|%-5s|%-40s|\n", "5.", "Exit");

                    System.out.print("================================================\n");

                    System.out.print("\nEnter your choice: ");

                    int userChoice= intInput.nextInt();

                    System.out.println();

                    Account account = new Account();

                    if (userChoice == 1) {

                        while (true) {

                            int length = food.countMenu();

                            food.displayFoodMenu();

                            System.out.print("\n\nEnter the food you want to order (1 - " + length + "): ");

                            int foodChoice = intInput.nextInt();

                            if (foodChoice < 1 || foodChoice > length) {

                                System.out.println("\nThere is no food with that number. Please try again.\n");

                                continue;

                            }

                            System.out.print("Enter the quantity: ");

                            int quantity = intInput.nextInt();

                            Food orderedFood = new Food();

                            orderedFood.determineFood(foodChoice, quantity);

                            // store the food object into the queue

                            foodQueueCustom.enqueue(orderedFood);

                            System.out.print("\n1. Order more food\n2. Proceed to checkout\n\nEnter your choice: ");

                            int orderChoice = intInput.nextInt();

                            if (orderChoice == 1) {

                                continue;

                            } else if (orderChoice == 2) {

                                break;

                            } else {

                                System.out.println("Invalid input. Please try again.");

                            }

                        }

                        // take from food queue and store into foodOrder.txt

                        FileHandling foodOrder = new FileHandling("foodOrder.txt");

                        String linesFoodOrder = foodOrder.read();

                        foodOrder.clear("foodOrder.txt");

                        String newOrder = "";

                        // use custom queue

                        while (!foodQueueCustom.isEmpty()) {

                            food = (Food) foodQueueCustom.dequeue();

                            String foodName = food.getFoodName();

                            int quantity = food.getQuantity();

                            double price = food.getPrice();

                            double netWeight = food.getNetWeight();

                            // generate a string of random numbers and letters for the order ID

                            String orderID = food.generateFoodID();

                            String userID = login.getUserID();

                            newOrder += userID + "," + orderID + "," + foodName + "," + quantity + ","

                                    + String.format("%,.2f", price) + "," + netWeight + "," + false + "\n";

                        }

                        newOrder += linesFoodOrder;

                        // trim the last \n to prevent new spaces being created every time new order is

                        // added

                        newOrder = newOrder.substring(0, newOrder.length() - 1);

                        foodOrder.write(newOrder);

                        // close the file

                        foodOrder.close();

                    } else if (userChoice == 2) {

                        account.viewOrder(login.getUserID(), true);

                    } else if (userChoice == 3) {

                        account.viewOrder(login.getUserID(), false);

                    } else if (userChoice == 4) {

                        double totalPrice = account.calculateTotalPrice(login.getUserID());

                        System.out.printf("Total price of finished order: RM %,.2f\n\n", totalPrice);

                    } else if (userChoice == 5) {

                        break;

                    } else {

                        System.out.println("Invalid input. Please try again.");

                    }

                }

                System.out.println("Press enter to continue...");

                strInput.nextLine();

            } // end of while loop

            System.out.println("Thank you for using the Food Inventory System.");

        }

        // register a new account

        else if (choice == 2) {

            Account register = new Account();

            register.registers("user");

        } else if (choice == 3) {

            System.out.println("Thank you for using the Food Inventory System.");

        } else {

            System.out.println("Invalid input. Please try again.");

        }

        // close the scanner

        strInput.close();

        intInput.close();

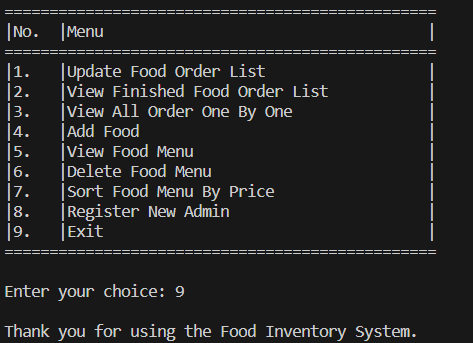
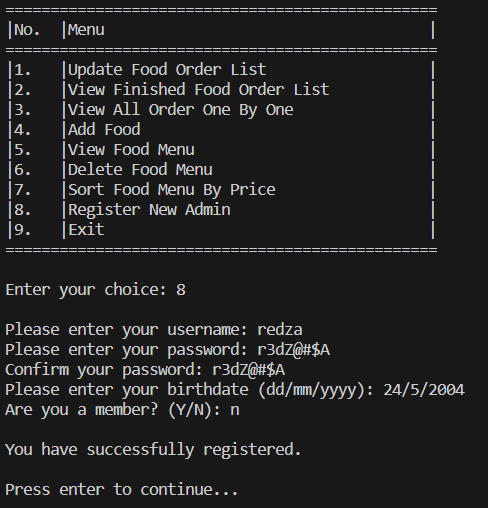
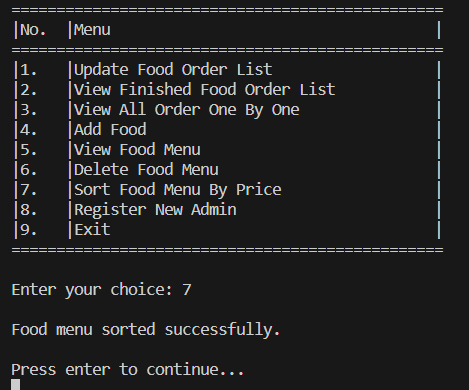
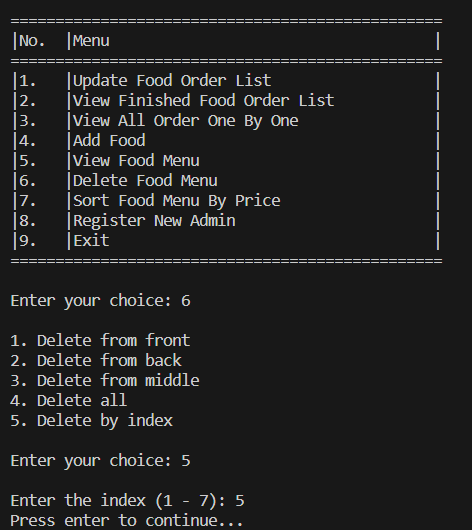
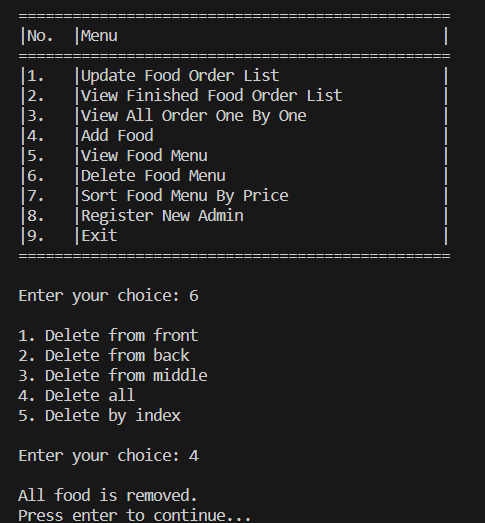
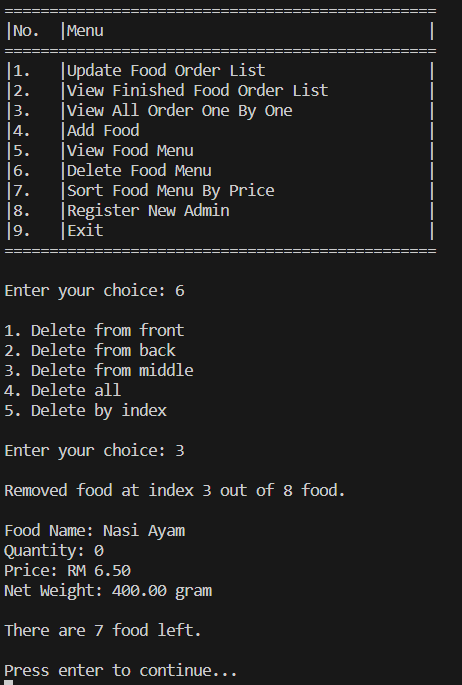
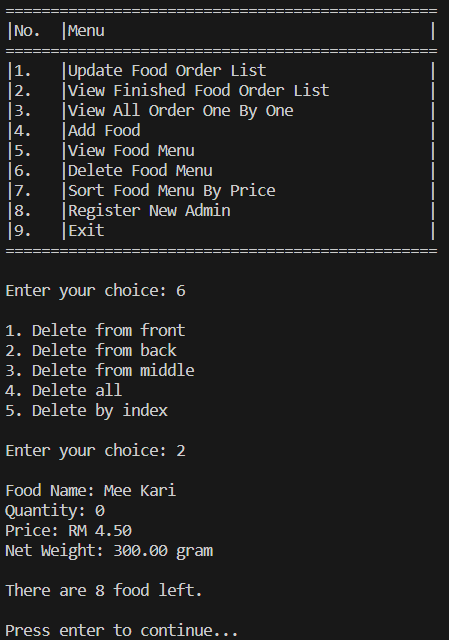
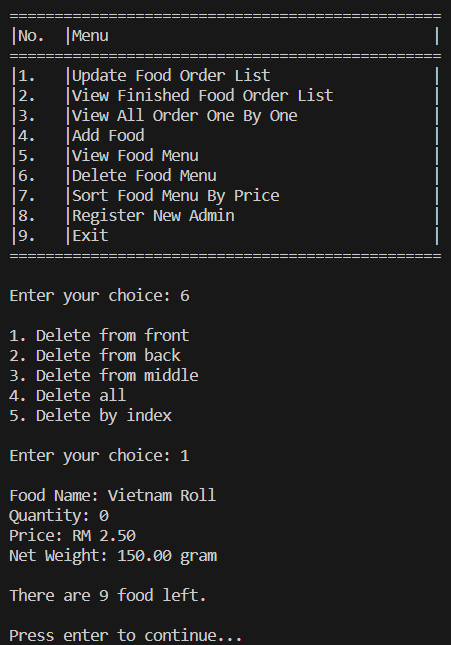
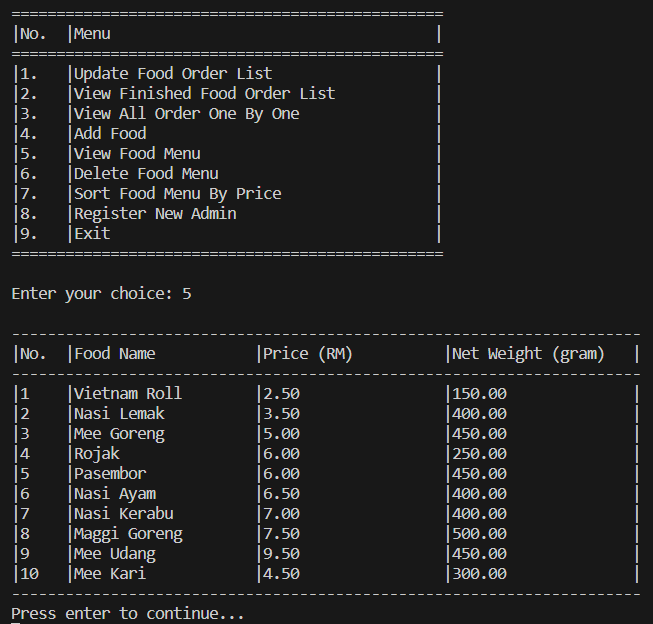
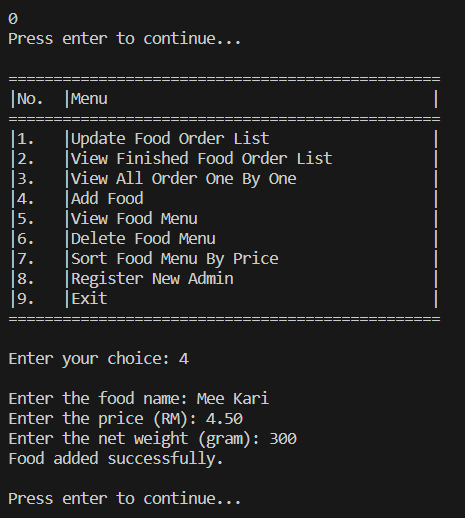
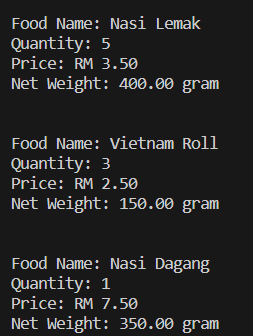
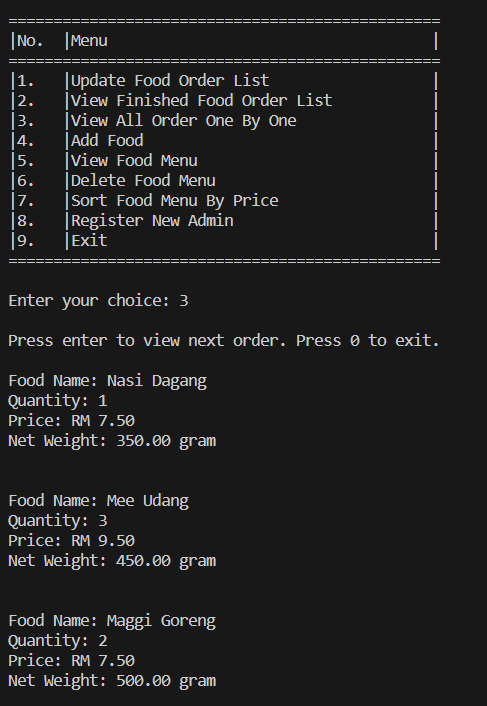
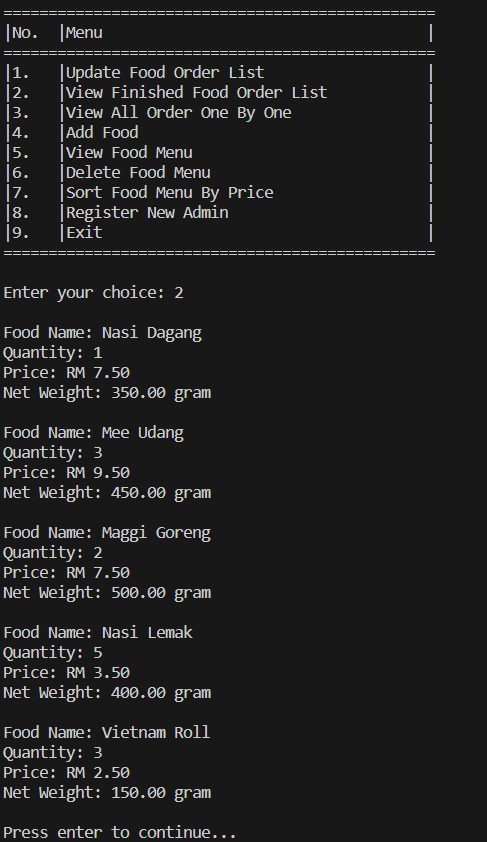
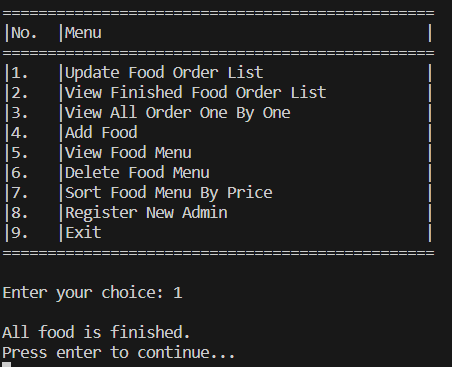
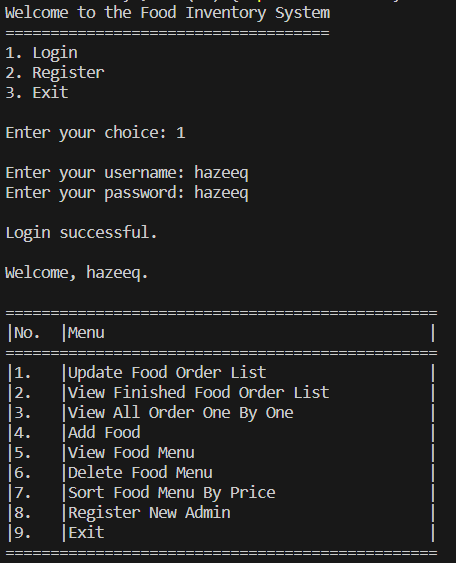
    }

}

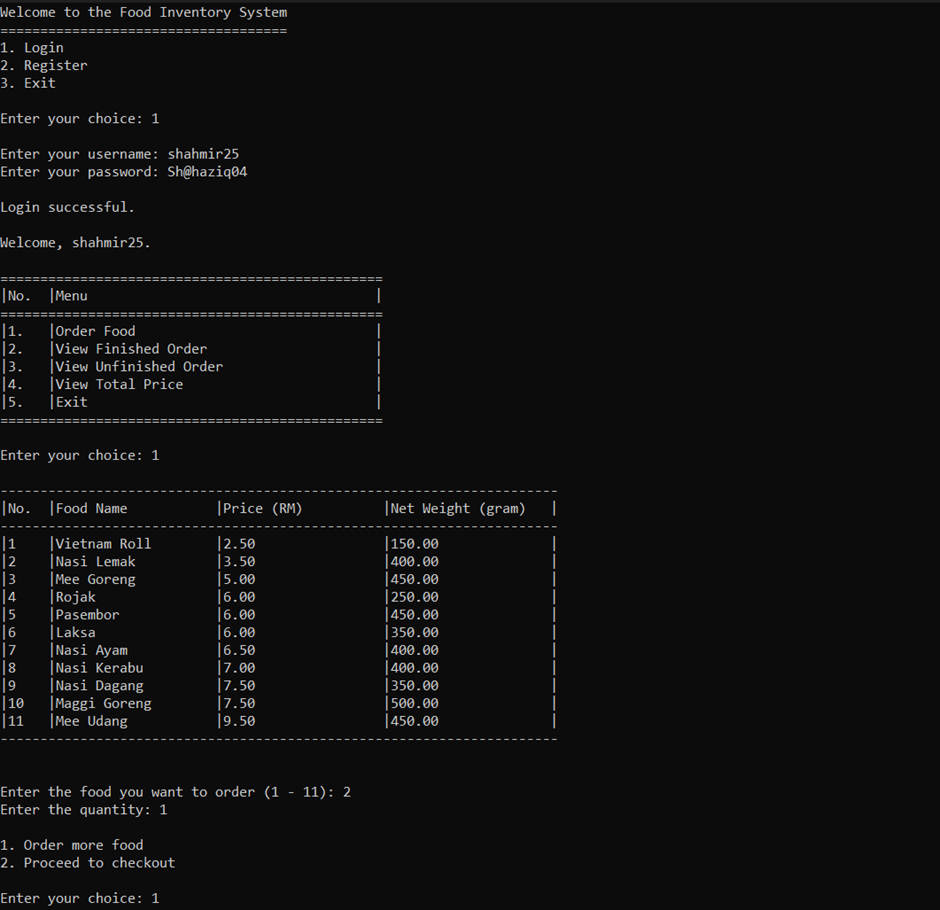
**9.0 SAMPLE INTERFACES**

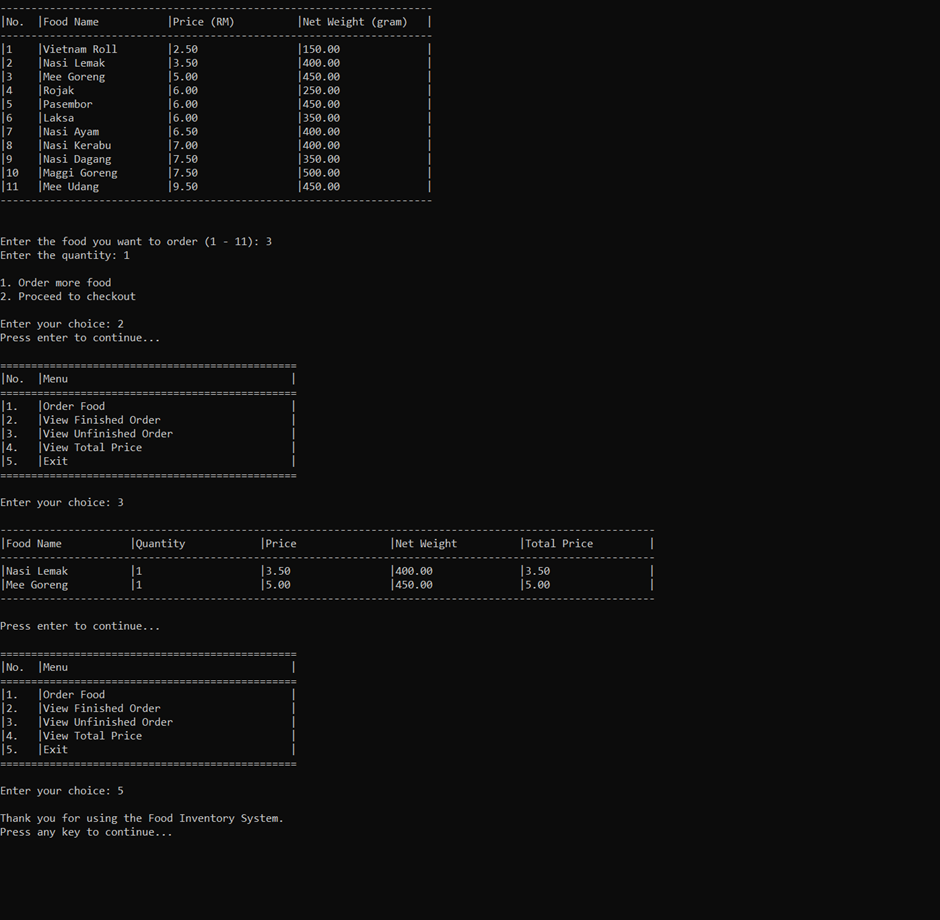
**Sample Input/Output**

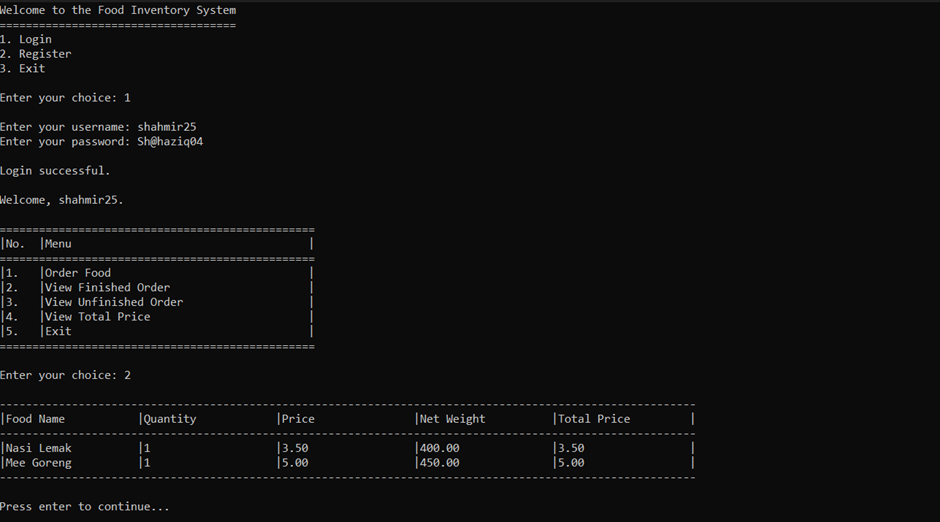
ADMIN INTERFACE

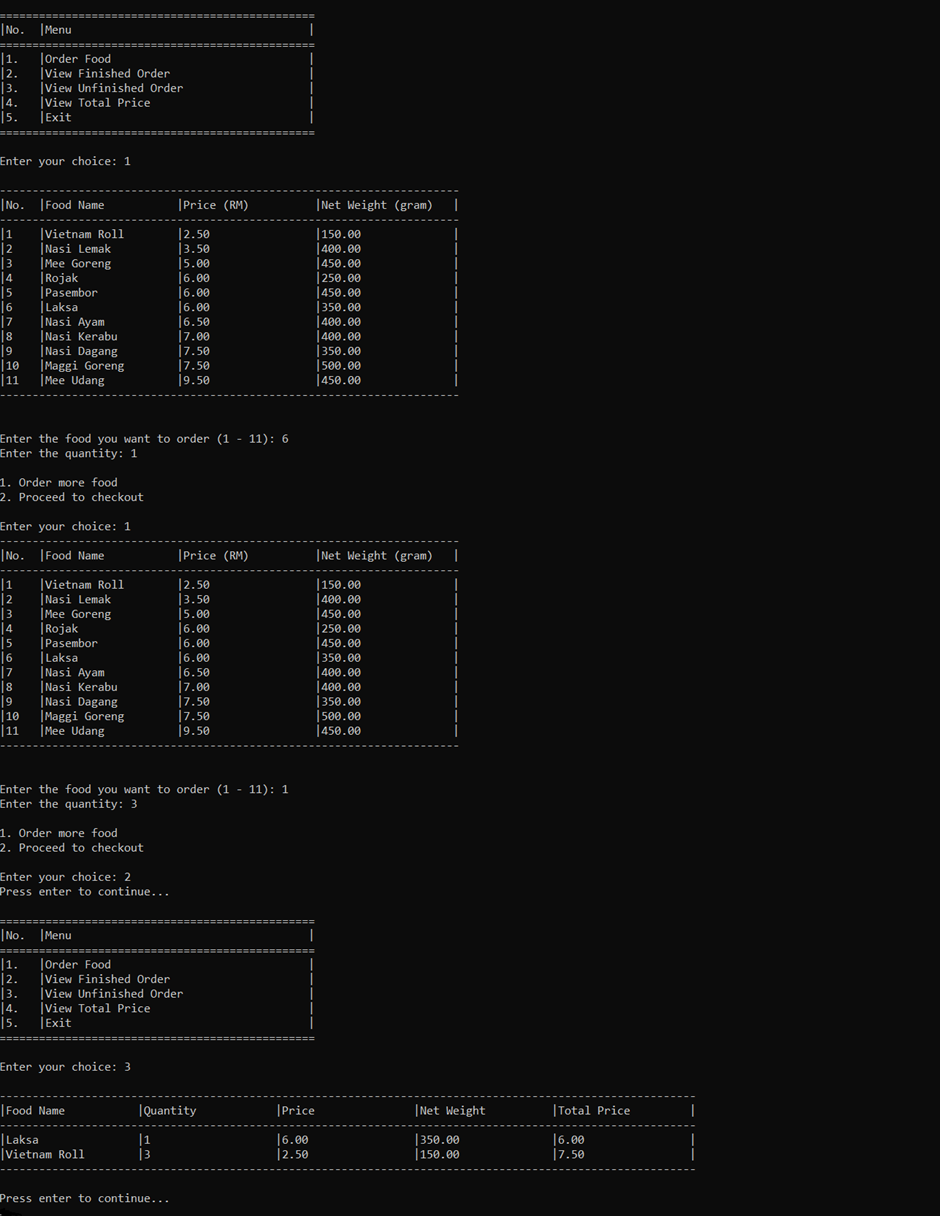


USER INTERFACE











# 10.0 CONCLUSION

In the rapidly evolving landscape of food service, the need for systems that enhance efficiency, accuracy, and user experience is more critical than ever. The Food Inventory System is a testament to this need, offering a comprehensive solution that addresses several key areas in food inventory management.

Firstly, the system automates complex calculations such as total price and discounts. This feature not only eliminates the need for manual computations but also significantly reduces waiting times for customers. By ensuring that customers receive their orders and invoices promptly, the system enhances customer satisfaction and loyalty.

Secondly, the Food Inventory System is a valuable tool for employees. It enables them to perform tasks more swiftly and accurately, thereby increasing their productivity. The system's user-friendly interface and intuitive design make it easy for staff to navigate and use, reducing the learning curve typically associated with new systems.

Moreover, the Food Inventory System is designed specifically to meet the unique requirements of a food inventory management system. Its features and functionalities align closely with the operational needs of food service establishments, making it a practical and effective solution.

The system also prioritizes data security and integrity. It securely stores and retrieves customer data and order information, ensuring that sensitive information is protected at all times. This commitment to data security not only complies with data protection regulations but also builds trust with customers.

In conclusion, the Food Inventory System is more than just a project. It represents a significant advancement in the realm of food inventory management. By enhancing operational efficiency, improving the customer experience, and providing a seamless and efficient ordering process, the system sets a new standard for food service establishments. As we look to the future, we are confident that the Food Inventory System will continue to evolve and adapt to meet the changing needs of the food service industry.

# 11.0 REFERENCES

Stack Overflow, (4 December 2013), Casting Objects in Java. Accessed on 17 July 2023, obtained from https://stackoverflow.com/questions/5306835/casting-objects-in-java

Stack Overflow, (6 December 2015), “What is the use of System.in.read()?”. Accessed on 17 July 2023, obtained from https://stackoverflow.com/questions/15446689/what-is-the-use-of-system-in-read

Stack Overflow, (23 August 2011), “When/Why to call System.out.flush() in java?”. Accessed on 17 July 2023, obtained from https://stackoverflow.com/questions/7166328/when-why-to-call-system-out-flush-in-java

Stack Overflow, (23 April 2011), “How to Print Color in Console using System.out.printIn?”. Accessed on 17 July 2023, obtained from https://stackoverflow.com/questions/5762491/how-to-print-color-in-console-using-system-out-println

Stack Overflow, (14 June 2009), Java 256-bit AES Password-based Encryption. Accessed on 17 July 2023, obtained from https://stackoverflow.com/questions/992019/java-256-bit-aes-password-based-encryption