BACS2063 Data Structures and Algorithms

ASSIGNMENT 202205

Student Name : Lee Ting Le

Student ID : 22PMR06024

Programme : Bachelor of Information Technology (Honours) in Software Systems Development

Tutorial Group : 2

Assignment Title : Food Ordering and Booking Service

Declaration

- I confirm that I have read and complied with all the terms and conditions of Tunku Abdul Rahman University College's plagiarism policy.
- I declare that this assignment is free from all forms of plagiarism and for all intents and purposes is my own properly derived work.

Student's signature

17/9/2022

Date

Table of Contents

1. Introduction

1.1 Overview

In this project, the application we have chosen is catering (meal) service. In this application, we will mainly focus on allowing customers to place their food order by using the application, which is a restaurant ordering service. The customer is given a meal menu so that they can add their desired food into the food cart.

1.2 Order Module

Moreover, after the customer has confirmed their food orders, the orders will be sent to the kitchen and start preparing. In order to ensure the ordered first will be disposed of first, the application will only allow the kitchen staff to prepare the first coming order. Thus, customers do not need to take a longer time in waiting for the orders. Other than this, in this system, staff are able to view the in processing orders and new coming orders which has been arranged in sequence based on the ordering date time. Besides, in order to handle uncertainty situations such as users accidentally placing orders twice, this system has provided a cancellation function which allows the user to cancel on the specific order. Furthermore, this system has also provided search functionality which allows staff to get the details of order by searching on the order ID.

2. Abstract Data Type (ADT) Specification

enQueue(T newElement)	
Description	Add a new element to end of the queue
Precondition	-
Postcondition	The newElement has been added to the end of the queue
Return	-

T deQueue()	
Description	Remove element from the queue and the element to be removed is always from the front of the queue
Precondition	The queue is not empty
Postcondition	The element is removed from the queue
Return	Return the element from the front of queue

T moveFirstToLast()	
Description	Remove the front element and add it back to the end of the queue
Precondition	There is at least 2 elements in the queue
Postcondition	The front element will become the last element of the queue, and the second element will be the front element
Return	The element being removed from the queue.

Integer getNumOfItems()	
Description	Get the total number of elements in the queue
Precondition	-

Postcondition	The total number of elements in queue remains unchanged
Return	The total counts of elements present in the queue

Boolean isEmpty()	
Description	Check whether the Queue is empty
Precondition	-
Postcondition	The queue remains unchanged
Return	True if the queue is empty otherwise false

Boolean isFull()	
Description	Check whether the Queue is full
Precondition	-
Postcondition	The queue remains unchanged
Return	True if the queue is full otherwise false

clear()	
Description	Remove all elements in the queue
Precondition	The queue is not empty
Postcondition	The current queue will become an empty queue
Return	-

Iterator<T> iterator()	
Description	Loop all the element in the iterator object
Precondition	-
Postcondition	The iterator remain unchanged
Return	The iterator object

3. ADT Implementation

3.1 Overview of ADT

Circular Array

Circular array has been chosen to be used as the implementation method. The reason why I chose circular array is because it resolves the problem of rightward drift as it is formed in a circular shape by connecting the last element in the queue to the first element in the queue. Through this, it can maximize the space used as it ensures that all the empty spaces will be used. So, there will be no wasted array locations. It is totally different from other implementation methods such as linear array with dynamic front which will cause the problem of empty space not being fully used.

3.2 ADT Implementation

Interface - QueueInterface.java

```
package adt;

import java.util.Iterator;

/**
 *
 * @author Ting Le
 */
public interface QueueInterface<T> {

    //Add a new element to end of the queue
    public void enQueue(T newElement);

    //Remove the first element in the Queue
    public T deQueue();

    //Remove the first element and add it back to the Queue at the
last position
    public T moveFirstToLast();

    //Retrieve the total number of elements in a Queue
    public int getNumOfItems();

    //Check whether the Queue is empty
    public boolean isEmpty();
```

```
    //Check whether the Queue is full
    public boolean isFull();

    //It is used to increase the Queue size/length when it is Full
    public void makeSpace();

    //Clear or Reset the Queue to an initial Queue
    public void clear();

    //It is used to loop the elements inside a Queue
    public Iterator<T> iterator();
}
```

Implementation - CircularArrayQueue.java

```
package adt;

import java.util.Iterator;

/**
 *
 * @author Ting Le
 */
public class CircularArrayQueue<T> implements QueueInterface<T> {

    private T[] arr;
    private int frontIndex;
    private int rearIndex;
    private static final int DEFAULT_MAX_BOOK = 5;

    public CircularArrayQueue() {
        this(DEFAULT_MAX_BOOK);
    }

    public CircularArrayQueue(int initialCapacity) {

        arr = (T[]) new Object[initialCapacity + 1];
        frontIndex = -1;
        rearIndex = -1;

    }

    @Override
    public void enQueue(T newElement) {

        if (isFull()) {
            makeSpace();
            rearIndex = (arr.length / 2);
```

```
            arr[rearIndex] = newElement;
        } else {
            if (frontIndex == -1) {
                frontIndex = 0;
            }
            rearIndex = (rearIndex + 1) % arr.length;
            arr[rearIndex] = newElement;
        }
    }

    @Override
    public T deQueue() {
        //create a temporary Queue
        T temp = null;

        if (!isEmpty()) {
            //assign the top queue to temp
            temp = arr[frontIndex];
            if (frontIndex == rearIndex) { //return to ori position
when Queue bcm empty after the deletion
                frontIndex = -1;
                rearIndex = -1;
            } else {
                frontIndex = (frontIndex + 1) % arr.length;
            }
        } else {
            return null;
        }

        return temp;
    }

    @Override
    // Method used to display queue element
    public T moveFirstToLast() {
        T temp = null;
        if (!isEmpty()) {
            temp = deQueue();
            enQueue(temp);
        } else {
            return null;
        }
        return temp;
    }

    @Override
    public int getNumOfItems() {
        if (rearIndex >= frontIndex) {
            return (rearIndex - frontIndex + 1);
        } else {
            return (arr.length - (frontIndex - rearIndex) + 1);
```

```
        }
    }

    @Override
    public boolean isEmpty() {
        return frontIndex == -1;
    }

    @Override
    public boolean isFull() {
        return ((frontIndex == 0 && rearIndex == arr.length - 1) ||
frontIndex == rearIndex + 1);
    }

    @Override
    public void makeSpace() {
        //Copy current array to a new temporary array
        T[] tempArr = arr;

        //assign new memory space to current array;
        arr = (T[]) new Object[arr.length * 2];

        //copy the temporary array to current array
        int i = 0;
        for (T temp : tempArr) {
            arr[i] = temp;
            i++;
        }
    }

    @Override
    public void clear() {
        frontIndex = -1;
        rearIndex = -1;
    }

    @Override
    public Iterator<T> iterator() {
        // OverRiding Default List Iterator //
        Iterator<T> it = new Iterator<T>() {
            private int currentIndex = frontIndex;

            @Override
            public boolean hasNext() {
                // OverRiding Default hasNext  Method//
                return currentIndex < getNumOfItems() &&
arr[currentIndex] != null;
            }

            @Override
            public T next() {
```

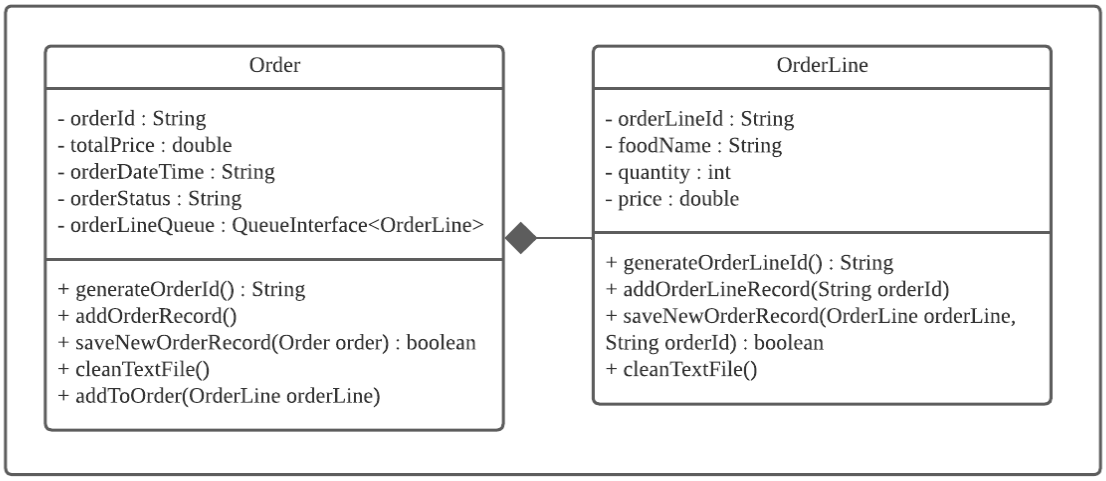
```
                // OverRiding Default next  Method//
                return arr[currentIndex++];
            }
        };
        return it;
    }

}
```

4. Entity Classes

4.1 Entity Class Diagram

Order Module



4.2 Entity Class Implementation

OrderLine.java

```
package entity;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

/**
 *
 * @author Ting Le
 */
public class OrderLine {

    //attributes
    private String orderLineId;
    private String foodName;
    private int quantity;
    private double price;
```

```
    //Default Constructor
    public OrderLine() {
        this("", "", 0, 0);
    }

    //Parameterized Constructor
    public OrderLine(String ID, String foodName, int quantity,
double price) {
        this.orderLineId = ID;
        this.foodName = foodName;
        this.quantity = quantity;
        this.price = price;
    }

    //Getter & Setter
    public String getFoodName() {
        return foodName;
    }

    public String getID() {
        return orderLineId;
    }

    public double getPrice() {
        return price;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setFoodName(String foodName) {
        this.foodName = foodName;
    }

    public void setID(String ID) {
        this.orderLineId = ID;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    //Methods/Functions
    public String generateOrderLineId() {
        String id = "";
```

```
        try {
            BufferedReader br = new BufferedReader(new
FileReader("src/db/OSrderLine.txt"));
            String nextLine, previousLine = null;

            //if there are no record
            //nextLine will be null and assigned an id "OL001" with
it
            if ((nextLine = br.readLine()) == null) {
                id = "OL001";
            } else {
                //find the last record and id + 1
                //because the new orderLineId = last orderLineId +
1
                while (nextLine != null) {
                    previousLine = nextLine;
                    nextLine = br.readLine();
                }

                String[] split = previousLine.split("\t");
                int no =
Integer.parseInt(split[1].replaceAll("\\D+", "")) + 1;
                id = "OL" + String.format("%03d", no);
            }
        } catch (IOException e) {
            System.out.println("ID An error occurred.");
            e.printStackTrace();
        }
        return id;
    }

    //Add an OrderLine record into OrderLine.txt
    public void addOrderLineRecord(String orderId) {
        //Save OrderLine Record
        try {
            PrintWriter out = new PrintWriter(new
BufferedWriter(new FileWriter("src/db/OrderLine.txt", true)));

            out.print(orderId + "\t"
                    + generateOrderLineId() + "\t"
                    + foodName + "\t"
                    + quantity + "\t"
                    + price + "\n");

            out.close();

        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
```

```
    //Save New OrderLine Record into OrderLine.txt
    public boolean saveNewOrderLineRecord(OrderLine orderLine,
String orderId) {
        boolean isSuccess = false;

        try {
            //write file
            PrintWriter out = new PrintWriter(new
BufferedWriter(new FileWriter("src/db/OrderLine.txt", true)));

            out.print(orderId + "\t"
                    + orderLine.getID() + "\t"
                    + orderLine.getFoodName() + "\t"
                    + orderLine.getQuantity() + "\t"
                    + orderLine.getPrice() + "\n");

            out.close();
            isSuccess = true;
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
        return isSuccess;
    }

    //Clean everything in OrderLine.txt
    public void cleanTextFile() {
        try {
            FileWriter fw = new FileWriter("src/db/OrderLine.txt",
false);
            PrintWriter pw = new PrintWriter(fw, false);
            pw.flush();
            pw.close();
            fw.close();

        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

}
```

Order.java

```
package entity;

import adt.CircularArrayQueue;
import adt.QueueInterface;
```

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.text.SimpleDateFormat;
import java.util.Calendar;

/**
 *
 * @author Ting Le
 */
public class Order {

    //ASSIGNMENT SCOPE OF WORK (B)
    //1 Order has MANY Order Lines
    private QueueInterface<OrderLine> orderLineQueue = new
CircularArrayQueue<>();

    //attributes
    private String orderId;
    private double totalPrice;
    private String orderDateTime;
    private String orderStatus;

    //Parameterized Constructor
    public Order(double totalPrice) {
        this.orderId = generateOrderId();
        this.totalPrice = totalPrice;
        this.orderDateTime = new SimpleDateFormat("yyyy/MM/dd
HH:mm:ss").format(Calendar.getInstance().getTime());
        this.orderStatus = "Preparing";

        //For every new Order will assign a new Order Queue
        //Avoid the previous order queue and new order queue mix
together
        orderLineQueue = new CircularArrayQueue<>();
    }

    public Order(String orderId, double totalPrice, String
orderDateTime, String orderStatus) {
        this.orderId = orderId;
        this.totalPrice = totalPrice;
        this.orderDateTime = orderDateTime;
        this.orderStatus = orderStatus;
    }

    public Order() {
    }
```

```
    //Getter & Setter
    public String getOrderDateTime() {
        return orderDateTime;
    }

    public String getOrderId() {
        return orderId;
    }

    public QueueInterface<OrderLine> getOrderLineQueue() {
        return orderLineQueue;
    }

    public String getOrderStatus() {
        return orderStatus;
    }

    public double getTotalPrice() {
        return totalPrice;
    }

    public void setOrderDateTime(String orderDateTime) {
        this.orderDateTime = orderDateTime;
    }

    public void setOrderId(String orderId) {
        this.orderId = orderId;
    }

    public void setOrderLineQueue(QueueInterface<OrderLine>
OrderLineQueue) {
        this.orderLineQueue = OrderLineQueue;
    }

    public void setOrderStatus(String orderStatus) {
        this.orderStatus = orderStatus;
    }

    public void setTotalPrice(double totalPrice) {
        this.totalPrice = totalPrice;
    }

    //Methods
    //Generate New Order ID for new Order Record
    public String generateOrderId() {
        String id = "";
        try {
            BufferedReader br = new BufferedReader(new
FileReader("src/db/Order.txt"));
            String nextLine, previousLine = null;
```

```
            //if there are no record
            //nextLine will be null and assigned an id "OL001" with
it
            if ((nextLine = br.readLine()) == null) {
                id = "OR001";
            } else {
                //find the last record and id + 1
                //because the new orderLineId = last orderLineId +
1
                while (nextLine != null) {
                    previousLine = nextLine;
                    nextLine = br.readLine();
                }

                String[] split = previousLine.split("\t");
                int no =
Integer.parseInt(split[0].replaceAll("\\D+", "")) + 1;
                id = "OR" + String.format("%03d", no);
            }
        } catch (IOException e) {
            System.out.println("ID An error occurred.");
            e.printStackTrace();
        }
        return id;
    }

    //Add an Order record into Order.txt
    public void addOrderRecord() {
        //Save Order Record
        try {
            PrintWriter out = new PrintWriter(new
BufferedWriter(new FileWriter("src/db/Order.txt", true)));

            out.print(generateOrderId() + "\t"
                    + totalPrice + "\t"
                    + orderDateTime + "\t"
                    + orderStatus + "\n");

            out.close();
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }

    //Save New Order Record into Order.txt
    public boolean saveNewOrderRecord(Order order) {
        boolean isSuccess = false;

        try {
            //write file
```

```
            PrintWriter out = new PrintWriter(new
BufferedWriter(new FileWriter("src/db/Order.txt", true)));

            out.print(order.getOrderId() + "\t"
                    + order.getTotalPrice() + "\t"
                    + order.getOrderDateTime() + "\t"
                    + order.getOrderStatus() + "\n");

            out.close();
            isSuccess = true;
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
        return isSuccess;
    }

    //Clean everything in Order.txt
    public void cleanTextFile() {
        try {
            FileWriter fw = new FileWriter("src/db/Order.txt",
false);
            PrintWriter pw = new PrintWriter(fw, false);
            pw.flush();
            pw.close();
            fw.close();

        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    public void addToOrder(OrderLine orderLine) {
        orderLineQueue.enQueue(orderLine);
    }

}
```

5. Client Program

Queue ADT

Queue is selected as the collection ADT that is used in my client classes because it follows the First-In-First Out (FIFO) principle. In my client classes, it only allows users to update the order status in sequence instead of allowing them to randomly update it. This means that, if the user needs to update the second order record status, they will need to update the first order record first before they can update the following one. This is because I have implied the First-Come-First-Serve concept in my client classes to ensure the first customer order will be the first to complete the preparation. Therefore, queue is the best ADT because it only allows data to be accessed sequentially, unlike list which can be accessed randomly and stack which is not able to support FIFO principle.

Console-based prototypes

Main - OrderManagement.java

```
package client;

/**
 *
 * @author Ting Le
 */
import entity.Order;
import java.util.Scanner;

public class OrderManagement {

    private OrderManager om = new OrderManager();
    private Scanner scanner = new Scanner(System.in);

    public void orderApp() {
        String choice;
        int orderChoice;
        String id;
        String updateChoice;

        do {
            System.out.println("|====================================|");
            System.out.println("|          Order Management          |");
            System.out.println("|====================================|");
            System.out.println("| 1. Display Orders                  |");
            System.out.println("| 2. Display All Order Lines         |");
            System.out.println("| 3. Search Order                    |");
            System.out.println("| 4. Update Order Status             |");
```

```
            System.out.println("| 5. Delete Order                    |");
            System.out.println("| 0. Close The System                |");
            System.out.println("|====================================|");

            System.out.printf("Select one choice : ");
            choice = scanner.next();

            while (!choice.equals("0") && !choice.equals("1") &&
!choice.equals("2") && !choice.equals("3") && !choice.equals("4") &&
!choice.equals("5")) {
                System.out.printf("Please reselect your choice : ");
                choice = scanner.next();
            }

            switch (choice) {
                case "1":
                    System.out.println("\n|====================|");
                    System.out.println("| 1. All             |");
                    System.out.println("| 2. Preparing       |");
                    System.out.println("| 3. Ready To Pick Up |");
                    System.out.println("|====================|");

                    System.out.printf("Select one choice : ");
                    orderChoice = scanner.nextInt();

                    while (orderChoice > 3 || orderChoice < 1) {
                        System.out.printf("Please reselect your choice : ");
                        orderChoice = scanner.nextInt();
                    }

                    switch (orderChoice) {
                        case 1:
                            System.out.println("\n\n\t\t\tDisplay All
Orders");
                            System.out.println("\t\t\t-----------------\n");
                            break;
                        case 2:
                            System.out.println("\n\n\t\t\tDisplay Preparing
Orders");

System.out.println("\t\t\t-----------------------\n");
                            break;
                        case 3:
                            System.out.println("\n\n\t\t\tDisplay Ready To
Pick Up Orders");

System.out.println("\t\t\t------------------------------\n");
                            break;
                    }

                    System.out.println("Order ID\tTotal Price\tOrder Date
Time\t\tStatus");

System.out.println("==========================================================
===============");
                    System.out.println(om.displayOrder(orderChoice));
```

```
                    om.clearQueue();
                    break;
                case "2":
                    System.out.println("\n\n\t\tDisplay All Order Lines");
                    System.out.println("\t\t-----------------------\n");
                    System.out.println("Order Line
ID\tName\t\tQuantity\tPrice");

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

                    om.clearQueue();
                    break;
                case "3":
                    System.out.println("\n\n\t\t\tSearch Order");
                    System.out.println("\t\t\t------------\n");
                    System.out.print("Enter order ID to search : ");
                    id = scanner.next();
                    om.searchOrder(id);

                    om.clearQueue();
                    break;
                case "4":
                    System.out.println("\n\n\t\t\tUpdate Order Status");
                    System.out.println("\t\t\t-------------------\n");

                    //Please be aware that only preparing order can be update
to ready order
                    //and will only update the latest preparing order
                    //this is because first order generated should be the
first order to be pick up
                    if (om.getLatestPreparingOrder()) {
                        System.out.printf("\nDo you wish do update current
order status [Preparing] -> [Ready To Pick Up] \nAre you sure to update ?
(Yes/No) : ");
                        updateChoice = scanner.next();

                        while (!updateChoice.toUpperCase().equals("YES") &&
!updateChoice.toUpperCase().equals("NO")) {
                            System.out.print("Please enter only (Yes/No) : ");
                            updateChoice = scanner.next();
                        }

                        if (updateChoice.toUpperCase().equals("YES")) {
                            if (om.updateOrderStatus()) {
                                System.out.println("Successfully Update Order
!\n");
                            } else {
                                System.out.println("Failed to Update Order
!\n");
                            }
                            om.clearQueue();
                        } else {
                            om.clearQueue();
```

```
                            System.out.println();
                            break;
                        }
                    }
                    break;
                case "5":
                    System.out.println("\n\n\t\t\tDelete Order");
                    System.out.println("\t\t\t------------\n");
                    System.out.print("Enter order id to delete the order and
its order details (Not Recoverable!) : ");
                    id = scanner.next();

                    om.deleteOrder(id);

                    om.clearQueue();
                    break;
                default:
                    System.out.println("\n|====================|");
                    System.out.println("|  Closing System...  |");
                    System.out.println("|====================|");
            }
        } while (!choice.equals("0"));
    }

    public static void main(String[] args) {
        new OrderManagement().orderApp();
    }
}
```

OrderManager.java (Contain all ADT methods used, so that driver program contain only design code and called methods)

```
package client;

import adt.CircularArrayQueue;
import adt.QueueInterface;
import entity.Order;
import entity.OrderLine;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Iterator;

/**
 *
 * @author Ting Le
 */
public class OrderManager {

    //REQUIREMENT : Scope of Works (c)
    //--- in this client, entity Order and OrderLine has separate queue
```

```
    //--- but each OrderLine has its own specific order
    private QueueInterface<Order> orderQueue = new CircularArrayQueue<>();
    private QueueInterface<OrderLine> orderLineQueue = new
CircularArrayQueue<>();

    //Get All Order record from Order.txt
    public void readOrderRecord() {

        try {
            BufferedReader br = new BufferedReader(new
FileReader("src/db/Order.txt"));
            String line;

            //Read Order.txt line by line
            //Save the record into OrderQueue
            while ((line = br.readLine()) != null) {
                String[] split = line.split("\t");
                orderQueue.enQueue(new Order(split[0],
Double.parseDouble(split[1]), split[2], split[3]));
            }
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }

    //Get All OrderLine record from OrderLine.txt
    //Move every Order Line into OrderLineQueue in their respective Order
object
    //By comparing their Order ID
    public void readOrderLineRecord() {

        readOrderRecord();
        Iterator<Order> itr = orderQueue.iterator();

        Order order = null;

        if (itr.hasNext()) {
            order = itr.next();
        }

        try {
            BufferedReader br = new BufferedReader(new
FileReader("src/db/OrderLine.txt"));
            String line;

            while ((line = br.readLine()) != null) {

                String[] split = line.split("\t");
                OrderLine ol = new OrderLine(split[1], split[2],
Integer.parseInt(split[3]), Double.parseDouble(split[4]));
                orderLineQueue.enQueue(ol);

                if (order.getOrderId().equals(split[0])) {
                    order.addToOrder(ol);
                } else {
```

```
                    order = itr.next();
                    order.addToOrder(ol);
                }

            }
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }

    //Display All Order Record
    public String displayOrder(int orderChoice) {

        readOrderRecord();
        Iterator<Order> itr = orderQueue.iterator();

        String displayText = "";
        int count = 0;

        while (itr.hasNext()) {
            Order displayOrder = itr.next();

            if (orderChoice == 1) {
                displayText += displayOrder.getOrderId() + "\t\t"
                        + displayOrder.getTotalPrice() + "\t\t"
                        + displayOrder.getOrderDateTime() + "\t"
                        + displayOrder.getOrderStatus() + "\n\n";
                count++;

            } else if (orderChoice == 2 &&
displayOrder.getOrderStatus().equals("Preparing")) {
                displayText += displayOrder.getOrderId() + "\t\t"
                        + displayOrder.getTotalPrice() + "\t\t"
                        + displayOrder.getOrderDateTime() + "\t"
                        + displayOrder.getOrderStatus() + "\n\n";
                count++;

            } else if (orderChoice == 3 &&
displayOrder.getOrderStatus().equals("Ready To Pick Up")) {
                displayText += displayOrder.getOrderId() + "\t\t"
                        + displayOrder.getTotalPrice() + "\t\t"
                        + displayOrder.getOrderDateTime() + "\t"
                        + displayOrder.getOrderStatus() + "\n\n";
                count++;
            }
        }

        if (count <= 0) {
            displayText = "No Record Found ! \n";
        }
        return displayText;
    }

    //Display All OrderLine Record
    public String displayOrderLine() {
```

```
        readOrderLineRecord();
        Iterator<OrderLine> itr = orderLineQueue.iterator();

        String displayText = "";

        while (itr.hasNext()) {
            OrderLine displayOrderLine = itr.next();

            displayText += displayOrderLine.getID() + "\t\t"
                    + displayOrderLine.getFoodName() + "\t\t"
                    + displayOrderLine.getQuantity() + "\t\t"
                    + displayOrderLine.getPrice() + "\n\n";
        }
        return displayText;
    }

    //Get Order Record By Order ID
    public Order getOrderById(String id) {

        Iterator<Order> itr = orderQueue.iterator();

        Order orderFound = null;

        //Loop the Order record to find same record ID
        while (itr.hasNext()) {
            Order order = itr.next();

            if (id.toUpperCase().equals(order.getOrderId())) {
                orderFound = order;
            }
        }

        return orderFound;
    }

    //Search Order and OrderLine Record
    public void searchOrder(String id) {

        readOrderLineRecord();

        int count = 1;
        Order orderFound = getOrderById(id);

        //If record found, print record
        if (orderFound != null) {

            Iterator<OrderLine> itrOL =
orderFound.getOrderLineQueue().iterator();


System.out.println("|=====================================================|"
);
            System.out.printf("|\t\t\t%-21s \t\t|\n",
orderFound.getOrderStatus());
```

```
System.out.println("|=========================================================|"
);
            System.out.printf("| ID : %-5s \t\t\t   %-20s |\n",
orderFound.getOrderId(), orderFound.getOrderDateTime());
            System.out.printf("| Total Price : RM%6.2f\t\t\t\t|\n",
orderFound.getTotalPrice());

System.out.println("|=========================================================|\
n");


System.out.println("|=========================================================|"
);
            System.out.println("|\t\t\tOrder Details\t\t\t|");

System.out.println("|=========================================================|"
);
            System.out.println("| No | Food Name\t\t     | Quantity | Price
|");

            while (itrOL.hasNext()) {
                OrderLine ol = itrOL.next();
                System.out.printf("| %-3d| %-30s|    %-5d | %-5.2f |\n",
count, ol.getFoodName(), ol.getQuantity(), ol.getPrice());
                count++;
            }


System.out.println("|=========================================================|\
n\n");

        } else {
            System.out.println("Order Record Not Found !\n");
        }
    }

    //Get the latest "Preparing" Order record
    public boolean getLatestPreparingOrder() {

        readOrderRecord();
        Iterator<Order> itr = orderQueue.iterator();

        boolean hasPreparing = false;
        Order latestPreparingOrder = null;

        //Get First/Latest "Preparing" Order Record
        while (itr.hasNext() && !hasPreparing) {
            latestPreparingOrder = itr.next();

            if (latestPreparingOrder.getOrderStatus().equals("Preparing")) {
                hasPreparing = true;
            }
        }

        //If got "Preparing" Order, print to user
        if (hasPreparing) {
```

```
            System.out.println("The Latest Preparing Order is ");

System.out.println("|==========================================================
|");
            System.out.println("| "
                    + latestPreparingOrder.getOrderId() + "\t\t"
                    + latestPreparingOrder.getTotalPrice() + "\t"
                    + latestPreparingOrder.getOrderDateTime() + "\t"
                    + latestPreparingOrder.getOrderStatus() + " |");

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

        } else {
            System.out.println("No Preparing Order Found !\n");
        }
        clearQueue();

        return hasPreparing;
    }

    //Update Order status from "Preparing" -> "Ready To Pick Up"
    public boolean updateOrderStatus() {

        readOrderRecord();
        Iterator<Order> itr = orderQueue.iterator();

        Order order = null;
        int count = 0;
        boolean isSuccess = false;
        boolean hasPreparing = false;

        //search for "Preparing" Order
        while (itr.hasNext() && !hasPreparing) {
            order = itr.next();

            if (order.getOrderStatus().equals("Preparing")) {
                hasPreparing = true;
            } else {
                count++;
            }

            if (hasPreparing) {
                //Update New Status
                orderQueue.deQueue();
                orderQueue.enQueue(new Order(order.getOrderId(),
order.getTotalPrice(), order.getOrderDateTime(), "Ready To Pick Up"));
                count++;
            } else {
                moveFirstOrderToLast();
            }
        }

        //Rearrange the record
        for (int i = 0; i < numOfOrder() - count; i++) {
            moveFirstOrderToLast();
```

```
        }

        //Clean Order.txt
        order.cleanTextFile();

        //Save the Updated Order Record into Order.txt
        for (int i = 0; i < numOfOrder(); i++) {
            order = moveFirstOrderToLast();
            isSuccess = order.saveNewOrderRecord(order);
        }
        return isSuccess;
    }

    //Delete selected Order and OrderLine record
    public void deleteOrder(String id) {

        readOrderLineRecord();
        Iterator<Order> itr = orderQueue.iterator();
        Iterator<Order> itr2 = orderQueue.iterator();

        Order order = null;
        OrderLine ol;
        OrderLine allOL = null;
        boolean hasEqual = false;
        boolean isSuccessOrder = false;
        boolean isSuccessOrderLine = false;
        int countOrder = 0;
        int countOrderLine = 0;

        Order orderFound = getOrderById(id);

        //If Order record found
        if (orderFound != null) {

            //Loop Order Queue and remove the record found from Order Queue
            while (itr.hasNext() && !hasEqual) {
                order = itr.next();

                if (order.equals(orderFound)) {
                    orderQueue.deQueue();
                    hasEqual = true;
                } else {
                    moveFirstOrderToLast();
                    countOrder++;
                }
            }

            //All OrderLine Record in this orderFound
            Iterator<OrderLine> itrOL =
orderFound.getOrderLineQueue().iterator();

            hasEqual = false;

            while (itrOL.hasNext()) {
                ol = itrOL.next();
```

```
                //Loop ALL OrderLine Queue and remove the matched record
                for (int i = 0; i < numOfOrderLine() && !hasEqual; i++) {
                    allOL = orderLineQueue.deQueue();

                    if (ol.equals(allOL)) {
                        hasEqual = true;
                    } else {
                        orderLineQueue.enQueue(allOL);
                        countOrderLine++;
                    }
                }
                hasEqual = false;
            }

            //Rearrange Order Queue
            for (int i = 0; i < numOfOrder() - countOrder; i++) {
                moveFirstOrderToLast();
            }

            //Rearrange OrderLine Queue
            for (int i = 0; i < numOfOrderLine() - countOrderLine; i++) {
                moveFirstOLToLast();
            }

            //Rewrite Order.txt with New Order Records
            order.cleanTextFile();

            for (int i = 0; i < numOfOrder(); i++) {
                order = moveFirstOrderToLast();
                isSuccessOrder = order.saveNewOrderRecord(order);
            }

            //Rewrite OrderLine.txt with New OrderLine Records
            allOL.cleanTextFile();

            for (int i = 0; i < numOfOrder(); i++) {
                order = moveFirstOrderToLast();
                for (int j = 0; j < order.getOrderLineQueue().getNumOfItems();
j++) {
                    allOL = moveFirstOLToLast();
                    isSuccessOrderLine = allOL.saveNewOrderLineRecord(allOL,
order.getOrderId());
                }
            }

        } else {
            System.out.println("Order Record Not Found !\n");
        }

        if (isSuccessOrder && isSuccessOrderLine) {
            System.out.println("Successfully Deleted Order and its Order
Details !\n");
        }
    }

    //adt methods
```

```
    //get total number of items in order queue
    public int numOfOrder() {
        return orderQueue.getNumOfItems();
    }

    //get total number of items in orderline queue
    public int numOfOrderLine() {
        return orderLineQueue.getNumOfItems();
    }

    //Move First Order record to the last
    public Order moveFirstOrderToLast() {
        return orderQueue.moveFirstToLast();
    }

    //Move First OrderLine record to the last
    public OrderLine moveFirstOLToLast() {
        return orderLineQueue.moveFirstToLast();
    }

    //clear Order and OrderLine Queue
    public void clearQueue() {
        orderQueue.clear();
        orderLineQueue.clear();
    }
}
```

Output Screenshot

Description	Screenshots
This is the Order Management Menu which displays all the available functions	

```
|=====================================|
|           Order Management          |
|=====================================|
| 1. Display Orders                   |
| 2. Display All Order Lines          |
| 3. Search Order                     |
| 4. Update Order Status              |
| 5. Delete Order                     |
| 0. Close The System                 |
|=====================================|
Select one choice : 1
```

There will be 3 options for user to choose which order status they want to display	

```
|====================|
| 1. All             |
| 2. Preparing       |
| 3. Ready To Pick Up |
|====================|
Select one choice : 1


                        Display All Orders
                        ------------------

Order ID        Total Price     Order Date Time         Status
=========================================================================
OR001           15.0            2022/09/01 18:38:05     Ready To Pick Up

OR002           5.0             2022/09/01 18:38:05     Ready To Pick Up
```

This is the sample output after user choose to display all order lines	

```
                Display All Order Lines
                -----------------------

Order Line ID   Name            Quantity        Price
==============================================================
OL001           fish            1               5.0

OL002           fish            1               5.0

OL003           chicken         10              10.0
```

This is the search order function which will display the order and its order details	

```
                        Search Order
                        ------------


Enter order ID to search : or001
|=========================================================|
|                        Ready To Pick Up                 |
|=========================================================|
| ID : OR001                        2022/09/01 18:38:05   |
| Total Price : RM 15.00                                  |
|=========================================================|


|=========================================================|
|                        Order Details                    |
|=========================================================|
| No | Food Name                      | Quantity | Price |
| 1  | fish                           |    1     | 5.00  |
| 2  | fish                           |    1     | 5.00  |
|=========================================================|
```

This is the update order status function. The changes can only be viewed in the text file.	Update Order Status ---------- The Latest Preparing Order is |==========| | OR004 5.0 2022/09/01 18:38:05 Preparing | |==========| Do you wish do update current order status [Preparing] -> [Ready To Pick Up] Are you sure to update ? (Yes/No) : yes Successfully Update Order !
This is the delete function and the changes can only be viewed in the text file.	Delete Order ---------- Enter order id to delete the order and its order details (Not Recoverable!) : or004 Successfully Deleted Order and its Order Details !
This is the output when the user chooses to exit the system.	|==========| | Order Management | |==========| | 1. Display Orders | | 2. Display All Order Lines | | 3. Search Order | | 4. Update Order Status | | 5. Delete Order | | 0. Close The System | |==========| Select one choice : 0 |==========| | Closing System... | |==========|