BACS2063 Data Structures and Algorithms

ASSIGNMENT 202205

Student Name	:	Low Zhi Yi
Student ID	:	21WMR02962
Programme	:	RDS
Tutorial Group	:	RDS2(S1)G2
Assignment Title	:	Catering (Meal) 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.

ZHIYI

Student's signature

10 September 2022

Date

Table of Contents

1. Introduction

I am in charge of the **Ordering Module** of Catering (Meal) Services. I chose to create, implement and apply Sorted Array List as the collection ADT in this module. The user interface for my application is console-based. I have implemented an entity class which is named Order which includes 2 collections of entity objects which are a list of menus ordered with the pax of events based on the menu ordered. I have also implemented a sub client class which is the OrderManager class to implement operation on entity and ADT. It also includes all 3 entity objects as its data fields and a variety of methods to manipulate the collection entity objects such as menu ordered and pax amount in order to perform various functionalities. OrderDriver is the main client class where the program starts and ends running.

2. Abstract Data Type (ADT) Specification

ADT Title : ADT ***Sorted Array List***
Description : Sorted array list is a linear collection of entries of a type T which allows duplicate elements in sorted order.
Objective : This ADT automatically sorts generic objects by comparing two objects with a comparator at the moment when the object is added into the list.

Method 1 : `add(T newEntry)`
Description : Adds `newEntry` to the sorted array list such that the list remains sorted.
Pre-condition : Array must not be full before adding a `newEntry`. If the array is full, expand the array.
Post-condition : `newEntry` has been added to the sorted list in a position which ensures that the list remains sorted.

Method 2 : `boolean remove(T anEntry)`
Description : Removes the first or only occurrence of `anEntry` from the sorted list.
Post-condition : `anEntry` has been removed from the list. If `anEntry` was not located in the array list, the list remains unchanged.
Returns : True if `anEntry` was located and removed, else false.

Method 3 : `T getEntry(int givenPosition)`
Description : Retrieves the entry at position `givenPosition` in the array list.
Pre-condition : `newPosition` must be between 1 to total number of entries.
Post-condition : The list remains unchanged.
Returns : The entry at position `givenPosition`.

Method 4 : `T getEntry(T anEntry)`
Description : Retrieves the entry by comparing it with `anEntry` using `equals` function.
Pre-condition : `anEntry` must match the entry in the array by the requirement in `equals`.
Post-condition : The list remains unchanged.
Returns : The entry that equals to `anEntry`.

Method 5 : `clear()`
Description : Removes all entries from the list.
Post-condition : The list is now empty. Number of entries is also cleared to zero.

Method 6 : `int getNumberOfEntries()`
Description : Gets the number of entries currently in the list.
Post-condition : The list remains unchanged.
Returns : The number of entries currently in the list.

Method 7 : `boolean isEmpty()`
Description : Determines whether the list is empty.
Post-condition : The list remains unchanged.
Returns : True if the list is empty, else false.

Method 8 : `boolean isFull()`
Description : Determines whether the list is full.
Post-condition : The list remains unchanged.
Returns : True if the list is full, else false.

Method 9 : `String toString()`
Description : Convert array items to string.
Post-condition : The list remains unchanged.
Returns : A string of array items as an output.

Method 10 : `expandArray()`
Description : Expand to double the size of the array when the list is full.
Pre-condition : Old array is full.
Post-condition : The array list items remain unchanged but the array size is doubled.

Method 11 : `makeRoom(int newPosition)`
Description : Makes room at `newPosition` by moving items behind `newPosition` one index behind starting from the last occupied index.
Pre-condition : The position is occupied by another array item.
Post-condition : Array items behind `newPosition` are moved one index behind starting from the last occupied index to the `newPosition`.

Method 12 : `removeGap(int givenPosition)`
Description : Remove a gap by moving the array items behind `givenPosition` one index front.
Pre-condition : An item is removed from the array at `givenPosition`.
Post-condition : Array items behind `givenPosition` are moved one index front to remove a gap.

3. ADT Implementation

3.1 Overview of ADT

The collection ADT created by me is the **Sorted Array List**. All of my 3 entity classes which are Order, Menu and Pax, implement the same ADT. The purpose I chose sorted array lists is to automatically sort the array items based on the IDs at the moment while the item is added into the list. However, in the Pax class, the array is sorted according to the foreign keys from Order and Menu classes. This ADT can make my list items look neat and easier to read. Sorted Array List is better than an array list which includes a sorting algorithm method in terms of time consuming. The ADT includes 12 methods stated in the ADT specification section above in detail. In short, the array list stores customer's orders, catering menu list, customer's ordered menu list and the pax of customer's event.

3.2 ADT Implementation

Java interface for Sorted Array List

```
public interface SortedListInterface<T extends Comparable<T>>
{
    public void add(T newEntry);
    public boolean remove(T anEntry);
    public T getEntry(int givenPosition);
    public T getEntry(T anEntry);
    public void clear();
    public int getNumberOfEntries();
    public boolean isEmpty();
    public boolean isFull();
}
```

Java implementation class for Sorted Array List

```
import java.io.Serializable;

public class SortedArrayList<T extends Comparable<T>>
implements SortedListInterface<T>, Serializable {

    private T[] array;
    private int numberOfEntries;
    private static final int DEFAULT_CAPACITY = 10;

    public SortedArrayList() {
        this(DEFAULT_CAPACITY);
    }

    public SortedArrayList(int initialCapacity) {
        numberOfEntries = 0;
        array = (T[]) new Comparable[initialCapacity];
    }

    @Override
    public void add(T newEntry) {
        int i = 0;
        if (isFull()) {
```

```
            expandArray();
        }
        while (i < numberOfEntries &&
newEntry.compareTo(array[i]) > 0) {
            i++;
        }
        makeRoom(i + 1);
        array[i] = newEntry;
        numberOfEntries++;
    }

    @Override
    public boolean remove(T anEntry) {
        boolean found = false;
        for (int i = 0; i < numberOfEntries; i++) {
            if (anEntry.compareTo(array[i]) == 0) {
                removeGap(i + 1);
                numberOfEntries--;
                found = true;
                break;
            }
        }
        return found;
    }

    @Override
    public void clear() {
        for (int i = 0; i < numberOfEntries; i++) {
            array[i] = null;
        }
        numberOfEntries = 0;
    }

    @Override
    public T getEntry(int givenPosition) {
        T result = null;
        if ((givenPosition >= 1) && (givenPosition <=
numberOfEntries)) {
            result = array[givenPosition - 1];
        }

        return result;
    }

    @Override
    public T getEntry(T anEntry) {
        T result = null;
        for (int i = 0; i < numberOfEntries; i++) {
            if (array[i].equals(anEntry)) {
                result = array[i];
                break;
            }
        }

        return result;
    }
```

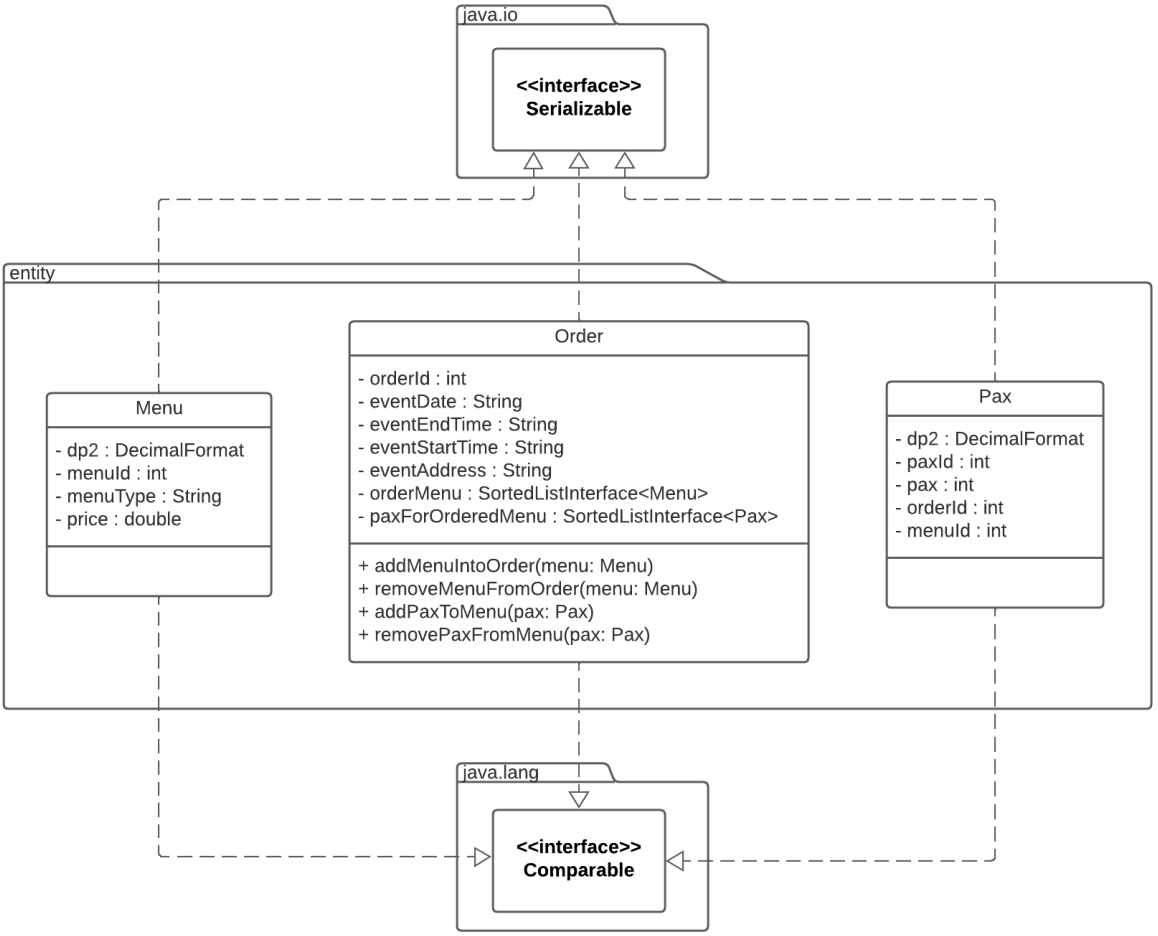
```
    @Override
    public int getNumberOfEntries() {
        return numberOfEntries;
    }

    @Override
    public boolean isEmpty() {
        return numberOfEntries == 0;
    }

    @Override
    public boolean isFull() {
        return numberOfEntries == array.length;
    }

    @Override
    public String toString() {
        String outputStr = "";
        for (int i = 0; i < numberOfEntries; ++i) {
            outputStr += array[i] + "\n";
        }

        return outputStr;
    }

    // utility methods
    private void expandArray() {
        T[] oldList = array;
        int oldSize = oldList.length;

        array = (T[]) new Comparable[2 * oldSize];

        for (int i = 0; i < oldSize; i++) {
            array[i] = oldList[i];
        }
    }

    private void makeRoom(int newPosition) {
        int newIndex = newPosition - 1;
        int lastIndex = numberOfEntries - 1;

        for (int i = lastIndex; i >= newIndex; i--) {
            array[i + 1] = array[i];
        }
    }

    private void removeGap(int givenPosition) {
        int removedIndex = givenPosition - 1;
        int lastIndex = numberOfEntries - 1;

        for (int idx = removedIndex; idx < lastIndex; idx++)
{
            array[idx] = array[idx + 1];
        }
    }
```

}

4. Entity Classes

4.1 Entity Class Diagram

(Omitted the constructors, setters, getters, toString(), equals, compareTo() in each class)



4.2 Entity Class Implementation

4.2.1 Order

```
package entity;

import adt.SortedArrayList;
import adt.SortedListInterface;

import java.io.Serializable;

/** Order - An entity class which includes sorted array lists
of Menu and Pax.
  * @author Low Zhi Yi
  */

public class Order implements Comparable<Order>, Serializable
{

    // attributes
    private int orderId;
    private String eventDate;
    private String eventStartTime;
    private String eventEndTime;
    private String eventAddress;

    private SortedListInterface<Menu> orderMenu;
    private SortedListInterface<Pax> paxForOrderedMenu;

    // constructors
    public Order() {
        this.orderMenu = new SortedArrayList<>();
        this.paxForOrderedMenu = new SortedArrayList<>();
    }

    public Order(int id) {
        this.orderId = id;
        this.orderMenu = new SortedArrayList<>();
        this.paxForOrderedMenu = new SortedArrayList<>();
    }

    // getter
    public int getOrderId() {
        return orderId;
    }

    public String getEventDate() {
        return eventDate;
    }

    public SortedListInterface<Menu> getAllMenu() {
        return orderMenu;
    }

    public SortedListInterface<Pax> getAllPax() {
```

```
        return paxForOrderedMenu;
    }

    // Setter
    public void setEventDate(int day, int month, int year) {
        String dateSeparator1 = "/", dateSeparator2 = "/";
        if (month < 10) {
            dateSeparator1 += "0";
        }
        if (day < 10) {
            dateSeparator2 += "0";
        }
        this.eventDate = year + dateSeparator1 + month +
dateSeparator2 + day;
    }

    public void setEventStartTime(int hour, int min) {
        String hourFormat = "", minFormat = ":";
        if (hour < 10) {
            hourFormat += "0";
        }
        if (min < 10) {
            minFormat += "0";
        }
        this.eventStartTime = hourFormat + hour + minFormat +
min;
    }

    public void setEventEndTime(int hour, int min) {
        String hourFormat = "", minFormat = ":";
        if (hour < 10) {
            hourFormat += "0";
        }
        if (min < 10) {
            minFormat += "0";
        }
        this.eventEndTime = hourFormat + hour + minFormat +
min;
    }

    public void setEventAddress(String eventAddress) {
        this.eventAddress = eventAddress;
    }

    public void addMenuIntoOrder(Menu menu) {
        orderMenu.add(menu);
    }

    public void removeMenuFromOrder(Menu menu) {
        orderMenu.remove(menu);
    }

    public void addPaxToMenu(Pax pax) {
        paxForOrderedMenu.add(pax);
    }
```

```
    public void removePaxFromMenu(Pax pax) {
        paxForOrderedMenu.remove(pax);
    }

    @Override
    public String toString() {
        return "Order ID\t: " + orderId
                +
"\n----------\t-----------------------------------------------
---"
                + "\nEvent Date\t: " + eventDate
                + "\nEvent Time\t: " + eventStartTime + " - "
+ eventEndTime
                + "\nEvent Address\t: " + eventAddress;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }

        final Order other = (Order) obj;

        if (this.orderId != other.orderId) {
            return false;
        }

        return true;
    }

    @Override
    public int compareTo(Order compareId) {
        int compareOrderId = ((Order)
compareId).getOrderId();
        return this.orderId - compareOrderId;
    }
}
```

4.2.2 Menu

```
package entity;

import java.io.Serializable;
import java.text.DecimalFormat;

/** Menu - An entity class of menu list.
  * @author Low Zhi Yi
  */

public class Menu implements Comparable<Menu>, Serializable {

    private DecimalFormat dp2 = new DecimalFormat("0.00");

    // attributes
    private int menuId;
    private String menuType;
    private double price;

    // constructors
    public Menu() {
    }

    public Menu(int id) {
        this.menuId = id;
    }

    public Menu(int id, String menu, double price) {
        this.menuId = id;
        this.menuType = menu;
        this.price = price;
    }

    public int getMenuId() {
        return menuId;
    }

    public String getMenuType() {
        return menuType;
    }

    public double getPrice() {
        return price;
    }

    public void setMenuType(String menuType) {
        this.menuType = menuType;
    }

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

    @Override
```

```
    public String toString() {
        String indent = "";
        if (menuId < 10) {
            indent += " ";
        }
        return indent + "[" + menuId + "]\tRM" +
dp2.format(price) + "\t\t" + menuType;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }

        final Menu other = (Menu) obj;

        if (this.menuId != other.menuId) {
            return false;
        }

        return true;
    }

    @Override
    public int compareTo(Menu compareId) {
        int compareMenuId = ((Menu) compareId).getMenuId();
        return this.menuId - compareMenuId;
    }
}
```

4.2.3 Pax

```
package entity;

import java.io.Serializable;
import java.text.DecimalFormat;

/** Pax - An entity class of event pax.
  * @author Low Zhi Yi
  */

public class Pax implements Comparable<Pax>, Serializable {

    private DecimalFormat dp2 = new DecimalFormat("0.00");

    // attributes
    private int paxId;
    private int pax;
    private int orderId;
    private int menuId;

    // constructors
    public Pax() {
    }

    public Pax(int id) {
        this.paxId = id;
    }

    public Pax(int orderId, int menuId) {
        this.orderId = orderId;
        this.menuId = menuId;
    }

    public Pax(int paxId, int pax, int orderId, int menuId) {
        this.paxId = paxId;
        this.pax = pax;
        this.orderId = orderId;
        this.menuId = menuId;
    }

    public int getPaxId() {
        return paxId;
    }

    public int getPax() {
        return pax;
    }

    @Override
    public String toString() {
        return "" + pax;
    }

    @Override
```

```
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }

        final Pax other = (Pax) obj;

        if (this.paxId != other.paxId) {
            if (this.orderId == other.orderId && this.menuId
== other.menuId) {
                return true;
            }
            return false;
        }

        return true;
    }

    @Override
    public int compareTo(Pax compareId) {
        int comparePaxId = ((Pax) compareId).getPaxId();
        return this.paxId - comparePaxId;
    }
}
```

5. Client Program

I selected Sorted Array List because array list stores items based on index and the in and out of the list is not restricted like stacks and queues. Besides, a sorted array list can sort the items based on the implementation in the `compareTo` method in the entity classes. In the sub-client class, `OrderManager`, 3 sorted array lists are created respectively for the 3 entity classes.

5.1 Sub-client class - OrderManager

```
import adt.SortedArrayList;
import adt.SortedListInterface;
import entity.Menu;
import entity.Order;
import entity.Pax;

import java.io.*;
import java.text.DecimalFormat;
import java.util.Scanner;

/**
 * OrderManager - A sub client class to implement operation on
entity and ADT
 *
 * @author Low Zhi Yi
 */
public class OrderManager {

         private  SortedListInterface<Order>  orderList  =  new
SortedArrayList<>();
          private  SortedListInterface<Menu>  menuList  =  new
SortedArrayList<>();
           private  SortedListInterface<Pax>  paxList  =  new
SortedArrayList<>();

    private Scanner input = new Scanner(System.in);
    private DecimalFormat dp2 = new DecimalFormat("0.00");

    // Constructor: dummy data for menu
    public OrderManager() {
        databaseInit();
        loadList();

        // for the first time running program
        if (menuList.getNumberOfEntries() == 0) {
            System.out.println("No product in the database");
                    System.out.println("Default  data  is  use  to
display");
```

```
            // cuisine
            menuList.add(new Menu(7, "Barbeque", 30.5));
            menuList.add(new Menu(8, "Chinese", 38.8));
            menuList.add(new Menu(9, "Indian", 26));
            menuList.add(new Menu(10, "Japanese", 36.5));
            menuList.add(new Menu(11, "Malay", 35));
            menuList.add(new Menu(12, "Thai", 36.5));
            menuList.add(new Menu(13, "Western", 45.9));
            menuList.add(new Menu(14, "Finger Food", 20));
            // occasion
            menuList.add(new Menu(1, "Baby Full Moon", 25.9));
            menuList.add(new Menu(2, "Birthday", 35));
            menuList.add(new Menu(3, "Chinese New Year", 27));
            menuList.add(new Menu(4, "Christmas", 29));
            menuList.add(new Menu(5, "Ramadan / Raya", 32));
            menuList.add(new Menu(6, "Wedding", 45));
        }
    }

    // Create database
    private void databaseInit() {
        try {
            File file1 = new File("database/order.txt");
            if (!file1.exists()) {
                file1.createNewFile();
            }
            File file2 = new File("database/menu.txt");
            if (!file2.exists()) {
                file2.createNewFile();
            }
            File file3 = new File("database/pax.txt");
            if (!file3.exists()) {
                file3.createNewFile();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private void loadList() {
        File[] files = new File("database").listFiles();
        for (File file : files) {
            System.out.println(file.getName());

              // If the files exists, load all data into driver
program
            if (file.exists()) {
                try {
```

```
                                FileInputStream fileInput = new
FileInputStream("database/" + file.getName());
                                ObjectInputStream objInput = new
ObjectInputStream(fileInput);

                    switch (file.getName()) {
                        case "order.txt" ->
                            orderList = (SortedArrayList<Order>)
objInput.readObject();
                        case "menu.txt" ->
                            menuList = (SortedArrayList<Menu>)
objInput.readObject();
                        case "pax.txt" ->
                            paxList = (SortedArrayList<Pax>)
objInput.readObject();
                        default ->
                            System.out.println("Unknown file to
load into arraylist " + file.getName());
                    }
                    objInput.close();
                        System.out.println("The progress of the
programs is loaded from the database.");
                } catch (EOFException e) {
                    // First time using this system
                            System.out.println("The database is
empty.");
                     System.out.println("Progress will be stored
in database.");
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }

    public void storeArrayList() {
        File[] files = new File("database").listFiles();

        for (File file : files) {
            if (file.exists()) { //If the files exists, load all
data into driver program
                try {
                                FileOutputStream fileOutput = new
FileOutputStream("database/" + file.getName());
                                ObjectOutputStream objOutput = new
ObjectOutputStream(fileOutput);
                    switch (file.getName()) {
                        case "order.txt" ->
```

```
                            objOutput.writeObject(orderList);
                        case "menu.txt" ->
                            objOutput.writeObject(menuList);
                        case "pax.txt" ->
                            objOutput.writeObject(paxList);
                        default ->
                                System.out.println("Unknown file
stored in database to be serialized in " + file.getName());
                    }
                    objOutput.close();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
               System.out.println("Progress  has  been  stored  in
database.");
    }

    public int addNewOrder() {
        int id;
        boolean validId;
        Order newEntry;

         System.out.println("\n\n<-- ORDERING FOR CATERING MEAL
SERVICE -->");
        do {
                        System.out.println("\nSuggested  id:  "  +
(orderList.getEntry(orderList.getNumberOfEntries()).getOrderId()
+ 1));
            System.out.print("\nEnter order ID: ");
            id = input.nextInt();
            input.nextLine();   // clear buffer
            newEntry = orderList.getEntry(new Order(id));

            validId = true;
            if (newEntry == null) {
                orderList.add(new Order(id));
            } else {
                System.out.println("Order ID exists!");
                validId = false;
            }
        } while (validId == false);

        // Input event details
        setEventDate(id);
        setEventTime(id);
        setEventAddress(id);
```

```
        return id;
    }

    public void addNewMenu() {
        int id;
        Menu newEntry;

        System.out.println("\n\n<-- ADDING NEW MENU -->");
        do {
                        System.out.println("\nSuggested  id:  "  +
(menuList.getEntry(menuList.getNumberOfEntries()).getMenuId()  +
1));
            System.out.print("\nEnter menu ID: ");
            id = input.nextInt();
            input.nextLine();   // clear buffer
            newEntry = menuList.getEntry(new Menu(id));

            if (newEntry != null) {
                System.out.println("Menu ID exists!");
            }
        } while (newEntry != null);

        boolean validName;
        String menuType;
        do {
            System.out.print("\nMenu Name: ");
            menuType = input.nextLine();

            validName = true;
             for (int i = 1; i <= menuList.getNumberOfEntries();
i++) {
                                                                  if
(menuType.toUpperCase().equals(menuList.getEntry(i).getMenuType(
).toUpperCase())) {
                    System.out.println("Menu Name exists!");
                    validName = false;
                    break;
                }
            }
        } while (validName == false);

        double price;
        boolean priceValid;
        do {
            priceValid = true;
            System.out.print("\nPrice per pax: ");
            price = input.nextDouble();
```

```
            input.nextLine();   // clear buffer

            if (price < 0) {
                System.out.println("Invalid price!");
                priceValid = false;
            }
        } while (priceValid == false);

        // adding new package info into menu list
        menuList.add(new Menu(id, menuType, price));
        System.out.println("\nMenu added successfully!");
    }

    public void addMenuToOrder(int orderId) {
        Menu menu = null;
        int menuId;

        displayMenu();

        do {
            System.out.print("Enter Menu ID: ");
            menuId = input.nextInt();
            input.nextLine();   // clear buffer

            // to search if menu exists in Menu List
            menu = menuList.getEntry(new Menu(menuId));

            if (menu == null) {
                System.out.println("Invalid menu!\n");
            }
        } while (menu == null);

        // assign the menu to a customer order
                                          orderList.getEntry(new
Order(orderId)).addMenuIntoOrder(menu);

        addPaxToOrderedMenu(orderId, menuId);
    }

    public void removeMenuFromOrder(int orderId) {
           System.out.println("\n<-- REMOVING a MENU from ORDER
-->");
          System.out.print("Enter Menu ID to be removed from the
order: ");
        int menuId = input.nextInt();
        input.nextLine();   // clear buffer

        Menu menu = menuList.getEntry(new Menu(menuId));
```

```
        if (menu != null) {
                                            orderList.getEntry(new
Order(orderId)).removeMenuFromOrder(menu);
            removePaxFromOrderedMenu(orderId, menuId);
              System.out.println("Menu [" + menuId + "] removed
from order.");
        } else {
            System.out.println("Menu not found in order!");
        }
    }

    public void editOrder() {
        boolean validChoice;
        do {
            validChoice = true;

            System.out.println("\n\n<-- EDITING an ORDER -->");
            System.out.println("[1] Edit event date");
            System.out.println("[2] Edit event time");
            System.out.println("[3] Edit event address");
            System.out.println("[4] Edit pax");
            System.out.println("[5] Add menu to order");
            System.out.println("[6] Remove menu to order");
            System.out.println("[0] Exit");

            System.out.print("\nSelect an option: ");
            int choice = input.nextInt();
            input.nextLine();   // clear buffer

            // check if order exists
            if (choice >= 1 && choice <= 6) {
                    System.out.print("\nEnter the order ID to be
edited: ");
                int orderId = input.nextInt();
                input.nextLine();   // clear buffer

                            Order order = orderList.getEntry(new
Order(orderId));

                if (order != null) {
                    displayOneOrder(order);
                    System.out.println();

                    switch (choice) {
                        case 1:
                                System.out.println("\n<-- Editing
EVENT DATE -->");
                            setEventDate(orderId);
```

```
                            break;
                        case 2:
                               System.out.println("\n<-- Editing
EVENT TIME -->");
                            setEventTime(orderId);
                            break;
                        case 3:
                               System.out.println("\n<-- Editing
EVENT ADDRESS -->");
                            setEventAddress(orderId);
                            break;
                        case 4:
                               System.out.println("\n<-- Editing
PAX -->");
                                System.out.print("Enter menu ID to
change pax: ");
                            int menuId = input.nextInt();
                            input.nextLine();   // clear buffer

                               Menu menu = menuList.getEntry(new
Menu(menuId));
                            if (menu != null) {

removePaxFromOrderedMenu(orderId, menuId);
                                    addPaxToOrderedMenu(orderId,
menuId);
                                  System.out.println("Pax of menu
[" + menuId + "] in order [" + orderId + "] is updated.");
                            } else {
                                    System.out.println("Menu not
found in order!");
                            }
                            break;
                        case 5:
                                System.out.println("\n<-- Adding
MENU to ORDER -->");
                            addMenuToOrder(orderId);
                            break;
                        case 6:
                            removeMenuFromOrder(orderId);
                            break;
                    }
                } else {
                    System.out.println("Order not found!");
                }
            } else if (choice != 0) {
                System.out.println("Invalid choice!");
                validChoice = false;
```

```
            }
        } while (validChoice == false);
    }

    public void editMenu() {
        boolean validChoice;
        do {
            validChoice = true;

                System.out.println("\n\n<-- EDITING a Menu item
-->");
            System.out.println("[1] Edit menu name");
            System.out.println("[2] Edit price of package");
            System.out.println("[0] Exit");

            System.out.print("\nSelect an option: ");
            int choice = input.nextInt();
            input.nextLine();   // clear buffer

            // check if order exists
            if (choice >= 1 && choice <= 2) {
                    System.out.print("\nEnter the menu ID to be
edited: ");
                int menuId = input.nextInt();
                input.nextLine();   // clear buffer

                Menu menu = menuList.getEntry(new Menu(menuId));

                if (menu != null) {
                    displayOneMenu(menu);
                    System.out.println();

                    switch (choice) {
                        case 1:
                            boolean validName;
                            String menuType;

                              System.out.println("< Editing MENU
NAME >");
                            do {
                                    System.out.print("\nEnter new
menu name: ");
                                menuType = input.nextLine();

                                validName = true;
                                          for (int i = 1; i <=
menuList.getNumberOfEntries(); i++) {
```

```
                                                                    if
(menuType.toUpperCase().equals(menuList.getEntry(i).getMenuType(
).toUpperCase())) {
                                            System.out.println("Menu
Name exists!");
                                            validName = false;
                                            break;
                                        }
                                    }
                                } while (validName == false);

                                            menuList.getEntry(new
Menu(menuId)).setMenuType(menuType);
                                break;
                            case 2:
                                  System.out.println("< Editing MENU
PRICE >");
                                System.out.print("\nEnter new price:
");
                                double price = input.nextDouble();
                                input.nextLine();   // clear buffer

                                            menuList.getEntry(new
Menu(menuId)).setPrice(price);
                                break;
                        }
                    } else {
                        System.out.println("Menu item not found!");
                    }
                } else if (choice != 0) {
                    System.out.println("Invalid choice!");
                    validChoice = false;
                }
            } while (validChoice == false);
        }

        public void searchOrder() {
                 System.out.println("\n\n<-- SEARCHING for an Order
-->");
            System.out.print("Enter Order ID: ");
            int id = input.nextInt();
            input.nextLine();   // clear buffer

            Order order = orderList.getEntry(new Order(id));
            if (order != null) {
                displayOneOrder(order);
            } else {
                System.out.println("Order not found!");
```

```
        }
    }

    public void searchMenu() {
           System.out.println("\n\n<-- SEARCHING for an item in
Menu -->");
        System.out.print("Enter Menu ID: ");
        int id = input.nextInt();
        input.nextLine();   // clear buffer

        Menu menu = menuList.getEntry(new Menu(id));
        if (menu != null) {
            displayOneMenu(menu);
        } else {
            System.out.println("Menu item not found!");
        }
    }

    public void displayOneOrder(Order order) {
        System.out.println("\n" + order);
        System.out.println("Menu packages\t: \n");
             System.out.println("MenuID\tPrice per pax\tPackage
Name\t\tPax\tSubtotal");

System.out.println("------\t-------------\t-------------------\t
----\t-----------");
                              for (int j = 1; j <=
order.getAllMenu().getNumberOfEntries(); j++) {
                                      System.out.printf("%-38s",
order.getAllMenu().getEntry(j));
              System.out.println(order.getAllPax().getEntry(j) +
"\tRM"
                                                              +
dp2.format(calcSubtotal(order.getOrderId(),

order.getAllMenu().getEntry(j).getMenuId())));
        }
    }

    public void displayOneMenu(Menu menu) {
            System.out.println("\nMenuID\tPrice per pax\tPackage
Name");

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

    public void removeOrder() {
```

```
        System.out.println("\n\n<-- REMOVING an order -->");
        System.out.print("Enter Order ID: ");
        int id = input.nextInt();
        input.nextLine();   // clear buffer

        Order anOrder = orderList.getEntry(new Order(id));

        if (anOrder != null) {
            orderList.remove(anOrder);
            System.out.println("Order successfully removed!");
        } else {
            System.out.println("Order not found!");
        }
    }

    public void removeMenu() {
        System.out.println("\n\n<-- REMOVING a menu item -->");
        System.out.print("Enter Menu ID: ");
        int id = input.nextInt();
        input.nextLine();   // clear buffer

        Menu menuItem = menuList.getEntry(new Menu(id));

        if (menuItem != null) {
            menuList.remove(menuItem);
                    System.out.println("Menu  Item  successfully
removed!");
        } else {
            System.out.println("Menu Item not found!");
        }
    }

    public void displayOrder() {
        if (orderList.getNumberOfEntries() != 0) {
            System.out.print("\n\n<-- ORDER LIST -->");
            for (int i = 1; i <= orderList.getNumberOfEntries();
i++) {
                                        System.out.println("\n\n"  +
orderList.getEntry(i));
                System.out.println("Menu packages\t: \n");
                            System.out.println("MenuID\tPrice  per
pax\tPackage Name\t\tPax\tSubtotal");

System.out.println("------\t-------------\t------------------\t
----\t-----------");
                                        for  (int  j  =  1;  j  <=
orderList.getEntry(i).getAllMenu().getNumberOfEntries(); j++) {
```

```
                                        System.out.printf("%-38s",
orderList.getEntry(i).getAllMenu().getEntry(j));

System.out.println(orderList.getEntry(i).getAllPax().getEntry(j)
+ "\tRM"
                                                                +
dp2.format(calcSubtotal(orderList.getEntry(i).getOrderId(),

orderList.getEntry(i).getAllMenu().getEntry(j).getMenuId())));
                }
            }
        } else {
            System.out.println("\n< No order in list! >");
        }
    }

    public void displayMenu() {
        if (menuList.getNumberOfEntries() != 0) {
            System.out.println("\n<-- MENU LIST -->");
              System.out.println("MenuID\tPrice per pax\tPackage
Name");

System.out.println("------\t-------------\t----------------");
            System.out.println(menuList);
        } else {
            System.out.println("\n< No menu in list! >");
        }
    }

    public void setEventDate(int orderId) {
        boolean validDate;
        int day, month, year;

        do {
            System.out.print("Enter event date(dd MM yyyy): ");
            day = input.nextInt();
            month = input.nextInt();
            year = input.nextInt();
            input.nextLine();   // clear buffer

            // validation for date input
            validDate = dateValidation(day, month, year);
        } while (validDate == false);

        orderList.getEntry(new Order(orderId)).setEventDate(day,
month, year);
    }
```

```
    public void setEventTime(int orderId) {
        boolean validTime;
        int startHour, startMin, endHour, endMin;

        do {
             System.out.println("\nEnter event time in 24 hours
format (HH MM).");
            System.out.print("Start time: ");
            startHour = input.nextInt();
            startMin = input.nextInt();
            input.nextLine();   // clear buffer
            System.out.print("End time: ");
            endHour = input.nextInt();
            endMin = input.nextInt();
            input.nextLine();   // clear buffer

            // validation for time input
                validTime = timeValidation(startHour, startMin,
endHour, endMin);
        } while (validTime == false);

                                            orderList.getEntry(new
Order(orderId)).setEventStartTime(startHour, startMin);
                                            orderList.getEntry(new
Order(orderId)).setEventEndTime(endHour, endMin);
    }

    public void setEventAddress(int orderId) {
        System.out.print("\nEnter event address: ");
        String address = input.nextLine();

                                            orderList.getEntry(new
Order(orderId)).setEventAddress(address);
    }

    public void addPaxToOrderedMenu(int orderId, int menuId) {
        int pax;
        do {
            System.out.print("\nEnter pax: ");
            pax = input.nextInt();
            input.nextLine();   // clear buffer

            if (pax < 0) {
                System.out.println("Invalid pax!");
            } else if (pax < 20) {
                 System.out.println("The minimum pax requirement
is 20.");
            }
```

```
        } while (pax < 20);

        int paxId = paxList.getNumberOfEntries() + 1;
        paxList.add(new Pax(paxId, pax, orderId, menuId));

        Pax paxItem = paxList.getEntry(new Pax(paxId));
                                            orderList.getEntry(new
Order(orderId)).addPaxToMenu(paxItem);
    }

       public void removePaxFromOrderedMenu(int orderId, int
menuId) {
              Pax paxItem = paxList.getEntry(new Pax(orderId,
menuId));
                                            orderList.getEntry(new
Order(orderId)).removePaxFromMenu(paxItem);
    }

    public double calcSubtotal(int orderId, int menuId) {
                int pax = paxList.getEntry(new Pax(orderId,
menuId)).getPax();
                    double price = menuList.getEntry(new
Menu(menuId)).getPrice();
        double result = price * (double) pax;

        return result;
    }

    public boolean dateValidation(int day, int month, int year)
{
        boolean validDate = true;
        if (year >= 2020 && year <= 2030) {
            switch (month) {
                case 1:
                case 3:
                case 5:
                case 7:
                case 8:
                case 10:
                case 12:
                    if (day < 1 || day > 31) {
                        System.out.println("Invalid day!");
                        validDate = false;
                    }
                    break;
                case 2:
                    if (day < 1 || day > 29) {
                        System.out.println("Invalid day!");
```

```
                        validDate = false;
                    } else if (day == 29) {
                        if (year % 4 != 0) {
                            System.out.println("Invalid day!");
                            validDate = false;
                        }
                    }
                    break;
                case 4:
                case 6:
                case 9:
                case 11:
                    if (day < 1 || day > 30) {
                        System.out.println("Invalid day!");
                        validDate = false;
                    }
                    break;
                default:
                    System.out.println("Invalid month!");
                    validDate = false;
            }
        } else {
            System.out.println("Invalid year!");
            validDate = false;
        }
        return validDate;
    }

     public boolean timeValidation(int startHour, int startMin,
int endHour, int endMin) {
        boolean validTime = true;
         if (startHour < 24 && endHour < 24 && startMin < 60 &&
endMin < 60
                && startHour >= 0 && endHour >= 0 && startMin >=
0 && endMin >= 0) {
            if (startHour == endHour) {
                if (startMin > endMin) {
                    System.out.println("Invalid time!");
                    validTime = false;
                }
            } else if (startHour > endHour) {
                System.out.println("Invalid time!");
                validTime = false;
            }
        } else {
            System.out.println("Invalid time!");
            validTime = false;
        }
```

```
        return validTime;
    }
}
```

5.2 Client class - OrderDriver

```
import java.util.Scanner;
import java.util.InputMismatchException;

/** OrderDriver - A main client class for user interface.
  * @author Low Zhi Yi
  */

public class OrderDriver {

    private OrderManager manageOrder = new OrderManager();
    private Scanner input = new Scanner(System.in);

    public static void main(String[] args) {
        new OrderDriver().runApp();
    }

    public void runApp() {
        int choice = 0;
        do {
            // display choices
            System.out.println("\n< Booking for Catering Service
>\n");
            System.out.println("\t-- ORDER --");
                  System.out.println(" [1] Add a new order");
// Add a new order & assign menu + pax to order
                      System.out.println(" [2] Edit order");
// Edit order details
                System.out.println(" [3] Search for an order");
// Search and display order details by order id
                 System.out.println(" [4] Display all orders");
// Display all orders with details
                     System.out.println(" [5] Remove order");
// Remove an order record including all details
            System.out.println("\n\t-- MENU --");
                       System.out.println(" [6] Add menu");
// Add a new menu
                      System.out.println(" [7] Edit menu");
// Edit menu details
             System.out.println(" [8] Search for a menu item");
// Search and display menu details by menu id
                  System.out.println(" [9] Display menu list");
// Display all menu
```

```
                System.out.println("[10] Remove a menu item");
// Remove a menu item including all details
                    System.out.println("\n [0] End system");
// Exit

            // get user's choice
            try {
                System.out.print("\nSelect an option: ");
                choice = input.nextInt();
                input.nextLine();   // clear buffer
            } catch (InputMismatchException e) {
                choice = -1;
                input.nextLine();   // clear buffer
            }

            // call functions based on user's choice
            switch (choice) {
                case 1:
                    // Create order and return the id
                    int orderId = manageOrder.addNewOrder();
                    promptEnterKey();

                    // Assign menu(s) to order
                    char moreMenu;
                    do {
                        manageOrder.addMenuToOrder(orderId);

                          System.out.println("Do you want to add
more menu?");
                         System.out.print("Enter 'Y' to add more
menu: ");
                        moreMenu = input.nextLine().charAt(0);
                        } while (moreMenu == 'Y' || moreMenu ==
'y');

                                System.out.println("\nOrder added
successfully!");
                    break;
                case 2:
                    manageOrder.editOrder();
                    promptEnterKey();
                    break;
                case 3:
                    manageOrder.searchOrder();
                    promptEnterKey();
                    break;
                case 4:
                    manageOrder.displayOrder();
```

```
                    promptEnterKey();
                    break;
                case 5:
                    manageOrder.displayOrder();
                    manageOrder.removeOrder();
                    promptEnterKey();
                    break;
                case 6:
                    manageOrder.addNewMenu();
                    promptEnterKey();
                    break;
                case 7:
                    manageOrder.editMenu();
                    promptEnterKey();
                    break;
                case 8:
                    manageOrder.searchMenu();
                    promptEnterKey();
                    break;
                case 9:
                    manageOrder.displayMenu();
                    promptEnterKey();
                    break;
                case 10:
                    manageOrder.displayMenu();
                    manageOrder.removeMenu();
                    promptEnterKey();
                    break;
                case 0:
                    manageOrder.storeArrayList();
                    System.out.println("-- EXIT SYSTEM --");
                    break;
                default:
                    System.out.println("Invalid choice!\n");
            }
        } while (choice != 0);
    }

     // Method to read a random string from user to proceed the
program
    public static void promptEnterKey() {
        Scanner enterKey = new Scanner(System.in);
        System.out.println();
        System.out.print("Press \"ENTER\" to continue...");
        enterKey.nextLine();
        System.out.println();
    }
}
```