BACS2063 Data Structures and Algorithms

ASSIGNMENT 202205

Student Name : Lee Jing Jet

Student ID : 2209855

Programme : Data Structures and Algorithms

Tutorial Group : G7

Assignment Title : Catering (meal) Services

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.

Jet

Student's signature

11/9/2022

Date

Table of Contents

1. Introduction **3**

2. Abstract Data Type (ADT) Specification **3**

3. ADT Implementation **3**
3.1 Overview of ADT 3
3.2 ADT Implementation 3

4. Entity Classes **3**
4.1 Entity Class Diagram 3
4.2 Entity Class Implementation 3

5. Client Program **3**

1. Introduction

I chose Catering (meal) Services as my application. The adt I chose to use was Doubly Linked List to create a system of catering services. This system will have two entities, one is Booking and the other is MenuEntity. This system can have the ability to let users handle menus and make bookings. According to the needs of the times, applications are gradually needed. Because of market demand, business is gradually becoming online, so we chose to create a catering online system, so that the catering industry can not only accept business offline, but also accept customers online to choose services.

2. Abstract Data Type (ADT) Specification

// Write the complete ADT specification for ONE (1) collection ADT that you selected. Remember to follow the ADT specification format.

ADT DoublyLinkedList

A doubly linked list is a linked data structure made up of nodes, or groups of entries that are progressively connected. A different type of linked list is a doubly linked list. In contrast to a simply linked list, you may move ahead and backward with ease in two different directions. The terms listed below are crucial for comprehending the idea of doubly linked lists. Either the end of the list or the location you specify will get an addition. The list's index begins at 0.

void clear() Description : the list's whole elements are removed. Postcondition : The list is currently empty.
boolean add(T newElement) Description : Add newElement into the end of the list Postcondition : Add newElement into the end of the list Returns : if newElement was successfully added into the list will return True. Or not will return False
boolean add(int index, T newElement) Description : The doubly linked list should now include newElement. The newElement will be inserted into the doubly linked list at the provided position and index. Precondition : index must fall between 0 and the sum of the list's entries, minus 1. Postcondition : add newElement into the specified index of the list. Returns : if newElement was successfully added into the list will return True. Or not will return False
boolean addAll(T… newElements) Description : add all newElements into the end of the list. Postcondition : add newElement into the doubly linked list at the end of the doubly linked

list.

Returns : if newElement was successfully added into the list will return True. Or not will return False

boolean contains(T element)
Description :determines if an element is present in the list.

Postcondition : The list is still the same.

Returns : if an element contains any entry in the list it will return True. Or not will return false.

T get(int index)
Description : Obtain the index entry. The first entry is represented by 0, the second by 1, and so on.

Precondition : index must fall between 0 and the sum of the list's entries, minus 1.

Postcondition : The list is still the same.

Returns : the item that is shown at position (index + 1)

int indexOf(T element)
Description : Obtain the element's index.

Postcondition : The list has not altered.

Returns : wil return integer as the element’s index

boolean isEmpty()
Description : checks to see whether the list is empty

Postcondition : The list has not altered

Returns : If the list is empty return True or not
return false.

boolean remove(T element)
Description : Take the specified element out of the list.

Postcondition : The list entry that included the element has been deleted.

Returns : if the entry is successfully removed return True or not return false.

boolean remove(int index)

Description : Delete the list item with the specified index.

Precondition : index must fall between 0 and the sum of the list's entries, minus 1.

Postcondition : The listing's entry at position index has been deleted.

Returns : if the entry is successfully removed return True or not return false.

boolean removeAll(T elements)

Description : Eliminate all the items listed there. The elements may have only one entry or several entries.

Postcondition : The listing's entry at givenPosition has been eliminated.

Returns : if the entry is successfully removed return True or not return false.

boolean set(int index, T newElement

Description : With newElement, replace the entry at the specified index.

Precondition : index must fall between 0 and the sum of the list's entries, minus 1.

Postcondition : NewElement has successfully replaced the entry.

Returns : if the entry is replaced by newElement successfully return True or not return false.

int sizeOf()

Description : Check the list's size by number.

Postcondition : The list has not altered.

Returns : The number of entries in the list is represented by an integer.

ListInterface where(WhereClause<T> list)

Description : Get every entry in a certain list; every entry corresponds to the list's detailed information. Only one or more entries are permitted.

Postcondition : The list has not altered.

Returns : Each entry that contains the same information in list void

void orderBy(OrderClause<T> list)

Description : may arrange the list by any data column. Both ascending and descending orders are possible.

Postcondition : The list has been arranged by the specified data field in either ascending or descending order.

T firstOrDefault(FirstOrDefaultClause<T> list) Description : Obtain a list entry that has the same thorough information as the list. Only the first match in the list or one can exist. Postcondition : The list has not altered. Returns : The item that shares all of the details with the list
Iterator<T> getIterator() **boolean hasNext()** Description : Identify the iteration's completion Postcondition : if there is further structure to see, returns true: Likewise, if value (next) may yield a valuable value Returns : if the iterator has more elements to be considered return True or not return false.
T next() Description : Identify the iteration's completion Precondition : traversal contains more components.. Postcondition : returns the most recent value and advances the iterator. Returns: The value as it is right now, without increment.

3. ADT Implementation

3.1 Overview of ADT

The collection ADTs that i used in my client classes is doubly linked list. I will use this collection ADTs because ss opposed to a single linked list, it is simpler to implement. Even while the code for the doubly linked approach is slightly longer than for the singly linked version, it tends to be a little more "obvious" in its goal, making it simpler to implement and debug.The implementation of a doubly linked list takes use of a first node. We round up the list by adding a last node. The last node is comparable to the first node in that it is a valueless node that exists continuously. The initialization of the doubly linked list creates the first and end nodes. Both the head and tail of the data member link to the same node. By removing any requirement for special-case code when the list is empty or when we insert at the start or end of the list, these nodes serve the objective of simplifying the insert, edit, and delete functions.

3.2 ADT Implementation

3.2.1 Methods As Defined in Interface

Clear whole elements

```
public void clear() {
      firstNode = lastNode = null;    // set first and last node to empty
      num = 0;                        // set number of size to 0
   }
```

Add new element

```
public boolean add(T newElement) {
      if (newElement != null) {
         Node newNode = new Node(newElement);

         if (isEmpty()) {
            firstNode = newNode;        // set new node if empty
            lastNode = newNode;
         } else {
            newNode.prev = lastNode;    // arrange the node
            lastNode.next = newNode;
            lastNode = newNode;
         }

         num++;
      }
      return false;
   }
```

Check the element is inside the list or not

```
public boolean contains(T element) {
      if (element != null) {
         return travel(element) != null;
      }
      return false;
   }
```

To get the index entry

```
public T get(int index) {
      T data = null;
      if (inRange(index)) {
         Node nodeCurrent = travel(index);
         data = nodeCurrent.data;
      }
      return data;
   }
```

To get the element index

```
public int indexOf(T element) {
        if (element != null) {
            int index = 0;
            for (Node nodeCurrent = firstNode; nodeCurrent != null &&
                    inRange(index); index++, nodeCurrent = nodeCurrent.next) {
                if (nodeCurrent.data.equals(element)) {
                    return index;
                }
            }
        }
        return -1;
    }
```

To check the list is empty or not

```
public boolean isEmpty() {
        return num == 0;
    }
```

To remove the element

```
public boolean remove(T element) {
        if (element == null || isEmpty()) {
            return false;
        } else {
            Node nodeCurrent = travel(element);
            if (nodeCurrent != null) {
                remove(nodeCurrent);
                return true;
            }
            return false;
        }
    }
```

To replace the entry at the specified index

```
 public boolean set(int index, T newElement) {
        if (isEmpty() || !inRange(index) || newElement == null) {
            return false;
        } else {
            Node nodeCurrent = travel(index);
            nodeCurrent.data = newElement;
            return true;
        }
    }
```

To check the list size

```
public int sizeOf() {
        return num;
    }
```

To get every entry in a certain list

```
public ListInterface where(WhereClause<T> list) {
        ListInterface<T> linkedList = new DoublyLinkedList<>();

        for (Node nodeCurrent = firstNode; nodeCurrent != null; nodeCurrent =
nodeCurrent.next) {
            if (list.match(nodeCurrent.data)) {
                linkedList.add(nodeCurrent.data);
            }
        }
        return linkedList;
    }
```

To arrange the list by any data column

```
public void orderBy(OrderClause<T> list) {
        int endIndex = num - 1;
        // Return true if bubble sort pass has changed
        // Return false if end index reduced by 1 and continue until next sorting
        while (bubbleSort(endIndex--, list)) {
        }
    }
```

To obtain a list entry that has the same thorough information as the list.

```
public T firstOrDefault(FirstOrDefaultClause<T> list) {
        T data = null;
        boolean found = false;
        for (Node nodeCurrent = firstNode; nodeCurrent != null && !found; nodeCurrent =
nodeCurrent.next) {
            if (list.match(nodeCurrent.data)) {
                data = nodeCurrent.data;

                found = true;
            }
        }
        return data;
    }
```

3.2.2 Utility Methods (makeroom, removegap, etc)

(Add) Add element into doubly linked list at the provided position and index

```
public boolean add(int index, T newElement) {
        if (newElement == null || !inAddRange(index)) {
            return false;
        } else {
            Node newNode = new Node(newElement);
            if (index == 0) {
```

```
            if (isEmpty()) {
                add(newElement);
                return true;        // Return because add function will num++
            } else {
                newNode.next = firstNode;
                firstNode.prev = newNode;
                firstNode = newNode;
            }
        } else if (index == num) {
            lastNode.next = newNode;
            newNode.prev = lastNode;
            lastNode = newNode;
        } else {
            Node nodeCurrent = travel(index);
            nodeCurrent.prev.next = newNode;
            newNode.prev = nodeCurrent.prev;
            newNode.next = nodeCurrent;
            nodeCurrent.prev = newNode;
        }
        num++;
        return true;
    }
  }
```

(Add) Add all element into the end of the list

```
public boolean addAll(T... newElements) {
    if (newElements != null) {
        if (isElementsValid(newElements)) {
            for (T element : newElements) {
                add(element);
            }
            return true;
        }
    }
    return false;
  }
```

To the remove the specified index element

```
public boolean remove(int index) {
    if (isEmpty() || !inRange(index)) {
        return false;
    } else {
        remove(travel(index));
        return true;
    }
  }
```

To remove all the element in the list

```
public boolean removeAll(T... elements) {
```

```
        if (isEmpty() || !isElementsValid(elements)) {
            return false;
        } else {
            for (T element : elements) {
                remove(element);
            }
            return true;
        }
    }
```

3.2.3 Overriden Java Standard Methods (iterator, tostring)

To Identify the iteration's completion. The method override because the derived class wants to give its own implementation it can give by overriding the method of the parent class.

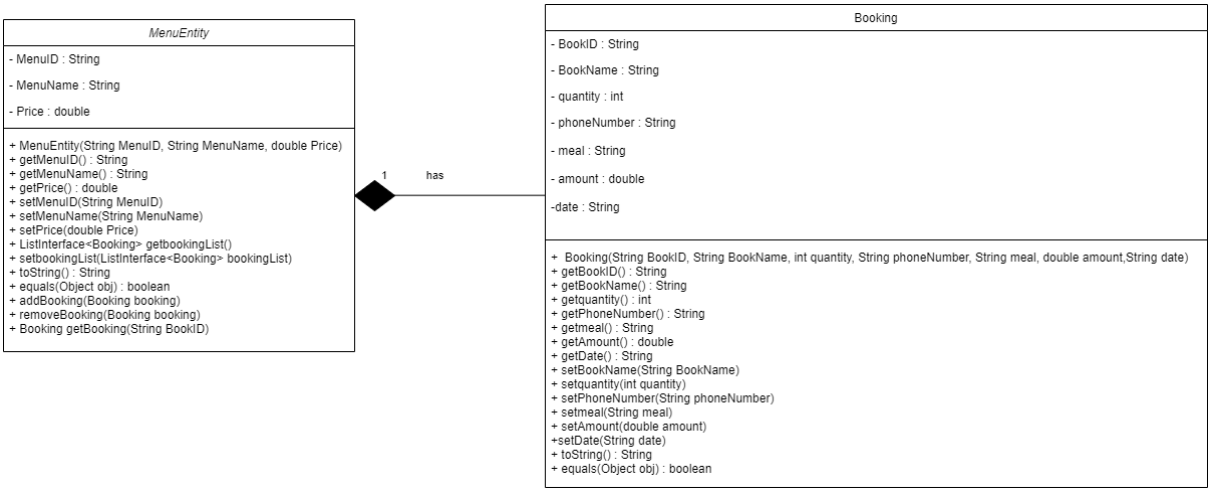
```
public Iterator<T> getIterator() {
        return new DoublyLinkListIterator();
    }
```

It is because this method was getting automatically called when the print statement is written. So this method is overridden in order to return the values of the object

```
public String toString() {
        String str = "";
        for (Node nodeCurrent = firstNode; nodeCurrent != null; nodeCurrent = nodeCurrent.next)
{
            str += nodeCurrent.data + "\n";
        }
        return str;
    }
```

4. Entity Classes

4.1 Entity Class Diagram



4.2 Entity Class Implementation

MenuEntity

```
package client;

import adt.DoublyLinkedList;
import adt.ListInterface;
import adt.OrderClause;
import entity.MenuEntity;
import utility.DesignConsole;
import utility.ScannerHandler;
import java.util.Iterator;
import static utility.DesignConsole.BLUE;
import static utility.DesignConsole.PURPLE;

/**
 *
 * @author Lee Jing Jet
 */
public class MealsMenu {

    private ListInterface<MenuEntity> MenuList;
    private final ScannerHandler scanner = new ScannerHandler();
    private final DesignConsole print = new DesignConsole();

    private int menuIndex = 1004;
    private static final int TABLE_WIDTH = 40, LIST_TABLE_WIDTH = 60,
SPECIFY_TABLE_WIDTH = 50;

    public MealsMenu(ListInterface<MenuEntity> MenuList) {
        this.MenuList = MenuList;
    }

    public void main() {
        int choice;

        do {
            print.tableHeader("Meals Menu", TABLE_WIDTH);
            System.out.println(BLUE + "| 1. View Menu                    |");
            System.out.println(BLUE + "| 2. Add Menu                     |");
            System.out.println(BLUE + "| 3. Remove Menu                  |");
```

```
            System.out.println(BLUE + "| 4. Edit Menu                    |");
            System.out.println(BLUE + "| 5. Search Menu                  |");
            System.out.println(BLUE + "| 6. Sort Menu By                 |");
            System.out.println(BLUE + "| 7. Exit                       |");
            System.out.println(PURPLE +
"=======================================");

            choice = scanner.nextInt(BLUE + "Enter Your Choice: ", "Please enter a number
between 1 to 7", 1, 7);

            switch (choice) {
                case 1:
                    showMenuRecord();
                    break;
                case 2:
                    addMeal();
                    break;
                case 3:
                    removeMeal();
                    break;
                case 4:
                    editMenu();
                    break;
                case 5:
                    searchMenu();
                    break;
                case 6:
                    sortByList();
                    break;
                default: {
                }
                break;
            }

        } while (choice != 7);
    }

    public void addMeal() {
        print.tableHeader("ADD MEAL", TABLE_WIDTH);
        print.otherMsg("New MENU MEAL", 1);

        // generate menu id
        String MenuID = "M" + String.format("%4d", menuIndex++);
        System.out.println(BLUE +"Category ID         : " + MenuID);

        // input Menu Name
        String MenuName = scanner.nextLine(BLUE + "Enter Menu Name : ");

        // input price
        Double price = scanner.nextDouble(BLUE +"Enter Price: ");


        // Show the details
        print.tableHeader("New Category Details", SPECIFY_TABLE_WIDTH);
        System.out.printf(BLUE +"| %-18s | %-25s |\n", "Menu ID", MenuID);
```

```
        System.out.printf(BLUE +"| %-18s | %-25s |\n", "Menu Name", MenuName);
        System.out.printf(BLUE +"| %-18s | %-25.2f |\n", "Price", price);
        print.tableFooter(SPECIFY_TABLE_WIDTH);

        // Confirmation
        print.otherMsg("Please make sure that all the details given above is correct", 1);
        print.hint("The above details can be edited in the future");

        if (scanner.confirmation("Do You Sure You Want To Add This Meals (Y = yes / N = no)?
>> ")) {
            MenuList.add(new MenuEntity(MenuID, MenuName, price));
            print.success("The Meal Is Successful Added");
        } else {
            print.cancelled("Your Add Request Has Been Cancelled");
        }
    }

    public void removeMeal() {
        // Check whether is empty
        if (MenuList.isEmpty()) {
            print.failed("No Meal Record!");
        } else {
            // Show all meal record
            print.tableHeader("REMOVE MENU", LIST_TABLE_WIDTH);
            displayMenu(MenuList);
            print.tableFooter(LIST_TABLE_WIDTH);
            print.otherMsg(String.format("Total Number Of Meals: %d", MenuList.sizeOf()), 0);

            // Get the Menu id
            String MenuID = scanner.nextLine(BLUE +"\nEnter The Menu ID To Remove (e.g.
M1001): ");
            MenuEntity removeMenu = MenuList.firstOrDefault(d ->
d.getMenuID().equalsIgnoreCase(MenuID));

            // Compare the id with record
            if (removeMenu == null) {
                print.failed("Menu ID Not Found");
            } else {
                // Show user search result
                displaySearchMenu(BLUE+"Menu to remove", removeMenu);

                if (scanner.confirmation(BLUE+"Are You Sure You Want To Remove This Meal? (Y
= yes / N = no) >> ")) {
                    MenuList.remove(removeMenu);
                    print.success("The Menu You Choose Has Been Removed Successful");
                } else {
                    print.cancelled("Your Remove Request Has Been Cancelled");
                }
            }
        }
    }

    public void editMenu() {
        // Check whether is empty
        if (MenuList.isEmpty()) {
```

```
            print.failed("No Menu Record!");
        } else {
            // Show all Menu Record
            print.tableHeader("EDIT MENU", LIST_TABLE_WIDTH);
            displayMenu(MenuList);
            print.tableFooter(LIST_TABLE_WIDTH);
            print.otherMsg(String.format("Total Number Of Meal: %d", MenuList.sizeOf()), 0);

            // Input menu id
            String MenuID = scanner.nextLine(BLUE+"\nEnter The Menu ID To Edit (e.g.
M1001): ");
            MenuEntity editMenu = MenuList.firstOrDefault(m ->
m.getMenuID().equalsIgnoreCase(MenuID));

            // Get the category id
            if (editMenu == null) {
                print.failed("Menu ID Not Found");
            } else {
                displaySearchMenu(BLUE +"Menu to edit", editMenu);
                // Enter new meal's name
                String MenuName = scanner.nextLine("Enter New Menu Name      : ");
                double Price = scanner.nextDouble("Enter New Menu Price      : ");

                if (scanner.confirmation(BLUE+"Do Your Sure You Want To Edit This Menu? (Y =
yes / N = no) >> ")) {
                    // Set the details into the list
                    editMenu.setMenuName(MenuName);
                    editMenu.setPrice(Price);

                    print.success("Your Record Is Record Successful");
                } else {
                    print.cancelled("Your Edit Request Has Been Cancelled");
                }
            }
        }
    }

    public void searchMenu() {
        // Choose search by
        print.tableHeader("SEARCH MENU", TABLE_WIDTH);
        System.out.println(BLUE+"1. Search By Category ID");
        System.out.println(BLUE+"2. Search By Category Name");
        System.out.println(BLUE+"3. Back To Donation Category Main Page");

        int choice = scanner.nextInt(BLUE+"Enter Your Choice: ", "Please enter a number
between 1 to 3", 1, 3);

        switch (choice) {
            case 1:
                searchMenuId();
                break;
            case 2:
                searchMenuName();
                break;
            default: {
```

```
            }
            break;
        }
    }

    public void searchMenuId() {
        print.tableHeader("SEARCH MENU ID", TABLE_WIDTH);
        String MenuID = scanner.nextLine(BLUE+"Enter menu id to search (e.g. M1001): ");

        // Check id
        MenuEntity me = MenuList.firstOrDefault(m ->
m.getMenuID().equalsIgnoreCase(MenuID));

        // Show result
        if (me != null) {
            displaySearchMenu(BLUE+"Menu search result", me);
        } else {
            print.failed("Menu ID not found");
        }
    }

    public void searchMenuName() {
        print.tableHeader("SEARCH MENU NAME", TABLE_WIDTH);
        System.out.println(BLUE+"1. Search Name Starts With");
        System.out.println(BLUE+"2. Search Name Ends With");
        System.out.println(BLUE+"3. Search Name Contains");
        System.out.println(BLUE+"4. Back To Search Main Page");
        int choice = scanner.nextInt(BLUE+"Enter Your Choice: ", "Please enter a number
between 1 to 4", 1, 4);

        if (choice != 4) {
            String MenuName = scanner.nextLine(BLUE+"Enter name to search:
").toUpperCase();
            ListInterface<MenuEntity> mList = new DoublyLinkedList<>();

            // Get all menu which related with the input
            switch (choice) {
                case 1:
                    mList = MenuList.where(m ->
m.getMenuName().toUpperCase().startsWith(MenuName));
                    break;
                case 2:
                    mList = MenuList.where(m ->
m.getMenuName().toUpperCase().endsWith(MenuName));
                    break;
                case 3:
                    mList = MenuList.where(d ->
d.getMenuName().toUpperCase().contains(MenuName));
                    break;
                default:
                    break;
            }

            // Show result
            if (mList.isEmpty()) {
```

```
            print.failed("Menu name not found");
        } else {
            displaySearchMenu(mList);
        }
    }
}


public void sortByList() {
    print.tableHeader("SORT MENU LIST BY", TABLE_WIDTH);

    System.out.println(BLUE+"1. Sort By Menu ID");
    System.out.println(BLUE+"2. Sort By Menu Name");
    System.out.println(BLUE+"3. Back To Previous Page");

    int choice = scanner.nextInt(BLUE+"Enter Your Choice: ", "Please enter a number
between 1 to 4", 1, 4);

    if (choice != 3) {
        print.otherMsg("Sort Menu By", 1);
        System.out.println(BLUE+"1. Ascending Order");
        System.out.println(BLUE+"2. Descending Order");
        System.out.println(BLUE+"3. Back To Previous Page");

        int sequenceType = scanner.nextInt(BLUE+"Enter Your Choice: ", "Please enter a
number between 1 to 2", 1, 2);

        if (sequenceType != 3) {
            switch (choice) {
                case 1:
                    sortById(sequenceType);
                    break;
                case 2:
                    sortByName(sequenceType);
                    break;
                default: {
                }
                break;
            }
            print.success("Sort successfully, press 6 to view menu record");
        }
    }
}

public void sortById(int sequenceType) {
    switch (sequenceType) {
        case 1:
            MenuList.orderBy((m1, m2)
                    -> m1.getMenuID().compareToIgnoreCase(m2.getMenuID()) < 0
                    ? OrderClause.MOVE_FORWARD : OrderClause.MOVE_BACKWARD);
            break;
        case 2:
            MenuList.orderBy((m1, m2)
                    -> m1.getMenuID().compareToIgnoreCase(m2.getMenuID()) > 0
                    ? OrderClause.MOVE_FORWARD : OrderClause.MOVE_BACKWARD);
```

```
            break;
        default: {
        }
        break;
    }
}

public void sortByName(int sequenceType) {
    switch (sequenceType) {
        case 1:
            MenuList.orderBy((m1, m2)
                -> m1.getMenuName().compareToIgnoreCase(m2.getMenuName()) < 0
                ? OrderClause.MOVE_FORWARD : OrderClause.MOVE_BACKWARD);
            break;
        case 2:
            MenuList.orderBy((m1, m2)
                -> m1.getMenuName().compareToIgnoreCase(m2.getMenuName()) > 0
                ? OrderClause.MOVE_FORWARD : OrderClause.MOVE_BACKWARD);
            break;
        default: {
        }
        break;
    }
}


public void showMenuRecord() {
    print.tableHeader("MENU", LIST_TABLE_WIDTH);
    displayMenu(MenuList);
    print.tableFooter(LIST_TABLE_WIDTH);
    print.otherMsg(String.format("Total Number Of Menu: %d", MenuList.sizeOf()), 0);
}

public void displaySearchMenu(String title, MenuEntity menu) {
    print.tableHeader(title.toUpperCase(), SPECIFY_TABLE_WIDTH);
    System.out.printf(BLUE+"| %-18s | %-25s |\n", "Category ID", menu.getMenuID());
    System.out.printf(BLUE+"| %-18s | %-25s |\n", "Category Name",
menu.getMenuName());
    System.out.printf(BLUE+"| %-18s | %-25.2f |\n", "Price", menu.getPrice());
    print.tableFooter(SPECIFY_TABLE_WIDTH);
}

public void displaySearchMenu(ListInterface<MenuEntity> menu) {
    print.tableHeader("MENU SEARCH RESULT", LIST_TABLE_WIDTH);
    displayMenu(menu);
    print.tableFooter(LIST_TABLE_WIDTH);
}

public void displayMenu(ListInterface<MenuEntity> MenuList) {
    Iterator<MenuEntity> itr = MenuList.getIterator();

    System.out.printf(BLUE+"| %-5s | %-20s | %-25s |\n", "ID", "NAME", "PRICE");
    print.tableMiddleLine(LIST_TABLE_WIDTH);

    while (itr.hasNext()) {
```

```
            MenuEntity me = itr.next();
            System.out.println(me.toString());
        }
    }
}
```

Booking

```
package client;

import adt.DoublyLinkedList;
import adt.ListInterface;
import adt.OrderClause;
import entity.Booking;
import entity.MenuEntity;
import utility.ScannerHandler;
import utility.DesignConsole;
import java.util.Iterator;
import java.time.format.DateTimeFormatter;
import java.time.LocalDateTime;
import java.text.SimpleDateFormat;
import java.text.ParseException;
import static utility.DesignConsole.BLUE;
import static utility.DesignConsole.PURPLE;
import static utility.DesignConsole.RED;

/**
 *
 * @author Lee Jing Jet
 */
public class BookingMaintain {

    private ListInterface<Booking> bookList;
    private ListInterface<MenuEntity> MenuList;
    private final ScannerHandler scanner = new ScannerHandler();
    private final DesignConsole print = new DesignConsole();

    private int bookIDindex = 1011;
    private static final int TABLE_WIDTH = 40, LIST_TABLE_WIDTH = 97,
SPECIFY_TABLE_WIDTH = 45,
            MODIFY_REPORT_TABLE_WIDTH = 55, SUMMARY_TABLE_WIDTH = 42;

    private String addNewBook = "";
    private String editBook = "";
    private String removeBook = "";

    public BookingMaintain(ListInterface<Booking> bookList, ListInterface<MenuEntity>
MenuList) {
        this.bookList = bookList;
        this.MenuList = MenuList;

    }
```

```
    public void main() {
        int choice;

        do {
            print.tableHeader("Customer Booking", TABLE_WIDTH);
            System.out.println(BLUE + "| 1. Add Booking                    |");
            System.out.println(BLUE + "| 2. Remove Booking                 |");
            System.out.println(BLUE + "| 3. Edit Booking                   |");
            System.out.println(BLUE + "| 4. Search Booking                 |");
            System.out.println(BLUE + "| 5. Sort Booking By                |");
            System.out.println(BLUE + "| 6. Display Booking Details        |");
            System.out.println(BLUE + "| 7. Display Booking List By Meals  |");
            System.out.println(BLUE + "| 8. Display Reports                |");
            System.out.println(BLUE + "| 9. Exit                           |");
            System.out.println(PURPLE +
"=====================================");

            choice = scanner.nextInt(BLUE + "Please Enter Your Choice: ", "Please enter a
number between 1 to 9", 1, 9);

            switch (choice) {
                case 1:
                    addBooking();
                    break;
                case 2:
                    removeBooking();
                    break;
                case 3:
                    editBooking();
                    break;
                case 4:
                    searchBooking();
                    break;
                case 5:
                    sortBookingBy();
                    break;
                case 6:
                    allrecord();
                    break;
                case 7:
                    showBookingList();
                    break;
                case 8:
                    showReports();
                    break;
                default: {
                }
                break;
            }
        } while (choice != 9);
    }

    public void showReports() {
        print.tableHeader("REPORT LIST", TABLE_WIDTH);
        System.out.println(BLUE + "1. Modify Report                    |");
```

```
        System.out.println(BLUE + "2. Summary Report                    |");
        System.out.println(BLUE + "3. Back To Previous Page              |");
        print.tableFooter(TABLE_WIDTH);

        int choice = scanner.nextInt(BLUE + "Enter Your Choice: ", "Please enter a number
between 1 to 3", 1, 3);

        switch (choice) {
            case 1:
                ModifyReport();
                break;
            case 2:
                SummaryReport();
                break;
            default: {
            }
            break;

        }
    }

    public void ModifyReport() {
        print.tableHeader(" MODIFY REPORT", MODIFY_REPORT_TABLE_WIDTH);

        // what record have added today
        print.toCenter("Added Booking Today", MODIFY_REPORT_TABLE_WIDTH);
        print.tableMiddleLine(MODIFY_REPORT_TABLE_WIDTH);
        System.out.println(String.format(BLUE + "| %5s | %20s | %20s |", "ID", "NAME",
"MEAL"));
        print.tableMiddleLine(MODIFY_REPORT_TABLE_WIDTH);
        if ("".equals(addNewBook)) {
            print.toCenter("No Added Record Today", MODIFY_REPORT_TABLE_WIDTH);
            System.out.println();
        } else {
            System.out.println(addNewBook);
        }
        print.tableFooter(MODIFY_REPORT_TABLE_WIDTH);

        // what record have edited today
        print.toCenter("Edited Booking Today", MODIFY_REPORT_TABLE_WIDTH);
        print.tableMiddleLine(MODIFY_REPORT_TABLE_WIDTH);
        System.out.println(String.format(BLUE + "| %5s | %20s | %20s |", "ID", "NAME",
"MEAL"));
        print.tableMiddleLine(MODIFY_REPORT_TABLE_WIDTH);
        if ("".equals(editBook)) {
            print.toCenter("No Edited Record Today", MODIFY_REPORT_TABLE_WIDTH);
            System.out.println();
        } else {
            System.out.println(editBook);
        }
        print.tableFooter(MODIFY_REPORT_TABLE_WIDTH);

        // what record have been removed today
        print.toCenter("Removed Booking Today", MODIFY_REPORT_TABLE_WIDTH);
        print.tableMiddleLine(MODIFY_REPORT_TABLE_WIDTH);
```

```
        System.out.println(String.format(BLUE + "| %5s | %20s | %20s |", "ID", "NAME",
"MEAL"));
        print.tableMiddleLine(MODIFY_REPORT_TABLE_WIDTH);
        if ("".equals(removeBook)) {
            print.toCenter("No Removed Record Today", MODIFY_REPORT_TABLE_WIDTH);
            System.out.println();
        } else {
            System.out.println(removeBook);
        }
        print.tableFooter(MODIFY_REPORT_TABLE_WIDTH);
    }

    public void SummaryReport() {
        Iterator<MenuEntity> menuItr = MenuList.getIterator();
        //overrall record information
        print.tableHeader("SUMMARY REPORT", SUMMARY_TABLE_WIDTH);
        print.toCenter(String.format("Total of Booking: %d", bookList.sizeOf()),
SUMMARY_TABLE_WIDTH);
        print.tableMiddleLine(SUMMARY_TABLE_WIDTH);

        System.out.println(String.format(BLUE + "| %20s | %15s |", "Meal", "Amount"));
        print.tableMiddleLine(SUMMARY_TABLE_WIDTH);

        double totalAmount;

        while (menuItr.hasNext()) {
            MenuEntity m = menuItr.next();

            Iterator<Booking> bookItr = bookList.getIterator();

            totalAmount = 0;

            while (bookItr.hasNext()) {
                Booking bk = bookItr.next();

                if (m.getMenuName().equals(bk.getmeal())) {
                    totalAmount += bk.getAmount();
                }
            }
            System.out.println(String.format(BLUE + "| %20s | %15s |", m.getMenuName(),
totalAmount));
        }
        print.tableFooter(SUMMARY_TABLE_WIDTH);
    }

    public void addBooking() {
        print.tableHeader("Add Booking", TABLE_WIDTH);
        print.otherMsg("New Booking Details", 1);

        // generate Booking ID
        String BookID = "B" + String.format("%4d", bookIDindex++);
        System.out.println(BLUE + "Booking ID            : " + BookID);

        // Input booker name
        String BookName = nameValidation(BLUE + "Enter Booker Name     : ");
```

```
        // input quantity
        int quantity = scanner.nextInt(BLUE + "Enter Quantity Of Meals You Need      : ", "The
Maximum quantity of meals we can provide is 1000.Please Enter Your Quantity Again", 1,
1000);

        // input booker phone number
        String phoneNumber = phonenumberValidation(BLUE + "Enter Phone Number   : ");

        // Choose meals type
        print.otherMsg("Choose The Meals You Want", 0);
        displayMenuMeal();
        int mealChoose = scanner.nextInt(PURPLE + "Enter The Meals Your Want   : ",
String.format("Please enter a number between 1 to %d", MenuList.sizeOf()), 1,
MenuList.sizeOf());
        String MenuName = MenuList.get(mealChoose - 1).getMenuName();

        // count the amount ny using quantity and price
        MenuEntity m = MenuList.firstOrDefault(c ->
c.getMenuName().equalsIgnoreCase(MenuName));
        double amount = m.getPrice() * quantity;

        // Get local date
        DateTimeFormatter dte = DateTimeFormatter.ofPattern("dd-MM-yyyy");
        LocalDateTime now = LocalDateTime.now();

        // view for data have been input for checking purpose
        print.tableHeader("New Booking Details", SPECIFY_TABLE_WIDTH);
        System.out.printf(BLUE + "| %-18s | %-20s |\n", "Book ID", BookID);
        System.out.printf(BLUE + "| %-18s | %-20s |\n", "Booker Name ", BookName);
        System.out.printf(BLUE + "| %-18s | %-20d |\n", "Quantity", quantity);
        System.out.printf(BLUE + "| %-18s | %-20s |\n", "Phone Number", phoneNumber);
        System.out.printf(BLUE + "| %-18s | RM %-17.2f |\n", "Amount (RM)", amount);
        System.out.printf(BLUE + "| %-18s | %-20s |\n", "Date", dte.format(now));
        print.tableFooter(SPECIFY_TABLE_WIDTH);

        // confirm want to add data or not
        print.otherMsg("Please Makesure All The Details Are Correct.", 1);
        if (scanner.confirmation(BLUE + "Please Confirm Do You Want To Add This Booking?
(Y = yes / N = no) >> ")) {

            bookList.add(new Booking(BookID, BookName, quantity, phoneNumber, MenuName,
amount, dte.format(now)));
            addNewBook += String.format(BLUE + "| %5s | %20s | %20s |\n", BookID,
BookName, MenuName);

            print.success("This Booking Is Added Successful!");
        } else {
            print.cancelled("Your Booking Has Been Cancelled.");
        }
    }

    public void removeBooking() {
        // to makesure the record is empty
        if (bookList.isEmpty()) {
```

```
            print.failed("No Booking Record.");
        } else {
            print.tableHeader("Remove Booking", LIST_TABLE_WIDTH);
            displayList(bookList, MenuList);
            print.otherMsg(String.format("Total Booking Record : %d", bookList.sizeOf()), 0);

            // get the book id
            String BookID = scanner.nextLine(BLUE + "\nPlease Enter The Book ID To Remove:
");
            Booking bookRemove = bookList.firstOrDefault(r ->
r.getBookID().equalsIgnoreCase(BookID));

            // find the same id in the record
            if (bookRemove == null) {
                print.failed("Book ID Not Found");
            } else {
                // Show the record that search by user
                searchDisplay(bookRemove, MenuList);

                if (scanner.confirmation(BLUE + "Sure want to remove booking above? (Y = yes /
N = no) >> ")) {
                    bookList.remove(bookRemove);
                    removeBook += String.format(BLUE + "| %5s | %20s | %20s |\n",
bookRemove.getBookID(), bookRemove.getBookName(), bookRemove.getmeal());
                    print.success("The Record You Choose Is Successful Been Deleted!");
                } else {
                    print.cancelled("The Remove Request Has Been Cancelled");
                }
            }
        }
    }

    public void editBooking() {
        if (bookList.isEmpty()) {
            print.failed("No Such Booking In The Record");
        } else {
            print.tableHeader("EDIT BOOKING", LIST_TABLE_WIDTH);
            displayList(bookList, MenuList);
            print.otherMsg(String.format("Total Number Of Booking Record: %d",
bookList.sizeOf()), 0);

            String BookID = scanner.nextLine(BLUE + "Enter The Book ID To Edit: ");
            Booking editBooking = bookList.firstOrDefault(b ->
b.getBookID().equalsIgnoreCase(BookID));

            if (editBooking == null) {
                print.failed("Booking ID not found");
            } else {
                // Input booking name
                String BookName = nameValidation(BLUE + "Enter New Booking Name      : ");
                int quantity = scanner.nextInt(BLUE + "Enter New Quantity       : ", "The Maximum
quantity of meals we can provide is 1000.Please Enter Your Quantity Again", 1, 1000);
                String phoneNumber = phonenumberValidation(BLUE + "Enter New Phone
Number    : ");
```

```
            print.otherMsg("Choose The Meals You Want", 0);
            displayMenuMeal();
            int mealChoose = scanner.nextInt(BLUE + "Enter The Meal You Want   : ",
String.format(RED + "Please enter a number between 1 to %d", MenuList.sizeOf()), 1,
MenuList.sizeOf());
            String MenuName = MenuList.get(mealChoose - 1).getMenuName();
            MenuEntity m = MenuList.firstOrDefault(n ->
n.getMenuName().equalsIgnoreCase(MenuName));
            double amount = m.getPrice() * quantity;

            // confirm want to edit the record
            if (scanner.confirmation(BLUE + "Do you sure you want to edit this record? (Y =
yes / N = no) >> ")) {
                editBooking.setBookName(BookName);
                editBooking.setquantity(quantity);
                editBooking.setPhoneNumber(phoneNumber);
                editBooking.setmeal(MenuName);
                editBooking.setAmount(amount);
                editBook += String.format("| %5s | %20s | %20s |\n", editBooking.getBookID(),
BookName, amount);
                print.success("The record you edit is successful been edited! ");
            } else {
                print.cancelled("Your Edit Request Is Been Cancelled");
            }
        }
    }
  }

  public void searchBooking() {
    print.tableHeader("SEARCH BOOKING", TABLE_WIDTH);
    System.out.println(BLUE + "1. Search Book ID                    |");
    System.out.println(BLUE + "2. Search Booker Name                |");
    System.out.println(BLUE + "3. Search Phone Number               |");
    System.out.println(BLUE + "4. Back To Previous Page             |");  //which user want to
search by
    print.tableFooter(TABLE_WIDTH);

    int choice = scanner.nextInt(PURPLE + "Please Enter Which You Want To Search By: ",
"Please enter a number between 1 to 5", 1, 5);

    switch (choice) {
        case 1:
            searchBookingId();
            break;
        case 2:
            searchBookingName();
            break;
        case 3:
            searchBookingPhoneNumber();
            break;
        default: {
        }
        break;
    }
  }
```

```
    public void searchBookingId() {
        print.tableHeader("SEARCH BOOK ID", TABLE_WIDTH);
        String BookID = scanner.nextLine(BLUE + "Enter book id to search (e.g. B0001): ");

        Booking bk = bookList.firstOrDefault(b -> b.getBookID().equalsIgnoreCase(BookID));

        if (bk != null) {
            searchDisplay(bk, MenuList);
        } else {
            print.failed("Book ID not found");
        }
    }

    public void searchBookingName() {
        print.tableHeader("SEARCH BOOKER NAME", TABLE_WIDTH);
        System.out.println(BLUE + "1. Search Name Starts With");
        System.out.println(BLUE + "2. Search Name Ends With");
        System.out.println(BLUE + "3. Search Name Contains");
        System.out.println(BLUE + "4. Back To Search Main Page");
        print.tableFooter(TABLE_WIDTH);
        int choice = scanner.nextInt(BLUE + "Please Choose Which Ways You Want To Choose
By: ", "Please enter a number between 1 to 4", 1, 4);

        if (choice != 4) {
            final String BookName = scanner.nextLine("Please Enter Name To Search:
").toUpperCase();
            ListInterface<Booking> bList = new DoublyLinkedList<>();

            switch (choice) {
                case 1:
                    bList = bookList.where(b ->
b.getBookName().toUpperCase().startsWith(BookName));
                    break;
                case 2:
                    bList = bookList.where(b ->
b.getBookName().toUpperCase().endsWith(BookName));
                    break;
                case 3:
                    bList = bookList.where(b ->
b.getBookName().toUpperCase().contains(BookName));
                    break;
                default:
                    break;
            }

            if (bList.isEmpty()) {
                System.out.println(BLUE + "Booker Name not found");
            } else {
                searchDisplay(bList);
            }
        }
    }

    public void searchBookingPhoneNumber() {
```

```
        print.tableHeader("SEARCH BY PHONE NUMBER", TABLE_WIDTH);
        System.out.println(BLUE + "1. Phone Number starts with");
        System.out.println(BLUE + "2. Phone Number ends with");
        System.out.println(BLUE + "3. Phone Number contains");
        System.out.println(BLUE + "4. Back to Previous");
        int choice = scanner.nextInt(BLUE + "Please Choose Which Ways You Want To Choose
By: ", "Please enter a number between 1 to 4", 1, 4);

        if (choice != 4) {
            final String BookphoneNumber = scanner.nextLine("Enter phone number to search:
").toUpperCase();
            ListInterface<Booking> bList = new DoublyLinkedList<>();

            switch (choice) {
                case 1:
                    bList = bookList.where(b ->
b.getPhoneNumber().toUpperCase().startsWith(BookphoneNumber));
                    break;
                case 2:
                    bList = bookList.where(b ->
b.getPhoneNumber().toUpperCase().endsWith(BookphoneNumber));
                    break;
                case 3:
                    bList = bookList.where(b ->
b.getPhoneNumber().toUpperCase().contains(BookphoneNumber));
                    break;
                default:
                    break;
            }

            if (bList.isEmpty()) {
                print.failed("Booker phone number not found");
            } else {
                searchDisplay(bList);
            }
        }
    }

    public void sortBookingBy() {
        print.tableHeader("SORT BOOKING RECORD BY", TABLE_WIDTH);

        System.out.println(BLUE + "1. Sort By Book ID                    |");
        System.out.println(BLUE + "2. Sort By Booker Name                |");
        System.out.println(BLUE + "3. Sort By Date                     |");
        System.out.println(BLUE + "4. Back To Previous Page              |");
        print.tableFooter(TABLE_WIDTH);

        int choice = scanner.nextInt("Please Choose Which You Want To Sort By: ", "Please
enter a number between 1 to 6", 1, 6);

        if (choice != 4) {
            print.otherMsg("Sort By List", 1);
            System.out.println(BLUE + "1. Ascending Order");
            System.out.println(BLUE + "2. Descending Order");
            System.out.println(BLUE + "3. Back To Previous Page");
```

```
        int sequenceType = scanner.nextInt(BLUE + "Enter Your Choice: ", "Please enter a
number between 1 to 3", 1, 3);

        if (sequenceType != 3) {
            switch (choice) {
                case 1:
                    sortById(sequenceType);
                    break;
                case 2:
                    sortByName(sequenceType);
                    break;
                case 3:
                    sortByDate(sequenceType);
                    break;
                default: {
                }
                break;
            }
            print.success("Sort successfully, press 6 or 7 to view booking record");
        }
    }
}

public void sortById(int sequenceType) {
    switch (sequenceType) {
        case 1:
            bookList.orderBy((Booking bk1, Booking bk2) -> {
                int sortval = bk1.getBookID().compareToIgnoreCase(bk2.getBookID());

                if (sortval == 0) {
                    return bk1.getBookName().compareToIgnoreCase(bk2.getBookName()) < 0 ?
OrderClause.MOVE_FORWARD : OrderClause.MOVE_BACKWARD;
                } else {
                    return sortval < 0 ? OrderClause.MOVE_FORWARD :
OrderClause.MOVE_BACKWARD;
                }
            });
            break;
        case 2:
            bookList.orderBy((Booking bk1, Booking bk2) -> {
                int sortval = bk1.getBookID().compareToIgnoreCase(bk2.getBookID());

                if (sortval == 0) {
                    return bk1.getBookName().compareToIgnoreCase(bk2.getBookName()) < 0 ?
OrderClause.MOVE_FORWARD : OrderClause.MOVE_BACKWARD;
                } else {
                    return sortval > 0 ? OrderClause.MOVE_FORWARD :
OrderClause.MOVE_BACKWARD;
                }
            });
            break;
        default: {
        }
        break;
```

```
    }
  }

  public void sortByName(int sequenceType) {
    switch (sequenceType) {
      case 1:
        bookList.orderBy((Booking bk1, Booking bk2) -> {
          int sortval = bk1.getBookName().compareToIgnoreCase(bk2.getBookName());

          if (sortval == 0) {
            return bk1.getBookName().compareToIgnoreCase(bk2.getBookName()) < 0 ?
OrderClause.MOVE_FORWARD : OrderClause.MOVE_BACKWARD;
          } else {
            return sortval < 0 ? OrderClause.MOVE_FORWARD :
OrderClause.MOVE_BACKWARD;
          }
        });
        break;
      case 2:
        bookList.orderBy((Booking bk1, Booking bk2) -> {
          int sortval = bk1.getBookName().compareToIgnoreCase(bk2.getBookName());

          if (sortval == 0) {
            return bk1.getBookName().compareToIgnoreCase(bk2.getBookName()) < 0 ?
OrderClause.MOVE_FORWARD : OrderClause.MOVE_BACKWARD;
          } else {
            return sortval > 0 ? OrderClause.MOVE_FORWARD :
OrderClause.MOVE_BACKWARD;
          }
        });
        break;
      default: {

      }
    }
  }

  public void sortByDate(int sequenceType) {
    SimpleDateFormat dte = new SimpleDateFormat("dd-MM-yyyy");

    switch (sequenceType) {
      case 1:
        bookList.orderBy((Booking bk1, Booking bk2) -> {
          int sortval = 0;
          try {
            sortval = dte.parse(bk1.getDate()).compareTo(dte.parse(bk2.getDate()));
          } catch (ParseException e) {
          }

          if (sortval == 0) {
            return bk1.getBookName().compareToIgnoreCase(bk2.getBookName()) < 0 ?
OrderClause.MOVE_FORWARD : OrderClause.MOVE_BACKWARD;
          } else {
            return sortval < 0 ? OrderClause.MOVE_FORWARD :
OrderClause.MOVE_BACKWARD;
```

```
                }
            });
            break;
        case 2:
            bookList.orderBy((Booking bk1, Booking bk2) -> {
                int sortval = 0;
                try {
                    sortval = dte.parse(bk1.getDate()).compareTo(dte.parse(bk2.getDate()));
                } catch (ParseException e) {
                }

                if (sortval == 0) {
                    return bk1.getBookName().compareToIgnoreCase(bk2.getBookName()) < 0 ?
OrderClause.MOVE_FORWARD : OrderClause.MOVE_BACKWARD;
                } else {
                    return sortval > 0 ? OrderClause.MOVE_FORWARD :
OrderClause.MOVE_BACKWARD;
                }
            });
            break;
        default: {
        }
        break;
    }
  }

  public void searchDisplay(Booking book, ListInterface<MenuEntity> MenuList) {
// Search for 1 result
    Iterator<MenuEntity> menuItr = MenuList.getIterator();

    while (menuItr.hasNext()) {
      MenuEntity m = menuItr.next();

      if (book.getmeal().compareToIgnoreCase(m.getMenuName()) == 0) {
      }
    }

    print.tableHeader("DONOR SEARCH RESULT", SPECIFY_TABLE_WIDTH);
    System.out.printf(BLUE + "| %-18s | %-20s |\n", "Book ID", book.getBookID());
    System.out.printf(BLUE + "| %-18s | %-20s |\n", "Book Name ", book.getBookName());
    System.out.printf(BLUE + "| %-18s | %-20d |\n", "Quantity", book.getquantity());
    System.out.printf(BLUE + "| %-18s | %-20s |\n", "Phone Number",
String.valueOf(book.getPhoneNumber()).replaceFirst("(\\d{1})(\\d{2})(\\d{3})(\\d+)",
"(+6$1)$2-$3 $4"));
    System.out.printf(BLUE + "| %-18s | %-20s |\n", "Donation Category", book.getmeal());
    System.out.printf(BLUE + "| %-18s | %-20.2f |\n", "Amount", book.getAmount());
    System.out.printf(BLUE + "| %-18s | %-20s |\n", "Date", book.getDate());
    print.tableFooter(SPECIFY_TABLE_WIDTH);
  }

  public void searchDisplay(ListInterface<Booking> bookList) {      // Search for more than 1
results
    print.tableHeader("BOOKING SEARCH RESULT", LIST_TABLE_WIDTH);
    displayList(bookList, MenuList);
  }
```

```
    public void showBookingList() {
        Iterator<MenuEntity> menuItr = MenuList.getIterator();
        print.tableHeader("DISPLAY Booking Record", TABLE_WIDTH);

        int backNum = 1;
        for (int i = 0; menuItr.hasNext(); i++) {
            System.out.println(String.format(BLUE + "%d. Only Display %s Meals", (i + 1),
menuItr.next().getMenuName()));
            backNum++;
        }

        System.out.printf(BLUE + "%d. Back To Previous Main Page\n", backNum);
        print.tableFooter(TABLE_WIDTH);

        int choice = scanner.nextInt(PURPLE + "Enter Your Choice: ", String.format("Please
enter a number between 1 to %d", backNum), 1, backNum);

        if (choice != backNum) {
            int numOf = 0;
            // Get category name
            String menuName = MenuList.get(choice - 1).getMenuName();

            print.tableHeader(String.format("Booking Details (%s)", menuName.toUpperCase()),
LIST_TABLE_WIDTH);

            Iterator<Booking> bookItr = bookList.getIterator();

            System.out.printf(BLUE + "| %-5s | %-20s | %-5s | %-20s | %-15s | %-13s |\n", "ID",
"NAME", "AGE", "PHONE NUMBER", "AMOUNT", "DATE");
            print.tableMiddleLine(LIST_TABLE_WIDTH);
            while (bookItr.hasNext()) {
                Booking bk = bookItr.next();
                if (menuName.compareToIgnoreCase(bk.getmeal()) == 0) {
                    System.out.println(bk);
                    numOf++;
                }
            }

            if (numOf == 0) {
                print.toCenter(String.format("No Record Found In %s Meal", menuName),
LIST_TABLE_WIDTH);
            }

            print.tableFooter(LIST_TABLE_WIDTH);
            print.otherMsg(String.format("Total Number Of %s Meals: %d", menuName, numOf),
0);
        }
    }

    // Display all booking record
    public void allrecord() {
        print.tableHeader("ALL BOOK RECORD", LIST_TABLE_WIDTH);
        displayList(bookList, MenuList);
        print.otherMsg(String.format("Total Number Of Booking: %d", bookList.sizeOf()), 0);
```

```
    }

    // Display booking record by meal
    public void displayList(ListInterface<Booking> bookList, ListInterface<MenuEntity>
MenuList) {
        Iterator<MenuEntity> menuItr = MenuList.getIterator();
        int numOf;

        while (menuItr.hasNext()) {
            MenuEntity m = menuItr.next();
            print.toCenter(RED + m.getMenuName().toUpperCase(), 104);
            print.tableMiddleLine(LIST_TABLE_WIDTH);

            System.out.printf(BLUE + "| %-5s | %-20s | %-5s | %-20s | %-15s | %-10s |\n", "ID",
"NAME", "QUANTITY", "PHONE NUMBER", "AMOUNT", "DATE");
            print.tableMiddleLine(LIST_TABLE_WIDTH);

            Iterator<Booking> bookItr = bookList.getIterator();

            numOf = 0;

            while (bookItr.hasNext()) {
                Booking bk = bookItr.next();
                if (bk.getmeal().compareToIgnoreCase(m.getMenuName()) == 0) {
                    System.out.println(bk);
                    numOf++;
                }
            }

            if (numOf == 0) {
                print.toCenter("No record for this meal!", LIST_TABLE_WIDTH);
            }
            print.tableFooter(LIST_TABLE_WIDTH);
        }
    }

    public void displayMenuMeal() {
        Iterator<MenuEntity> menuItr = MenuList.getIterator();

        for (int i = 0; menuItr.hasNext(); i++) {
            System.out.println(String.format(PURPLE + "%d. %s", (i + 1),
menuItr.next().getMenuName()));
        }
    }

    // name sortvalidation
    public String nameValidation(String promptInfo) {
        boolean sortvalidname = false, inputname = true;
        String mainErrorName = "Please enter a sortvalid Booker name (e.g. Jonas)";
        int numChar = 0;
        String BookName;

        do {
            BookName = scanner.nextLine(promptInfo);
```

```
            inputname = true;
            for (int i = 0; i < BookName.length(); i++) {
                if (Character.isLetter(BookName.charAt(i)) == false && BookName.charAt(i) != ' ') {
                    inputname = false;                          //to makesure the input name only in char
format
                } else if (Character.isLetter(BookName.charAt(i))) {
                    numChar++;
                }
            }

            if (!inputname) {                                   // if the input name is insortvalid will print error
message
                print.error(mainErrorName);
                print.error("The Booker Name Is Insortvalid Because Contain Special Symbols Or
Numbers");
            } else if (BookName.length() <= 0 || numChar == 0) {
                print.error(mainErrorName);
                print.error("The Booker Name Cannot Be Empty!");
            } else if (BookName.length() >= 20) {
                print.error(mainErrorName);
                print.error("The Booker Name Must Less Than 20 Characters For Record
Purpose! ");
            } else {
                sortvalidname = true;
            }

        } while (!sortvalidname);

        return BookName;
    }

    public String phonenumberValidation(String promptInfo) {
        boolean sortvalidphone = false;
        String mainErrorPhoneNumber = "Please enter a sortvalid phone number (e.g.
0129431228)";
        String phoneNumber;

        do {
            phoneNumber = scanner.nextLine(promptInfo);

            if (!phoneNumber.matches("[0-9]+")) {
                print.error(mainErrorPhoneNumber);
                print.error("The Phone Number Cannot Contain Characters Or Special Symbols");
            } else if (phoneNumber.charAt(0) != '0' || phoneNumber.charAt(1) != '1') {
                print.error(mainErrorPhoneNumber);
                print.error("The Phone Number Format Must Start With 01!");
            } else if (phoneNumber.length() < 10) {
                print.error(mainErrorPhoneNumber);
                print.error("The Phone Number Cannot Less Than 10 Digits!");
            } else if (phoneNumber.length() > 11) {
                print.error(mainErrorPhoneNumber);
                print.error("The phone number cannot more than 11 Digits!");
            } else {
                sortvalidphone = true;
            }
```

```
        } while (!sortvalidphone);

        return phoneNumber;
    }
}
```

5. Client Program

I've been implementing a list of menu items and booking information in the client software using a doubly linked list.

The ability of a doubly linked list to go both ahead and backward is the primary reason I use one. For successful rearranging, my client category manages a list of menu items and a list of booking information. To make it simpler to sort the list in ascending and descending order, we may do it using forward and backward traversal. Clients will be able to more quickly and effectively sort the list and search results from it. The second reason is that the delete operation in the doubly linked list will be more effective if a reference to the node to be destroyed is provided. If we no longer require it, we may more successfully delete the menu items from the menu list or remove the reservation from the reservation record. If not, we may rapidly add a new node before the target node. A reference to the preceding node is required in a singly linked list in order to remove a node. Sometimes the list is traversed to obtain this preceding node. We may access the previous node in the doubly linked list using the prior pointer. Because of this, the doubly linked list is ideal in this scenario for the module's efficient and effective administration of several entities.

Menu class

```
package client;

import adt.DoublyLinkedList;
import adt.ListInterface;
import entity.Booking;
import entity.MenuEntity;
import utility.ScannerHandler;
import utility.DesignConsole;
import static utility.DesignConsole.BLUE;
import static utility.DesignConsole.PURPLE;

/**
 *
 * @author Lee Jing Jet
 */
public class Menu {

    /**
     * @param args the command line arguments
```

```
     */
    private ListInterface<Booking> bookList;
    private ListInterface<MenuEntity> MenuList;
    private BookingMaintain bookRecord;
    private MealsMenu MealsRecord;

    private final ScannerHandler scanner = new ScannerHandler();
    private final DesignConsole print = new DesignConsole();

    public Menu() {
        bookList = new DoublyLinkedList();
        addBookingDetails();

        MenuList = new DoublyLinkedList();
        addMenuList();

        bookRecord = new BookingMaintain(bookList, MenuList);
        MealsRecord = new MealsMenu(MenuList);
    }

    public void addBookingDetails() {
        Booking d1 = new Booking("B1001", "Jet", 50, "0127835566", "Fried Rice", 600,
"01-01-2022");
        Booking d2 = new Booking("B1002", "Jackson", 100, "0162049399", "Fried Noodles",
1000, "12-03-2022");
        Booking d3 = new Booking("B1003", "Shanice", 25, "0178832213", "Fried Noodles",
250, "25-03-2022");
        Booking d4 = new Booking("B1004", "Louis", 30, "0199431228", "Curry Chicken", 600,
"21-05-2022");
        Booking d5 = new Booking("B1005", "Miko", 10, "0112116711", "Curry Chicken", 200,
"18-06-2022");

        bookList.addAll(d1, d2, d3, d4, d5);
    }

    public void addMenuList() {
            MenuList.add(new MenuEntity("M1001", "Fried Rice", 12.00));
        MenuList.add(new MenuEntity("M1002", "Fried Noodles", 10.00));
        MenuList.add(new MenuEntity("M1003", "Curry Chicken", 20.00));
    }

    public void run() {
        int choice;
        do {
            print.tableHeader("Catering Services System", 40);
            System.out.println(BLUE + "| 1. Meals Menu                    |");
            System.out.println(BLUE + "| 2. Customer Booking                |");
            System.out.println(BLUE + "| 3. Exit                          |");
```

```
        System.out.println(PURPLE +
"=======================================");
        choice = scanner.nextInt(BLUE + "Please Enter Your Choice: ", "Your choice is
Invalid Pls Re-enter", 1, 3);

        switch (choice) {
            case 1:
                MealsRecord.main();
                break;
            case 2:
                bookRecord.main();
                break;
            default: {
            }
            break;
        }

    } while (choice != 3);

    print.otherMsg("Thank you. See You Next Time!", 1);
  }

  public static void main(String[] args) {
    new Menu().run();
  }

  public static String getDivider(char lineTable, int length) {
    StringBuilder stringBuilder = new StringBuilder();
    for (int i = 0; i < length; i++) {
      stringBuilder.append(lineTable);
    }
    return stringBuilder.toString();
  }
}
```

MealsMenu class

```
package client;

import adt.DoublyLinkedList;
import adt.ListInterface;
import adt.OrderClause;
import entity.MenuEntity;
import utility.DesignConsole;
import utility.ScannerHandler;
import java.util.Iterator;
import static utility.DesignConsole.BLUE;
import static utility.DesignConsole.PURPLE;
```

```
/**
 *
 * @author Lee Jing Jet
 */
public class MealsMenu {

    private ListInterface<MenuEntity> MenuList;
    private final ScannerHandler scanner = new ScannerHandler();
    private final DesignConsole print = new DesignConsole();

    private int menuIndex = 1004;
    private static final int TABLE_WIDTH = 40, LIST_TABLE_WIDTH = 60,
SPECIFY_TABLE_WIDTH = 50;

    public MealsMenu(ListInterface<MenuEntity> MenuList) {
        this.MenuList = MenuList;
    }

    public void main() {
        int choice;

        do {
            print.tableHeader("Meals Menu", TABLE_WIDTH);
            System.out.println(BLUE + "| 1. View Menu                    |");
            System.out.println(BLUE + "| 2. Add Menu                     |");
            System.out.println(BLUE + "| 3. Remove Menu                  |");
            System.out.println(BLUE + "| 4. Edit Menu                    |");
            System.out.println(BLUE + "| 5. Search Menu                  |");
            System.out.println(BLUE + "| 6. Sort Menu By                 |");
            System.out.println(BLUE + "| 7. Exit                         |");
            System.out.println(PURPLE +
"=======================================");

            choice = scanner.nextInt(BLUE + "Enter Your Choice: ", "Please enter a number
between 1 to 7", 1, 7);

            switch (choice) {
                case 1:
                    showMenuRecord();
                    break;
                case 2:
                    addMeal();
                    break;
                case 3:
                    removeMeal();
                    break;
                case 4:
                    editMenu();
```

```
                break;
            case 5:
                searchMenu();
                break;
            case 6:
                sortByList();
                break;
            default: {
            }
            break;
        }

    } while (choice != 7);
}

public void addMeal() {
    print.tableHeader("ADD MEAL", TABLE_WIDTH);
    print.otherMsg("New MENU MEAL", 1);

    // generate menu id
    String MenuID = "M" + String.format("%4d", menuIndex++);
    System.out.println(BLUE +"Category ID         : " + MenuID);

    // input Menu Name
    String MenuName = scanner.nextLine(BLUE + "Enter Menu Name : ");

    // input price
    Double price = scanner.nextDouble(BLUE +"Enter Price: ");


    // Show the details
    print.tableHeader("New Category Details", SPECIFY_TABLE_WIDTH);
    System.out.printf(BLUE +"| %-18s | %-25s |\n", "Menu ID", MenuID);
    System.out.printf(BLUE +"| %-18s | %-25s |\n", "Menu Name", MenuName);
    System.out.printf(BLUE +"| %-18s | %-25.2f |\n", "Price", price);
    print.tableFooter(SPECIFY_TABLE_WIDTH);

    // Confirmation
    print.otherMsg("Please make sure that all the details given above is correct", 1);
    print.hint("The above details can be edited in the future");

    if (scanner.confirmation("Do You Sure You Want To Add This Meals (Y = yes / N = no)?
>> ")) {
        MenuList.add(new MenuEntity(MenuID, MenuName, price));
        print.success("The Meal Is Successful Added");
    } else {
        print.cancelled("Your Add Request Has Been Cancelled");
    }
```

```
    }

    public void removeMeal() {
        // Check whether is empty
        if (MenuList.isEmpty()) {
            print.failed("No Meal Record!");
        } else {
            // Show all meal record
            print.tableHeader("REMOVE MENU", LIST_TABLE_WIDTH);
            displayMenu(MenuList);
            print.tableFooter(LIST_TABLE_WIDTH);
            print.otherMsg(String.format("Total Number Of Meals: %d", MenuList.sizeOf()), 0);

            // Get the Menu id
            String MenuID = scanner.nextLine(BLUE +"\nEnter The Menu ID To Remove (e.g.
M1001): ");
            MenuEntity removeMenu = MenuList.firstOrDefault(d ->
d.getMenuID().equalsIgnoreCase(MenuID));

            // Compare the id with record
            if (removeMenu == null) {
                print.failed("Menu ID Not Found");
            } else {
                // Show user search result
                displaySearchMenu(BLUE+"Menu to remove", removeMenu);

                if (scanner.confirmation(BLUE+"Are You Sure You Want To Remove This Meal? (Y
= yes / N = no) >> ")) {
                    MenuList.remove(removeMenu);
                    print.success("The Menu You Choose Has Been Removed Successful");
                } else {
                    print.cancelled("Your Remove Request Has Been Cancelled");
                }
            }
        }
    }

    public void editMenu() {
        // Check whether is empty
        if (MenuList.isEmpty()) {
            print.failed("No Menu Record!");
        } else {
            // Show all Menu Record
            print.tableHeader("EDIT MENU", LIST_TABLE_WIDTH);
            displayMenu(MenuList);
            print.tableFooter(LIST_TABLE_WIDTH);
            print.otherMsg(String.format("Total Number Of Meal: %d", MenuList.sizeOf()), 0);
```

```
        // Input menu id
        String MenuID = scanner.nextLine(BLUE+"\nEnter The Menu ID To Edit (e.g.
M1001): ");
        MenuEntity editMenu = MenuList.firstOrDefault(m ->
m.getMenuID().equalsIgnoreCase(MenuID));

        // Get the category id
        if (editMenu == null) {
            print.failed("Menu ID Not Found");
        } else {
            displaySearchMenu(BLUE +"Menu to edit", editMenu);
            // Enter new meal's name
            String MenuName = scanner.nextLine("Enter New Menu Name      : ");
            double Price = scanner.nextDouble("Enter New Menu Price      : ");

            if (scanner.confirmation(BLUE+"Do Your Sure You Want To Edit This Menu? (Y =
yes / N = no) >> ")) {
                // Set the details into the list
                editMenu.setMenuName(MenuName);
                editMenu.setPrice(Price);

                print.success("Your Record Is Record Successful");
            } else {
                print.cancelled("Your Edit Request Has Been Cancelled");
            }
        }
    }
  }

  public void searchMenu() {
    // Choose search by
    print.tableHeader("SEARCH MENU", TABLE_WIDTH);
    System.out.println(BLUE+"1. Search By Category ID");
    System.out.println(BLUE+"2. Search By Category Name");
    System.out.println(BLUE+"3. Back To Donation Category Main Page");

    int choice = scanner.nextInt(BLUE+"Enter Your Choice: ", "Please enter a number
between 1 to 3", 1, 3);

    switch (choice) {
        case 1:
            searchMenuId();
            break;
        case 2:
            searchMenuName();
            break;
        default: {
        }
```

```
            break;
        }
    }

    public void searchMenuId() {
        print.tableHeader("SEARCH MENU ID", TABLE_WIDTH);
        String MenuID = scanner.nextLine(BLUE+"Enter menu id to search (e.g. M1001): ");

        // Check id
        MenuEntity me = MenuList.firstOrDefault(m ->
m.getMenuID().equalsIgnoreCase(MenuID));

        // Show result
        if (me != null) {
            displaySearchMenu(BLUE+"Menu search result", me);
        } else {
            print.failed("Menu ID not found");
        }
    }

    public void searchMenuName() {
        print.tableHeader("SEARCH MENU NAME", TABLE_WIDTH);
        System.out.println(BLUE+"1. Search Name Starts With");
        System.out.println(BLUE+"2. Search Name Ends With");
        System.out.println(BLUE+"3. Search Name Contains");
        System.out.println(BLUE+"4. Back To Search Main Page");
        int choice = scanner.nextInt(BLUE+"Enter Your Choice: ", "Please enter a number
between 1 to 4", 1, 4);

        if (choice != 4) {
            String MenuName = scanner.nextLine(BLUE+"Enter name to search:
").toUpperCase();
            ListInterface<MenuEntity> mList = new DoublyLinkedList<>();

            // Get all menu which related with the input
            switch (choice) {
                case 1:
                    mList = MenuList.where(m ->
m.getMenuName().toUpperCase().startsWith(MenuName));
                    break;
                case 2:
                    mList = MenuList.where(m ->
m.getMenuName().toUpperCase().endsWith(MenuName));
                    break;
                case 3:
                    mList = MenuList.where(d ->
d.getMenuName().toUpperCase().contains(MenuName));
                    break;
```

```
            default:
                break;
        }

        // Show result
        if (mList.isEmpty()) {
            print.failed("Menu name not found");
        } else {
            displaySearchMenu(mList);
        }
    }
}


public void sortByList() {
    print.tableHeader("SORT MENU LIST BY", TABLE_WIDTH);

    System.out.println(BLUE+"1. Sort By Menu ID");
    System.out.println(BLUE+"2. Sort By Menu Name");
    System.out.println(BLUE+"3. Back To Previous Page");

    int choice = scanner.nextInt(BLUE+"Enter Your Choice: ", "Please enter a number
between 1 to 4", 1, 4);

    if (choice != 3) {
        print.otherMsg("Sort Menu By", 1);
        System.out.println(BLUE+"1. Ascending Order");
        System.out.println(BLUE+"2. Descending Order");
        System.out.println(BLUE+"3. Back To Previous Page");

        int sequenceType = scanner.nextInt(BLUE+"Enter Your Choice: ", "Please enter a
number between 1 to 2", 1, 2);

        if (sequenceType != 3) {
            switch (choice) {
                case 1:
                    sortById(sequenceType);
                    break;
                case 2:
                    sortByName(sequenceType);
                    break;
                default: {
                }
                break;
            }
            print.success("Sort successfully, press 6 to view menu record");
        }
    }
```

```
}

public void sortById(int sequenceType) {
    switch (sequenceType) {
        case 1:
            MenuList.orderBy((m1, m2)
                    -> m1.getMenuID().compareToIgnoreCase(m2.getMenuID()) < 0
                    ? OrderClause.MOVE_FORWARD : OrderClause.MOVE_BACKWARD);
            break;
        case 2:
            MenuList.orderBy((m1, m2)
                    -> m1.getMenuID().compareToIgnoreCase(m2.getMenuID()) > 0
                    ? OrderClause.MOVE_FORWARD : OrderClause.MOVE_BACKWARD);
            break;
        default: {
        }
        break;
    }
}

public void sortByName(int sequenceType) {
    switch (sequenceType) {
        case 1:
            MenuList.orderBy((m1, m2)
                    -> m1.getMenuName().compareToIgnoreCase(m2.getMenuName()) < 0
                    ? OrderClause.MOVE_FORWARD : OrderClause.MOVE_BACKWARD);
            break;
        case 2:
            MenuList.orderBy((m1, m2)
                    -> m1.getMenuName().compareToIgnoreCase(m2.getMenuName()) > 0
                    ? OrderClause.MOVE_FORWARD : OrderClause.MOVE_BACKWARD);
            break;
        default: {
        }
        break;
    }
}


public void showMenuRecord() {
    print.tableHeader("MENU", LIST_TABLE_WIDTH);
    displayMenu(MenuList);
    print.tableFooter(LIST_TABLE_WIDTH);
    print.otherMsg(String.format("Total Number Of Menu: %d", MenuList.sizeOf()), 0);
}

public void displaySearchMenu(String title, MenuEntity menu) {
    print.tableHeader(title.toUpperCase(), SPECIFY_TABLE_WIDTH);
```

```
        System.out.printf(BLUE+"| %-18s | %-25s |\n", "Category ID", menu.getMenuID());
        System.out.printf(BLUE+"| %-18s | %-25s |\n", "Category Name",
menu.getMenuName());
        System.out.printf(BLUE+"| %-18s | %-25.2f |\n", "Price", menu.getPrice());
        print.tableFooter(SPECIFY_TABLE_WIDTH);
    }

    public void displaySearchMenu(ListInterface<MenuEntity> menu) {
        print.tableHeader("MENU SEARCH RESULT", LIST_TABLE_WIDTH);
        displayMenu(menu);
        print.tableFooter(LIST_TABLE_WIDTH);
    }

    public void displayMenu(ListInterface<MenuEntity> MenuList) {
        Iterator<MenuEntity> itr = MenuList.getIterator();

        System.out.printf(BLUE+"| %-5s | %-20s | %-25s |\n", "ID", "NAME", "PRICE");
        print.tableMiddleLine(LIST_TABLE_WIDTH);

        while (itr.hasNext()) {
            MenuEntity me = itr.next();
            System.out.println(me.toString());
        }
    }
}
```

BookingMaintain class

```
package client;

import adt.DoublyLinkedList;
import adt.ListInterface;
import adt.OrderClause;
import entity.Booking;
import entity.MenuEntity;
import utility.ScannerHandler;
import utility.DesignConsole;
import java.util.Iterator;
import java.time.format.DateTimeFormatter;
import java.time.LocalDateTime;
import java.text.SimpleDateFormat;
import java.text.ParseException;
import static utility.DesignConsole.BLUE;
import static utility.DesignConsole.PURPLE;
import static utility.DesignConsole.RED;

/**
 *
```

```
 * @author Lee Jing Jet
 */
public class BookingMaintain {

    private ListInterface<Booking> bookList;
    private ListInterface<MenuEntity> MenuList;
    private final ScannerHandler scanner = new ScannerHandler();
    private final DesignConsole print = new DesignConsole();

    private int bookIDindex = 1011;
    private static final int TABLE_WIDTH = 40, LIST_TABLE_WIDTH = 97,
SPECIFY_TABLE_WIDTH = 45,
            MODIFY_REPORT_TABLE_WIDTH = 55, SUMMARY_TABLE_WIDTH = 42;

    private String addNewBook = "";
    private String editBook = "";
    private String removeBook = "";

    public BookingMaintain(ListInterface<Booking> bookList, ListInterface<MenuEntity>
MenuList) {
        this.bookList = bookList;
        this.MenuList = MenuList;

    }

    public void main() {
        int choice;

        do {
            print.tableHeader("Customer Booking", TABLE_WIDTH);
            System.out.println(BLUE + "| 1. Add Booking                    |");
            System.out.println(BLUE + "| 2. Remove Booking                 |");
            System.out.println(BLUE + "| 3. Edit Booking                   |");
            System.out.println(BLUE + "| 4. Search Booking                 |");
            System.out.println(BLUE + "| 5. Sort Booking By                |");
            System.out.println(BLUE + "| 6. Display Booking Details        |");
            System.out.println(BLUE + "| 7. Display Booking List By Meals  |");
            System.out.println(BLUE + "| 8. Display Reports                |");
            System.out.println(BLUE + "| 9. Exit                           |");
            System.out.println(PURPLE +
"=====================================");

            choice = scanner.nextInt(BLUE + "Please Enter Your Choice: ", "Please enter a
number between 1 to 9", 1, 9);

            switch (choice) {
                case 1:
                    addBooking();
```

```
                break;
            case 2:
                removeBooking();
                break;
            case 3:
                editBooking();
                break;
            case 4:
                searchBooking();
                break;
            case 5:
                sortBookingBy();
                break;
            case 6:
                allrecord();
                break;
            case 7:
                showBookingList();
                break;
            case 8:
                showReports();
                break;
            default: {
            }
            break;
        }
    } while (choice != 9);
}

public void showReports() {
    print.tableHeader("REPORT LIST", TABLE_WIDTH);
    System.out.println(BLUE + "1. Modify Report                    |");
    System.out.println(BLUE + "2. Summary Report                   |");
    System.out.println(BLUE + "3. Back To Previous Page            |");
    print.tableFooter(TABLE_WIDTH);

    int choice = scanner.nextInt(BLUE + "Enter Your Choice: ", "Please enter a number
between 1 to 3", 1, 3);

    switch (choice) {
        case 1:
            ModifyReport();
            break;
        case 2:
            SummaryReport();
            break;
        default: {
        }
```

```
            break;

        }
    }

    public void ModifyReport() {
        print.tableHeader(" MODIFY REPORT", MODIFY_REPORT_TABLE_WIDTH);

        // what record have added today
        print.toCenter("Added Booking Today", MODIFY_REPORT_TABLE_WIDTH);
        print.tableMiddleLine(MODIFY_REPORT_TABLE_WIDTH);
        System.out.println(String.format(BLUE + "| %5s | %20s | %20s |", "ID", "NAME",
"MEAL"));
        print.tableMiddleLine(MODIFY_REPORT_TABLE_WIDTH);
        if ("".equals(addNewBook)) {
            print.toCenter("No Added Record Today", MODIFY_REPORT_TABLE_WIDTH);
            System.out.println();
        } else {
            System.out.println(addNewBook);
        }
        print.tableFooter(MODIFY_REPORT_TABLE_WIDTH);

        // what record have edited today
        print.toCenter("Edited Booking Today", MODIFY_REPORT_TABLE_WIDTH);
        print.tableMiddleLine(MODIFY_REPORT_TABLE_WIDTH);
        System.out.println(String.format(BLUE + "| %5s | %20s | %20s |", "ID", "NAME",
"MEAL"));
        print.tableMiddleLine(MODIFY_REPORT_TABLE_WIDTH);
        if ("".equals(editBook)) {
            print.toCenter("No Edited Record Today", MODIFY_REPORT_TABLE_WIDTH);
            System.out.println();
        } else {
            System.out.println(editBook);
        }
        print.tableFooter(MODIFY_REPORT_TABLE_WIDTH);

        // what record have been removed today
        print.toCenter("Removed Booking Today", MODIFY_REPORT_TABLE_WIDTH);
        print.tableMiddleLine(MODIFY_REPORT_TABLE_WIDTH);
        System.out.println(String.format(BLUE + "| %5s | %20s | %20s |", "ID", "NAME",
"MEAL"));
        print.tableMiddleLine(MODIFY_REPORT_TABLE_WIDTH);
        if ("".equals(removeBook)) {
            print.toCenter("No Removed Record Today", MODIFY_REPORT_TABLE_WIDTH);
            System.out.println();
        } else {
            System.out.println(removeBook);
        }
```

```
        print.tableFooter(MODIFY_REPORT_TABLE_WIDTH);
    }

    public void SummaryReport() {
        Iterator<MenuEntity> menuItr = MenuList.getIterator();
        //overrall record information
        print.tableHeader("SUMMARY REPORT", SUMMARY_TABLE_WIDTH);
        print.toCenter(String.format("Total of Booking: %d", bookList.sizeOf()),
SUMMARY_TABLE_WIDTH);
        print.tableMiddleLine(SUMMARY_TABLE_WIDTH);

        System.out.println(String.format(BLUE + "| %20s | %15s |", "Meal", "Amount"));
        print.tableMiddleLine(SUMMARY_TABLE_WIDTH);

        double totalAmount;

        while (menuItr.hasNext()) {
            MenuEntity m = menuItr.next();

            Iterator<Booking> bookItr = bookList.getIterator();

            totalAmount = 0;

            while (bookItr.hasNext()) {
                Booking bk = bookItr.next();

                if (m.getMenuName().equals(bk.getmeal())) {
                    totalAmount += bk.getAmount();
                }
            }
            System.out.println(String.format(BLUE + "| %20s | %15s |", m.getMenuName(),
totalAmount));
        }
        print.tableFooter(SUMMARY_TABLE_WIDTH);
    }

    public void addBooking() {
        print.tableHeader("Add Booking", TABLE_WIDTH);
        print.otherMsg("New Booking Details", 1);

        // generate Booking ID
        String BookID = "B" + String.format("%4d", bookIDindex++);
        System.out.println(BLUE + "Booking ID            : " + BookID);

        // Input booker name
        String BookName = nameValidation(BLUE + "Enter Booker Name     : ");

        // input quantity
```

```
        int quantity = scanner.nextInt(BLUE + "Enter Quantity Of Meals You Need      : ", "The
Maximum quantity of meals we can provide is 1000.Please Enter Your Quantity Again", 1,
1000);

        // input booker phone number
        String phoneNumber = phonenumberValidation(BLUE + "Enter Phone Number   : ");

        // Choose meals type
        print.otherMsg("Choose The Meals You Want", 0);
        displayMenuMeal();
        int mealChoose = scanner.nextInt(PURPLE + "Enter The Meals Your Want   : ",
String.format("Please enter a number between 1 to %d", MenuList.sizeOf()), 1,
MenuList.sizeOf());
        String MenuName = MenuList.get(mealChoose - 1).getMenuName();

        // count the amount ny using quantity and price
        MenuEntity m = MenuList.firstOrDefault(c ->
c.getMenuName().equalsIgnoreCase(MenuName));
        double amount = m.getPrice() * quantity;

        // Get local date
        DateTimeFormatter dte = DateTimeFormatter.ofPattern("dd-MM-yyyy");
        LocalDateTime now = LocalDateTime.now();

        // view for data have been input for checking purpose
        print.tableHeader("New Booking Details", SPECIFY_TABLE_WIDTH);
        System.out.printf(BLUE + "| %-18s | %-20s |\n", "Book ID", BookID);
        System.out.printf(BLUE + "| %-18s | %-20s |\n", "Booker Name ", BookName);
        System.out.printf(BLUE + "| %-18s | %-20d |\n", "Quantity", quantity);
        System.out.printf(BLUE + "| %-18s | %-20s |\n", "Phone Number", phoneNumber);
        System.out.printf(BLUE + "| %-18s | RM %-17.2f |\n", "Amount (RM)", amount);
        System.out.printf(BLUE + "| %-18s | %-20s |\n", "Date", dte.format(now));
        print.tableFooter(SPECIFY_TABLE_WIDTH);

        // confirm want to add data or not
        print.otherMsg("Please Makesure All The Details Are Correct.", 1);
        if (scanner.confirmation(BLUE + "Please Confirm Do You Want To Add This Booking?
(Y = yes / N = no) >> ")) {

            bookList.add(new Booking(BookID, BookName, quantity, phoneNumber, MenuName,
amount, dte.format(now)));
            addNewBook += String.format(BLUE + "| %5s | %20s | %20s |\n", BookID,
BookName, MenuName);

            print.success("This Booking Is Added Successful!");
        } else {
            print.cancelled("Your Booking Has Been Cancelled.");
        }
```

```
    }

    public void removeBooking() {
        // to makesure the record is empty
        if (bookList.isEmpty()) {
            print.failed("No Booking Record.");
        } else {
            print.tableHeader("Remove Booking", LIST_TABLE_WIDTH);
            displayList(bookList, MenuList);
            print.otherMsg(String.format("Total Booking Record : %d", bookList.sizeOf()), 0);

            // get the book id
            String BookID = scanner.nextLine(BLUE + "\nPlease Enter The Book ID To Remove:
");
            Booking bookRemove = bookList.firstOrDefault(r ->
r.getBookID().equalsIgnoreCase(BookID));

            // find the same id in the record
            if (bookRemove == null) {
                print.failed("Book ID Not Found");
            } else {
                // Show the record that search by user
                searchDisplay(bookRemove, MenuList);

                if (scanner.confirmation(BLUE + "Sure want to remove booking above? (Y = yes /
N = no) >> ")) {
                    bookList.remove(bookRemove);
                    removeBook += String.format(BLUE + "| %5s | %20s | %20s |\n",
bookRemove.getBookID(), bookRemove.getBookName(), bookRemove.getmeal());
                    print.success("The Record You Choose Is Successful Been Deleted!");
                } else {
                    print.cancelled("The Remove Request Has Been Cancelled");
                }
            }
        }
    }

    public void editBooking() {
        if (bookList.isEmpty()) {
            print.failed("No Such Booking In The Record");
        } else {
            print.tableHeader("EDIT BOOKING", LIST_TABLE_WIDTH);
            displayList(bookList, MenuList);
            print.otherMsg(String.format("Total Number Of Booking Record: %d",
bookList.sizeOf()), 0);

            String BookID = scanner.nextLine(BLUE + "Enter The Book ID To Edit: ");
```

```
        Booking editBooking = bookList.firstOrDefault(b ->
b.getBookID().equalsIgnoreCase(BookID));

        if (editBooking == null) {
            print.failed("Booking ID not found");
        } else {
            // Input booking name
            String BookName = nameValidation(BLUE + "Enter New Booking Name      : ");
            int quantity = scanner.nextInt(BLUE + "Enter New Quantity       : ", "The Maximum
quantity of meals we can provide is 1000.Please Enter Your Quantity Again", 1, 1000);
            String phoneNumber = phonenumberValidation(BLUE + "Enter New Phone
Number    : ");

            print.otherMsg("Choose The Meals You Want", 0);
            displayMenuMeal();
            int mealChoose = scanner.nextInt(BLUE + "Enter The Meal You Want   : ",
String.format(RED + "Please enter a number between 1 to %d", MenuList.sizeOf()), 1,
MenuList.sizeOf());
            String MenuName = MenuList.get(mealChoose - 1).getMenuName();
            MenuEntity m = MenuList.firstOrDefault(n ->
n.getMenuName().equalsIgnoreCase(MenuName));
            double amount = m.getPrice() * quantity;

            // confirm want to edit the record
            if (scanner.confirmation(BLUE + "Do you sure you want to edit this record? (Y =
yes / N = no) >> ")) {
                editBooking.setBookName(BookName);
                editBooking.setquantity(quantity);
                editBooking.setPhoneNumber(phoneNumber);
                editBooking.setmeal(MenuName);
                editBooking.setAmount(amount);
                editBook += String.format("| %5s | %20s | %20s |\n", editBooking.getBookID(),
BookName, amount);
                print.success("The record you edit is successful been edited! ");
            } else {
                print.cancelled("Your Edit Request Is Been Cancelled");
            }
        }
    }
  }

  public void searchBooking() {
    print.tableHeader("SEARCH BOOKING", TABLE_WIDTH);
    System.out.println(BLUE + "1. Search Book ID                    |");
    System.out.println(BLUE + "2. Search Booker Name                |");
    System.out.println(BLUE + "3. Search Phone Number               |");
    System.out.println(BLUE + "4. Back To Previous Page             |");  //which user want to
search by
```

```
        print.tableFooter(TABLE_WIDTH);

        int choice = scanner.nextInt(PURPLE + "Please Enter Which You Want To Search By: ",
"Please enter a number between 1 to 5", 1, 5);

        switch (choice) {
            case 1:
                searchBookingId();
                break;
            case 2:
                searchBookingName();
                break;
            case 3:
                searchBookingPhoneNumber();
                break;
            default: {
            }
            break;
        }
    }

    public void searchBookingId() {
        print.tableHeader("SEARCH BOOK ID", TABLE_WIDTH);
        String BookID = scanner.nextLine(BLUE + "Enter book id to search (e.g. B0001): ");

        Booking bk = bookList.firstOrDefault(b -> b.getBookID().equalsIgnoreCase(BookID));

        if (bk != null) {
            searchDisplay(bk, MenuList);
        } else {
            print.failed("Book ID not found");
        }
    }

    public void searchBookingName() {
        print.tableHeader("SEARCH BOOKER NAME", TABLE_WIDTH);
        System.out.println(BLUE + "1. Search Name Starts With");
        System.out.println(BLUE + "2. Search Name Ends With");
        System.out.println(BLUE + "3. Search Name Contains");
        System.out.println(BLUE + "4. Back To Search Main Page");
        print.tableFooter(TABLE_WIDTH);
        int choice = scanner.nextInt(BLUE + "Please Choose Which Ways You Want To Choose
By: ", "Please enter a number between 1 to 4", 1, 4);

        if (choice != 4) {
            final String BookName = scanner.nextLine("Please Enter Name To Search:
").toUpperCase();
            ListInterface<Booking> bList = new DoublyLinkedList<>();
```

```
            switch (choice) {
                case 1:
                    bList = bookList.where(b ->
b.getBookName().toUpperCase().startsWith(BookName));
                    break;
                case 2:
                    bList = bookList.where(b ->
b.getBookName().toUpperCase().endsWith(BookName));
                    break;
                case 3:
                    bList = bookList.where(b ->
b.getBookName().toUpperCase().contains(BookName));
                    break;
                default:
                    break;
            }

            if (bList.isEmpty()) {
                System.out.println(BLUE + "Booker Name not found");
            } else {
                searchDisplay(bList);
            }
        }
    }

    public void searchBookingPhoneNumber() {
        print.tableHeader("SEARCH BY PHONE NUMBER", TABLE_WIDTH);
        System.out.println(BLUE + "1. Phone Number starts with");
        System.out.println(BLUE + "2. Phone Number ends with");
        System.out.println(BLUE + "3. Phone Number contains");
        System.out.println(BLUE + "4. Back to Previous");
        int choice = scanner.nextInt(BLUE + "Please Choose Which Ways You Want To Choose
By: ", "Please enter a number between 1 to 4", 1, 4);

        if (choice != 4) {
            final String BookphoneNumber = scanner.nextLine("Enter phone number to search:
").toUpperCase();
            ListInterface<Booking> bList = new DoublyLinkedList<>();

            switch (choice) {
                case 1:
                    bList = bookList.where(b ->
b.getPhoneNumber().toUpperCase().startsWith(BookphoneNumber));
                    break;
                case 2:
                    bList = bookList.where(b ->
b.getPhoneNumber().toUpperCase().endsWith(BookphoneNumber));
```

```
                break;
            case 3:
                bList = bookList.where(b ->
b.getPhoneNumber().toUpperCase().contains(BookphoneNumber));
                break;
            default:
                break;
        }

        if (bList.isEmpty()) {
            print.failed("Booker phone number not found");
        } else {
            searchDisplay(bList);
        }
    }
  }

  public void sortBookingBy() {
    print.tableHeader("SORT BOOKING RECORD BY", TABLE_WIDTH);

    System.out.println(BLUE + "1. Sort By Book ID                    |");
    System.out.println(BLUE + "2. Sort By Booker Name                |");
    System.out.println(BLUE + "3. Sort By Date                      |");
    System.out.println(BLUE + "4. Back To Previous Page              |");
    print.tableFooter(TABLE_WIDTH);

    int choice = scanner.nextInt("Please Choose Which You Want To Sort By: ", "Please
enter a number between 1 to 6", 1, 6);

    if (choice != 4) {
        print.otherMsg("Sort By List", 1);
        System.out.println(BLUE + "1. Ascending Order");
        System.out.println(BLUE + "2. Descending Order");
        System.out.println(BLUE + "3. Back To Previous Page");

        int sequenceType = scanner.nextInt(BLUE + "Enter Your Choice: ", "Please enter a
number between 1 to 3", 1, 3);

        if (sequenceType != 3) {
            switch (choice) {
                case 1:
                    sortById(sequenceType);
                    break;
                case 2:
                    sortByName(sequenceType);
                    break;
                case 3:
                    sortByDate(sequenceType);
```

```
                    break;
                default: {
                }
                break;
            }
            print.success("Sort successfully, press 6 or 7 to view booking record");
        }
    }
}

public void sortById(int sequenceType) {
    switch (sequenceType) {
        case 1:
            bookList.orderBy((Booking bk1, Booking bk2) -> {
                int sortval = bk1.getBookID().compareToIgnoreCase(bk2.getBookID());

                if (sortval == 0) {
                    return bk1.getBookName().compareToIgnoreCase(bk2.getBookName()) < 0 ?
OrderClause.MOVE_FORWARD : OrderClause.MOVE_BACKWARD;
                } else {
                    return sortval < 0 ? OrderClause.MOVE_FORWARD :
OrderClause.MOVE_BACKWARD;
                }
            });
            break;
        case 2:
            bookList.orderBy((Booking bk1, Booking bk2) -> {
                int sortval = bk1.getBookID().compareToIgnoreCase(bk2.getBookID());

                if (sortval == 0) {
                    return bk1.getBookName().compareToIgnoreCase(bk2.getBookName()) < 0 ?
OrderClause.MOVE_FORWARD : OrderClause.MOVE_BACKWARD;
                } else {
                    return sortval > 0 ? OrderClause.MOVE_FORWARD :
OrderClause.MOVE_BACKWARD;
                }
            });
            break;
        default: {
        }
        break;
    }
}

public void sortByName(int sequenceType) {
    switch (sequenceType) {
        case 1:
            bookList.orderBy((Booking bk1, Booking bk2) -> {
```

```
                int sortval = bk1.getBookName().compareToIgnoreCase(bk2.getBookName());

                if (sortval == 0) {
                    return bk1.getBookName().compareToIgnoreCase(bk2.getBookName()) < 0 ?
OrderClause.MOVE_FORWARD : OrderClause.MOVE_BACKWARD;
                } else {
                    return sortval < 0 ? OrderClause.MOVE_FORWARD :
OrderClause.MOVE_BACKWARD;
                }
            });
            break;
        case 2:
            bookList.orderBy((Booking bk1, Booking bk2) -> {
                int sortval = bk1.getBookName().compareToIgnoreCase(bk2.getBookName());

                if (sortval == 0) {
                    return bk1.getBookName().compareToIgnoreCase(bk2.getBookName()) < 0 ?
OrderClause.MOVE_FORWARD : OrderClause.MOVE_BACKWARD;
                } else {
                    return sortval > 0 ? OrderClause.MOVE_FORWARD :
OrderClause.MOVE_BACKWARD;
                }
            });
            break;
        default: {

        }
    }
  }

  public void sortByDate(int sequenceType) {
    SimpleDateFormat dte = new SimpleDateFormat("dd-MM-yyyy");

    switch (sequenceType) {
        case 1:
            bookList.orderBy((Booking bk1, Booking bk2) -> {
                int sortval = 0;
                try {
                    sortval = dte.parse(bk1.getDate()).compareTo(dte.parse(bk2.getDate()));
                } catch (ParseException e) {
                }

                if (sortval == 0) {
                    return bk1.getBookName().compareToIgnoreCase(bk2.getBookName()) < 0 ?
OrderClause.MOVE_FORWARD : OrderClause.MOVE_BACKWARD;
                } else {
                    return sortval < 0 ? OrderClause.MOVE_FORWARD :
OrderClause.MOVE_BACKWARD;
```

```
                }
            });
            break;
        case 2:
            bookList.orderBy((Booking bk1, Booking bk2) -> {
                int sortval = 0;
                try {
                    sortval = dte.parse(bk1.getDate()).compareTo(dte.parse(bk2.getDate()));
                } catch (ParseException e) {
                }

                if (sortval == 0) {
                    return bk1.getBookName().compareToIgnoreCase(bk2.getBookName()) < 0 ?
OrderClause.MOVE_FORWARD : OrderClause.MOVE_BACKWARD;
                } else {
                    return sortval > 0 ? OrderClause.MOVE_FORWARD :
OrderClause.MOVE_BACKWARD;
                }
            });
            break;
        default: {
        }
        break;
    }
  }

   public void searchDisplay(Booking book, ListInterface<MenuEntity> MenuList) {
// Search for 1 result
      Iterator<MenuEntity> menuItr = MenuList.getIterator();

      while (menuItr.hasNext()) {
         MenuEntity m = menuItr.next();

         if (book.getmeal().compareToIgnoreCase(m.getMenuName()) == 0) {
         }
      }

      print.tableHeader("DONOR SEARCH RESULT", SPECIFY_TABLE_WIDTH);
      System.out.printf(BLUE + "| %-18s | %-20s |\n", "Book ID", book.getBookID());
      System.out.printf(BLUE + "| %-18s | %-20s |\n", "Book Name ", book.getBookName());
      System.out.printf(BLUE + "| %-18s | %-20d |\n", "Quantity", book.getquantity());
      System.out.printf(BLUE + "| %-18s | %-20s |\n", "Phone Number",
String.valueOf(book.getPhoneNumber()).replaceFirst("(\\d{1})(\\d{2})(\\d{3})(\\d+)",
"(+6$1)$2-$3 $4"));
      System.out.printf(BLUE + "| %-18s | %-20s |\n", "Donation Category", book.getmeal());
      System.out.printf(BLUE + "| %-18s | %-20.2f |\n", "Amount", book.getAmount());
      System.out.printf(BLUE + "| %-18s | %-20s |\n", "Date", book.getDate());
      print.tableFooter(SPECIFY_TABLE_WIDTH);
```

```
    }

    public void searchDisplay(ListInterface<Booking> bookList) {      // Search for more than 1
results
        print.tableHeader("BOOKING SEARCH RESULT", LIST_TABLE_WIDTH);
        displayList(bookList, MenuList);
    }

    public void showBookingList() {
        Iterator<MenuEntity> menuItr = MenuList.getIterator();
        print.tableHeader("DISPLAY Booking Record", TABLE_WIDTH);

        int backNum = 1;
        for (int i = 0; menuItr.hasNext(); i++) {
            System.out.println(String.format(BLUE + "%d. Only Display %s Meals", (i + 1),
menuItr.next().getMenuName()));
            backNum++;
        }

        System.out.printf(BLUE + "%d. Back To Previous Main Page\n", backNum);
        print.tableFooter(TABLE_WIDTH);

        int choice = scanner.nextInt(PURPLE + "Enter Your Choice: ", String.format("Please
enter a number between 1 to %d", backNum), 1, backNum);

        if (choice != backNum) {
            int numOf = 0;
            // Get category name
            String menuName = MenuList.get(choice - 1).getMenuName();

            print.tableHeader(String.format("Booking Details (%s)", menuName.toUpperCase()),
LIST_TABLE_WIDTH);

            Iterator<Booking> bookItr = bookList.getIterator();

            System.out.printf(BLUE + "| %-5s | %-20s | %-5s | %-20s | %-15s | %-13s |\n", "ID",
"NAME", "AGE", "PHONE NUMBER", "AMOUNT", "DATE");
            print.tableMiddleLine(LIST_TABLE_WIDTH);
            while (bookItr.hasNext()) {
                Booking bk = bookItr.next();
                if (menuName.compareToIgnoreCase(bk.getmeal()) == 0) {
                    System.out.println(bk);
                    numOf++;
                }
            }

            if (numOf == 0) {
```

```
            print.toCenter(String.format("No Record Found In %s Meal", menuName),
LIST_TABLE_WIDTH);
        }

        print.tableFooter(LIST_TABLE_WIDTH);
        print.otherMsg(String.format("Total Number Of %s Meals: %d", menuName, numOf),
0);
      }
   }

   // Display all booking record
   public void allrecord() {
      print.tableHeader("ALL BOOK RECORD", LIST_TABLE_WIDTH);
      displayList(bookList, MenuList);
      print.otherMsg(String.format("Total Number Of Booking: %d", bookList.sizeOf()), 0);
   }

   // Display booking record by meal
   public void displayList(ListInterface<Booking> bookList, ListInterface<MenuEntity>
MenuList) {
      Iterator<MenuEntity> menuItr = MenuList.getIterator();
      int numOf;

      while (menuItr.hasNext()) {
         MenuEntity m = menuItr.next();
         print.toCenter(RED + m.getMenuName().toUpperCase(), 104);
         print.tableMiddleLine(LIST_TABLE_WIDTH);

         System.out.printf(BLUE + "| %-5s | %-20s | %-5s | %-20s | %-15s | %-10s |\n", "ID",
"NAME", "QUANTITY", "PHONE NUMBER", "AMOUNT", "DATE");
         print.tableMiddleLine(LIST_TABLE_WIDTH);

         Iterator<Booking> bookItr = bookList.getIterator();

         numOf = 0;

         while (bookItr.hasNext()) {
            Booking bk = bookItr.next();
            if (bk.getmeal().compareToIgnoreCase(m.getMenuName()) == 0) {
               System.out.println(bk);
               numOf++;
            }
         }

         if (numOf == 0) {
            print.toCenter("No record for this meal!", LIST_TABLE_WIDTH);
         }
         print.tableFooter(LIST_TABLE_WIDTH);
```

```
        }
    }

    public void displayMenuMeal() {
        Iterator<MenuEntity> menuItr = MenuList.getIterator();

        for (int i = 0; menuItr.hasNext(); i++) {
            System.out.println(String.format(PURPLE + "%d. %s", (i + 1),
menuItr.next().getMenuName()));
        }
    }

    // name sortvalidation
    public String nameValidation(String promptInfo) {
        boolean sortvalidname = false, inputname = true;
        String mainErrorName = "Please enter a sortvalid Booker name (e.g. Jonas)";
        int numChar = 0;
        String BookName;

        do {
            BookName = scanner.nextLine(promptInfo);

            inputname = true;
            for (int i = 0; i < BookName.length(); i++) {
                if (Character.isLetter(BookName.charAt(i)) == false && BookName.charAt(i) != ' ') {
                    inputname = false;                          //to makesure the input name only in char
format
                } else if (Character.isLetter(BookName.charAt(i))) {
                    numChar++;
                }
            }

            if (!inputname) {                                   // if the input name is insortvalid will print error
message
                print.error(mainErrorName);
                print.error("The Booker Name Is Insortvalid Because Contain Special Symbols Or
Numbers");
            } else if (BookName.length() <= 0 || numChar == 0) {
                print.error(mainErrorName);
                print.error("The Booker Name Cannot Be Empty!");
            } else if (BookName.length() >= 20) {
                print.error(mainErrorName);
                print.error("The Booker Name Must Less Than 20 Characters For Record
Purpose! ");
            } else {
                sortvalidname = true;
            }
```

```
        } while (!sortvalidname);

        return BookName;
    }

    public String phonenumberValidation(String promptInfo) {
        boolean sortvalidphone = false;
        String mainErrorPhoneNumber = "Please enter a sortvalid phone number (e.g.
0129431228)";
        String phoneNumber;

        do {
            phoneNumber = scanner.nextLine(promptInfo);

            if (!phoneNumber.matches("[0-9]+")) {
                print.error(mainErrorPhoneNumber);
                print.error("The Phone Number Cannot Contain Characters Or Special Symbols");
            } else if (phoneNumber.charAt(0) != '0' || phoneNumber.charAt(1) != '1') {
                print.error(mainErrorPhoneNumber);
                print.error("The Phone Number Format Must Start With 01!");
            } else if (phoneNumber.length() < 10) {
                print.error(mainErrorPhoneNumber);
                print.error("The Phone Number Cannot Less Than 10 Digits!");
            } else if (phoneNumber.length() > 11) {
                print.error(mainErrorPhoneNumber);
                print.error("The phone number cannot more than 11 Digits!");
            } else {
                sortvalidphone = true;
            }
        } while (!sortvalidphone);

        return phoneNumber;
    }
}
```