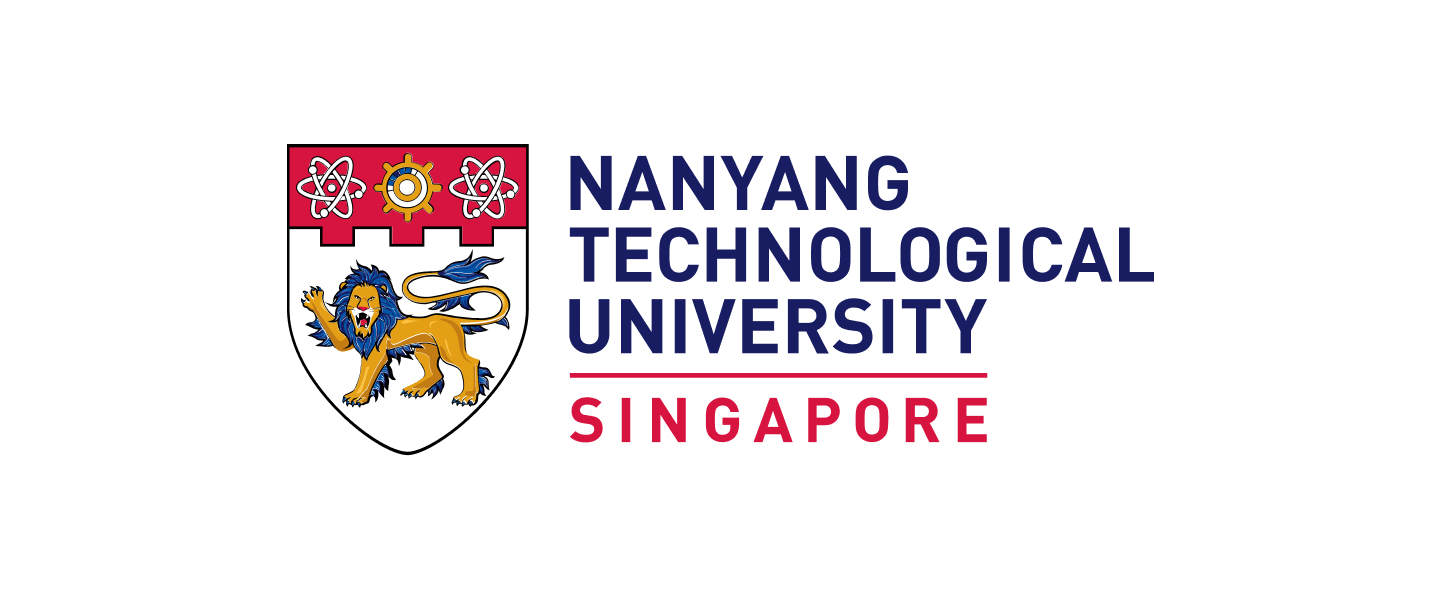
****

**CE/CZ2002: Object-Oriented Design & Programming**

Building an OO Application

(Restaurant Reservation and Point of Sale System)

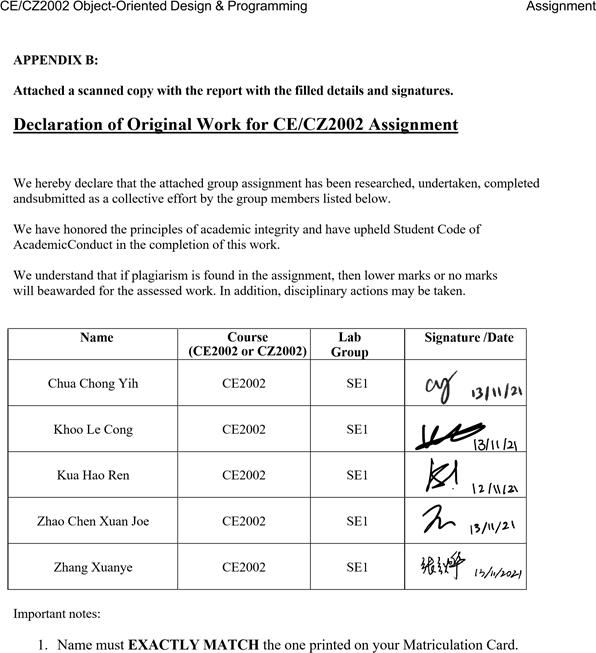
**Done By: SE1 Group 1**

|  |  |
| --- | --- |
| Chua Chong Yih | U2022784B |
| Khoo Le Cong | U2021262C |
| Kua Hao Ren | U2022650C |
| Zhao Chen Xuan Joe | U2020377E |
| Zhang Xuanye | U2022519K |

**2021/2022 SEMESTER 1**

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

* **Declaration of Work**

****

* **Video Deliverable Link**
  + <https://www.youtube.com/watch?v=3ovLJish0yc>
* **Design Consideration**

**Approach**

When we designed our application, we wanted it to retain data even after we closed the application. To achieve this, we had to store our data in a text file and retrieve it when we start our application. This text file can be seen as our pseudo database.

To perform operations on this database file, we needed a class that has the required methods. Our database classes have this functionality where each database can update, store, search and remove entries from our database.

Following the functional requirements, we created the following database classes: MemberDB, StaffDB, MenuDB, OrderDB, ReservationDB, TableDB and SalesRevenueDB. Each database class contains an ArrayList of objects where they defer in the types of objects being stored. For example, MenuDB stores an ArrayList of MenuItem objects and OrderDB stores an ArrayList of Order objects. Each object acts as an entry in the database where they have their own implementations.

We then implemented these database classes and their entry classes according to the functional requirements and have the main RRPSSApp class call the methods of these classes to form our application.

**OO Concept**

* Abstraction

The functional requirement requires an ala carte menu item as well as a promotional set menu item. These items have common features, such as price, name and description that can be abstracted. This was done by creating an abstract superclass MenuItem that is inherited by both PromoSet and AlaCarte classes.

Our goal was to make our program extendable. For example, in the future, if the restaurant were to decide to create a new type of MenuItem, such as MealSet or BuffetSet, it can be easily implemented by extending MenuItem. This forms the basis for our open-closed design principle in the next section.

* Encapsulation and Information Hiding

All variables in our classes are declared as private, including the ArrayList of objects. Furthermore, the ArrayLists are declared as static so there only exists one copy in the application. To access them, mutator and accessor methods are created where necessary.

Most of our methods are declared as public, except for a few methods that are only called by its own class. These methods are declared as private to hide them from other classes. For example, the Invoice class has a private printInvoice method since it should only be called in the Invoice class.

* Method Overloading

In SalesRevenueDB, we have a printRevenue method that iterates through all entries in the database and prints the sale records matching the parameter passed in. Since we should have methods that can print the day’s revenue and also the month’s revenue, we can overload these methods by having different parameters. The overloaded methods are shown below.

public void printRevenue(int month)

public void printRevenue(int day, int month)

The first method prints the revenue according to the month specified and the second method prints the revenue according to the day and month specified.

* Method Overriding

To sort our ArrayList of MenuItem in our Order class, we made use of the sort method in the List interface implemented by the ArrayList class. We then override the compare method in the comparator interface to specify the order to sort our ArrayList. As shown below, we are sorting the MenuItem objects by the name field of our MenuItem object.

orderList.sort(new Comparator<MenuItem>() {

@Override  
 public int compare(MenuItem o1, MenuItem o2) {

return o1.getName().compareTo(o2.getName());

}

});

Another application of method overriding is for the abstract methods in our MenuItem abstract class. The inherited subclasses override these abstract methods with their own implementations. This forms the basis for polymorphism that will be described next. The code segment is shown below.

@Override

public void displayItem() {

System.out.printf(“%-15s %-30s %-50s %.2f\n”, (“[“ +  
 foodType.toString() + “]”), super.getName()

,super.getDescription(), super.getPrice());

}

* Polymorphism

In the MenuDB class, we initialize an ArrayList of MenuItem objects that form our menu. Since AlaCarte and PromoSet are subclasses of MenuItem, they both can be upcasted and added into our ArrayList.

This allows us to perform polymorphism when we iterate through the MenuItem to print their corresponding details. Since AlaCarte and PromoSet have overridden the abstract displayItem method with their own implementations, the details displayed will be according to how the subclasses implemented the abstract method.

for (MenuItem dish : menu) {

if (dish.getClass() == menuClass) {

if (first) {

dish.displayItemDescription();

first = false;

}

dish.displayItem();

}

}

* **Design Principles**

1. Single Responsibility Principle

Each of our classes were designed with one specific role in mind. Database classes are responsible for database operations such as writing and reading to the ArrayList and text file. MenuItem, Order, Sales, Table and Staff are classes that represent an entry in our database. Invoice class is responsible for printing invoice orders.

1. Open-Closed Principle

MenuDB class is fully closed for modification but opened for extensions. All methods and variables in our MenuDB class only work with MenuItem objects. For example,

public void addMenuItem(MenuItem item)

takes in the abstract MenuItem object and adds it into the ArrayList. Printing the menu uses polymorphism to call the subclass’s displayItem method as described above.

By only referring to MenuItem in our MenuDB class, we have essentially closed the MenuDB since future extensions can inherit the MenuItem class. For example, a new subclass of MenuItem such as MealSet can extend MenuItem and be added into the MenuDB’s menu without any modification

* **UML Class Diagram**

Diagram

Description automatically generated

* **UML Sequence Diagram (Check / Remove Reservation)**

**Check Reservation**

**Graphical user interface

Description automatically generated**

**Remove ReservationApplication

Description automatically generated with low confidence**

* **Screenshots of Functional Requirements Testing**

|  |  |
| --- | --- |
| **1. Create / Update / Remove menu item** | |
| Create | Update price    Update description |
| Updated menu item | |

|  |  |
| --- | --- |
| **2. Create / Update / Remove promotion** | |
| Create | Update item in promotion |
| Updated promotion | |

|  |  |
| --- | --- |
| **3. Create Order / 4. View Order / 5. Add & Remove item from order** | |
| Create | Add Item to order    Remove Item from order |
| View order | Confirm order |

|  |  |
| --- | --- |
| **6. Create reservation booking / 7. Check/Remove reservation booking** | |
| Create | Confirm (Auto passes details to order) |
| Check | |

|  |  |
| --- | --- |
| **8. Check table availability** | |
| Check all tables (Allows multiple reservations to a table, allows reservation to occupied table, allows using of reserved table if reserved time is not close to current time) | |
| Check table availability according to time and pax | Before confirming order table status is OCCUPIED with/without reservation    After confirming order, table status changes to RESERVED/AVAILABLE accordingly |

|  |  |
| --- | --- |
| **9. Print order invoice** | |
| Confirm order to print invoice  (Releases table, print invoice) | Invoice  (Invoice generated in invoice folder as txt)  (Includes GST, Service Charge, Member discount) |

|  |  |
| --- | --- |
| **10. Print sales revenue report by period (Today, day, month)** | |
| Today | By day |
| By month | |

* **Additional features**

|  |
| --- |
| **Staff features** |
| Add, Remove, Update Staff |
| **Membership features** |
| Check, add, remove members |
| **Reservation features** |
| Auto checking of membership and passing reservation details to create order when confirming reservation |
| Allows multiple reservation to a table even if it is occupied as long as timing is not close to current occupation  (Assume each walk-in customer/reservation takes 2 hours to dine) |