28 JUNE 2022

University of Dhaka

Department of Computer Science and Engineering

Project Report

Object oriented programming lab (CSE -2112)

Project Name

Smart Restaurant Management System

Team Members

Yeamin Kaiser (01)

Md. Tasnim Bin Anwar (05)

Mohima Ahmed Joyee (42)

Course Instructors

Dr. Muhammad Ibrahim

Md. Ashraful Islam

A. Requirement Analysis

Smart Restaurant Management System is a desktop application that aims to help restaurant owners/managers to manage the restaurant more effectively and efficiently by computerising meal orders of customers, billing and keeping track of financial records. It greatly reduces manual workload by enabling a seamless flow of record keeping by eliminating the possibilities of error while maintaining manual records.

The system processes transactions and stores the resulting data. This will enable easy calculation of daily sales. This will also ensure better communication with the kitchen and wait staff. This system aims to minimise order mishandling as much as possible. The system also provides a seamless way to manage menus and keep the staff up-to-date with the latest ongoings of the restaurant.

The application is coded in Java (back-end) with the help of JavaFX (front-end). A file system is used for handling and storing the necessary data.

This project aims to implement the following modules:

1. **Administration Login**

The admin login requires the owner/manager to enter their username and password to have access to the system and secure every information and transaction done in the system. If someone does not have an existing account, he can create one providing required information.

2. **Recording orders based on customer types**
    a. **Online orders**
        Orders are recorded by taking the customer's name, address and what dishes they would like.
    b. **Offline orders**
        Orders are recorded by taking the customer's name, table number and what dishes they would like.

3. **Updating order status**
    a. **Pending orders**
        Orders are by default set to “pending” mode.
    b. **Delivered orders**
        The manager can update the order status to be delivered as required.

4. **Updating menu**
    a. **Add new items to the menu**
    Add dishes to the menu with each dish having a name, a price, a description and an image.
    b. **Delete items from the menu**
    Delete all information associated with a dish already present in the menu.
    c. **Update price list of already existing items on the menu**
    Change information (name/price/description) of a pre-existing dish on the menu.

5. **Keeping sales records and information**
    a. **Total sales of the day**
    The total amount earned in a day can be viewed. This information ijs auto-updated when an order is placed.
    b. **Cost incurred**
    The cost incurred to produce a dish and deliver it is auto-updated on the Sales Insight Page. Discounts provided are also taken into account.
    c. **Profits earned**
    The profit earned is also auto-updated based on customer types.

B. System Design

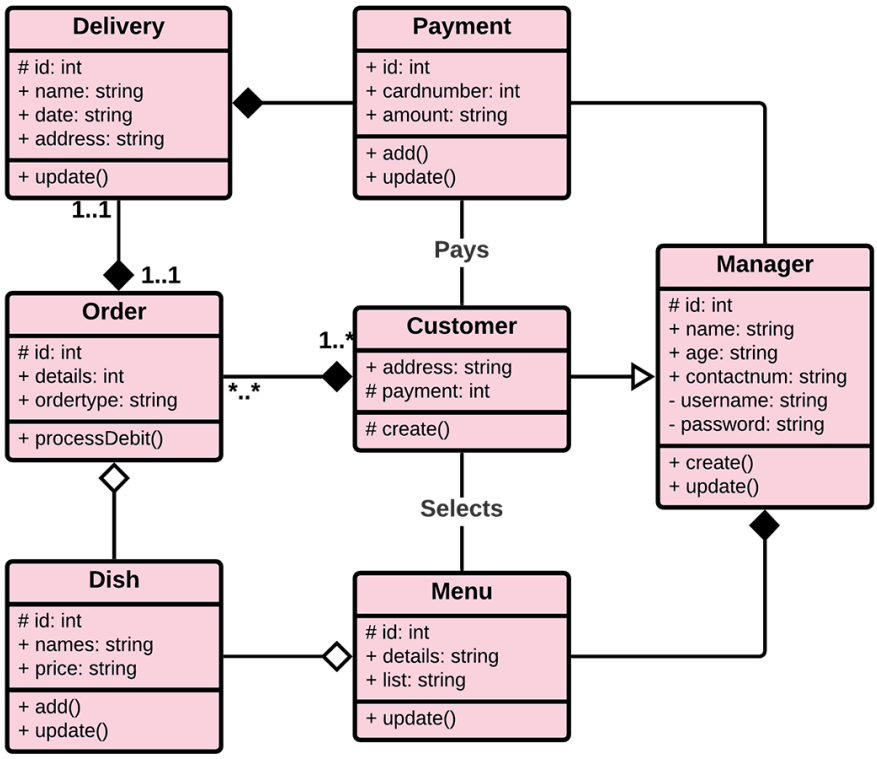

Figure: Class diagram of the system

The system implements several classes and interfaces taking advantage of Java features to provide a seamless flow of work for the user. As this application is coded in Java, it is by default a cross-platform application.

The classes, interfaces and their methods are discussed below:

1. Main Class/HelloApplication Class

The HelloApplication class extends the Application class. It contains the “start” method which takes an object of the “Stage” class of JavaFx as a parameter. This is where the screen that we see is rendered. The Main class launches the application.

2. Manager Class

This is the most important class in the entire program. It encapsulates several methods and variables. It has access to all the other classes. The following methods are implemented in this class:

a. Manager(): This is the constructor. Objects of different classes are created.
b. take_offline_order(): This creates an object of Offline Customer. This offline customer object calls its order method which returns an object of the Order class. This Order object is then added to the Pending Order Collection and All Order Collection by the add_order method.
c. take_online_order(): This method creates an online customer class and performs the order taking activity same as the offline order method.
d. update_order(): Each Order has a private member called State which contains data about whether an order is delivered, prepared or on route to delivery. This method updates that member of each order when required.
e. profit(): Returns a percentage of the total sales as profit.
f. calculate_sales_and_orders_insight(): All variables are initialised to 0. This method iterates over the total number of orders to calculate the total sales of the day, total cost incurred, net profit and net loss based on the type of customer.
g. write_in_data(): Calls the loader class to write the required data in the required file system.

3. Menu Class

This class deals with all the methods required to make the menu section functional.

a. Menu(): This is the constructor. Objects of different classes are created.
b. add_dish(): This method creates an object of Dish class and adds it to the “all dishes” list.
c. remove_dish(): This method removes the specified Dish from the menu by removing it from “all_dishes” list.
d. edit_price_of_dish(): Edits price of a dish by iterating over “all_dishes”.
e. get_price(): Gets the price of the specified dish from “prices” Map.

4. Dish Class

a. Dish(): This is the constructor.
b. getPrice(): This is a getter method that provides the price of the dish.
c. update(): Updates the price of the dish.

5. Order Class

This class implements methods to make the ordering food system work efficiently.

a. Order(): This is the constructor. Variables are initialised here.
b. getState(): Gets state of the object.
c. next_state(): Changes the state.
d. getIsDelivered(): Gets if the order is delivered.
e. getIsOnline(): Gets if the order is online.
f. changeIsOnline(): Changes “isOnline”(a boolean).
g. setCustomer_name(): Sets customer name.
h. getCustomer_name(): Gets customer name.

6. Order Collection Class

This class deals with the number of orders placed and the statuses of the orders.

a. Order_collection(): This is the constructor. It initialises variables and creates new HashMaps for all_orders and pending_orders.
b. add_order(): Increases the count of orders and pending orders. The orders are mapped to integers so that they can be updated later on.
c. update(): When an order is updated, the current state of the order is changed to its next state. If an order is delivered, the count of pending orders is decreased and the pending order is removed.

7. Order Food Interface

This interface is implemented because we wanted to make sure both Online and Offline Customer classes implemented the order function. Otherwise, the customer classes would have lost their functionalities.

8. Offline Customer Class

This class implements the Order_food interface and deals with methods that let us take orders from them.

a. Offline_customer(): This is the constructor. Variables are initialised here.
b. order(): Implements the order method of the “Order_food” interface. Creates an object of “Order” class and calculates the total price of the order and “order_summary” of the object and returns it.

9. Online Customer Class

This class implements the Order_food interface and extends the Offline_customer class. It deals with methods that let us take orders from them.

a. Online_customer(): This is the constructor. Variables are initialised here.

b. order(): Implements the order method of the “Order_food” interface. “super” keyword is used so that the object created here has the same features as the object created for Offline_customer class. The order of the offline customer is overridden.

10. Sales Collection Class

This class keeps track of all the financial ongoings of the day, i.e how much profit is made from different types of customers, etc.

a. Sales_collection(): This is the constructor.
b. update(): This updates “completed_daily_sales” and “pending_sales”.

11. Loader Class

This class takes information from different files and loads it onto our custom-built classes.
Loader(): This is the constructor. It does all the work for the class.

12. Unloader Class

This class takes information from different classes and loads it into our different files.
Unloader(): This is the constructor. It does all the work for the class.

C. Discussion

This section of the report discusses the important features of Java that are implemented in the application.

Data that are generally required by almost all classes are kept public. Information about sales and dishes are controlled by private access modifiers so that they are not changed by mistake. To access and use them, getters and setters are used.

The manager class encapsulates quite a few classes used in our project like the menu, sales overview and order overview, each of which encapsulates smaller classes like dishes and orders.

To ensure reuse of code the Online_customer class inherits the Offline_customer class. They both implement an interface which is Order_food. The implementation of this interface in the Offline_customer class is overridden in the Online_customer class.

As the food ordering process in a restaurant can be quite different each time, there are many overloaded functions and constructors in classes including menu, dishes and manager.

The menu and login credentials are saved after each session in files by using File I/O. Input from these files are loaded into our application during later sessions.

As we did not use databases in our project, we mostly used the Java collection frameworks in a lot of our classes. We used Lists, Maps, Hashmaps in our Order collection, sales collection, menu and many other classes.

Exception handling using try-catch was used whenever there was a possibility of error or unexpected behaviour from the user's end.

D. Conclusion and Future Work

We believe that we were able to create an effective computerised system for a restaurant to handle billing, menu and customer records. Due to the shortage of time, we could not implement a few things we initially planned. There are plenty of scopes for us to increase the functionality of the application.

We plan to implement the following features in the near future:

1. Making a web version of the application.
2. Making an interface for customers.
3. More enhanced and rich UI.
4. Adding a loss calculator.
5. Adding different payment options.
6. Adding a system to convert receipt to PDF and enabling printing option.
7. Implementing a Special Offer system other than the default discount system already implemented.
8. Improving our current loyalty system for customers.

References and Resources:

1. Java - The Complete Reference, 10th Edition, Herbert Schildt
2. Introduction to Java Programming, 10th Edition, Y.Daniel Liang
3. JavaFX documentation (openjfx.io)
4. docs.oracle.com

Github Repository: *https://github.com/Bernini0/Restaurant-Management-System*