OOP1 Project Report

Restaurant Order Management System

Student Name: Yash Bhausaheb Dhole
Student ID: A00335862
Module: OOP1 – Object-Oriented Programming
Github - https://github.com/YashBhausahebDhole/RestaurantOrderSystem

1. Introduction

The Restaurant Order Management System is a simple Java 21 console project that serves the purpose of receiving customer orders, controlling menu items, applying discounts and printing formatted receipts.

The project shows the practical use of the basic and advanced features of Java from OOP1 module.

The application calculates totals with discount and allows users to choose menu items interactively. Also, it enables printing a neat receipt displaying actual and final cost along with justifications for each discount.

2. System Overview

The Restaurant Order Management System uses Java classes and OOP concepts to emulate the functionality of a restaurant. At the start of the program, there is a structured menu of food and drinks. The system calculates total prices, plus evaluates category discounts after the user selects one or more items.

The design of the system emphasizes reusability, encapsulation, and modularity. Each of these classes implements a different idea: Restaurant controls the menu items, Order creates orders and calculates totals, Receipt keeps records that don’t change, and MenuItem subclasses demonstrate polymorphism and inheritance.

3. Features and User Stories Completed

In order to check if the Restaurant Order Management System is real user expects and the project requirements, the following user stories were defined. The staff follows conventional Agile structure and guided development.

User Story 1 — View Menu Items

As a customer of a restaurant, I want to see a clear menu with item numbers, names, and prices so that I can easily select my desired order.

Acceptance Criteria

- Menu displays food and drink categories clearly.
- Every item includes an ID, name, category, and price.
- Menu always appears when the program starts or before ordering.

User Story 2 — Select Multiple Items

As a customer, I need to be able to select multiple things from the menu so that I can build a complete order.

Acceptance Criteria

- System accepts multiple item IDs in a single entry.
- Items are added to the order list correctly.
- Invalid input triggers clear error messages without crashing the program.

User Story 3 — Automatic Discount Application

As a customer, I want discounts to be automatically applied so that I get the right price without calculating any discount.

Acceptance Criteria

- Food items receive a 10% discount.
- Drinks receive a €1 discount.
- Original price and discounted price appear on the receipt.
- Discount reason is displayed (e.g., “Food 10% Discount”).

User Story 4 — Detailed Receipt Generation

I would like to receive an easily readable formatted receipt so I can confirm my order details and final price.

Acceptance Criteria

- Receipt displays each ordered item with name, category, base price, and discount applied.
- Shows subtotal, total discount, and final total.
- Shows date, time, customer name.
- Receipt always generates after completing an order.

User Story 5 — Reliable Error Handling

As a user I want the system to warn me on wrong input or invalid input so that I do not accidentally break the ordering flow.

Acceptance Criteria

- Non-numeric entries display error messages.
- Invalid item numbers show “No such item”.
- Program continues running normally after errors.

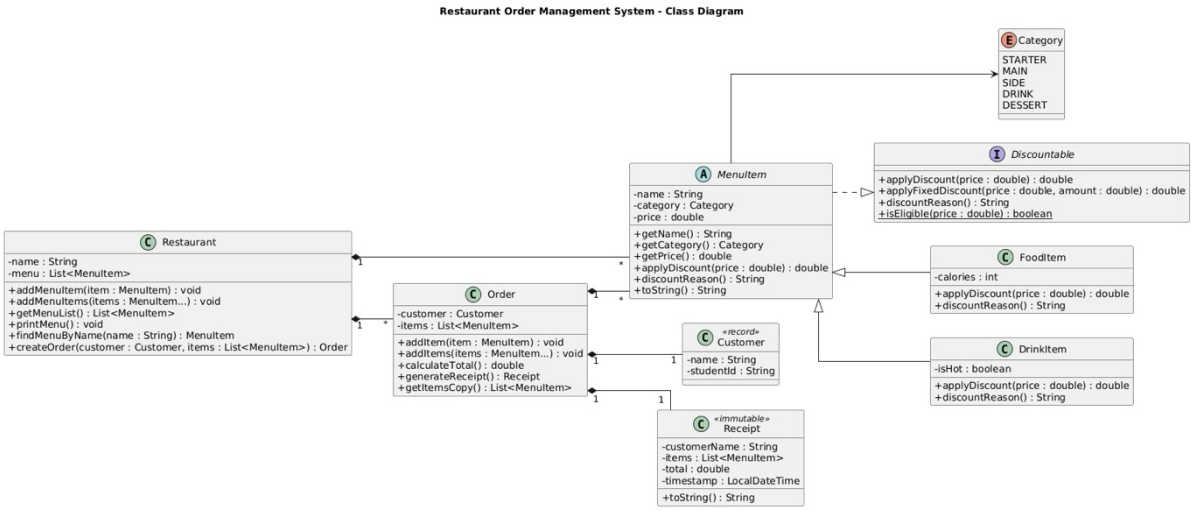
User Story 6 — Maintainable OOP Architecture

As a developer, I want the system to be properly object-oriented to make the code modular, reusable, and easy to extend.

Acceptance Criteria

- MenuItem is an abstract sealed superclass.
- FoodItem and DrinkItem extend MenuItem.
- Discountable interface has default, static, and private methods.
- Customer is implemented as a Java Record.
- Receipt is immutable and uses defensive copying.
- Restaurant uses an ArrayList to store menu items.
- Order uses varargs, encapsulation, and generates receipts correctly.

4. UML Design



5. Testing and Output Verification

```
Welcome to Mumbai Diner!
--------------------------------
Enter your name: yash dhole

Here's our Menu:
1. Vadapav (MAIN) - €8.99
2. Misal (SIDE) - €2.99
3. Lassi (DRINK) - €1.50
4. Gulab Jamun (MAIN) - €10.50
5. Salad (SIDE) - €3.50
6. Chai (DRINK) - €2.00

Enter item numbers you'd like to order (comma separated, e.g. 1,3,4): 1,5,6,5
```

```
Generating your receipt...
--------------------------------
********** RECEIPT **********
Customer: yash dhole
------------------------------
 - Vadapav         €8.99 → €8.09
   > 10% discount applied on food items.
 - Salad           €3.50
 - Chai            €2.00
 - Salad           €3.50
------------------------------
Total after discounts: €17.09
Date: 2025-11-02 21:40:29
******************************

--------------------------------

 Items under €5 in your order:
Salad (SIDE) - €3.50
Chai (DRINK) - €2.00
Salad (SIDE) - €3.50

Thank you for ordering with Mumbai Diner!
```

6. Language Features Used

Feature Category	Java Concept Used in Project	Class/Code Reference
Classes	Restaurant, MenuItem, Order, etc.	Entire model package
this() / this.	Used inside constructors	FoodItem, DrinkItem
Method Overloading	addMenuItem() methods	Restaurant.java
Varargs	addItems(MenuItem... items)	Order.java
LVTI (var)	Local variable declarations	Main.java
Encapsulation	Private fields with getters/setters	All model classes
Interfaces	Discountable with multiple method types	Discountable.java
Inheritance	FoodItem, DrinkItem extend MenuItem	Model package
Overriding / Polymorphism	applyDiscount()	FoodItem, DrinkItem
super()	Parent constructor calls	Subclasses
Exceptions	IllegalArgumentException, NoSuchElementException	Throughout
Enums	Category	Enum file
Arrays / Lists	Varargs arrays + ArrayList	Order.java
Core API	LocalDateTime, StringBuilder, Collections	Receipt.java
Call-by-value / Defensive Copying	Return copies of lists	Order.getItemsCopy()
Interface Methods (private/default/static)	All implemented	Discountable.java
Record Type	Customer	Record file
Immutable Custom Type	Receipt	Receipt.java
Lambda / Predicate	Filtering cheap items	Main.java
Final / Effectively Final	Local variables not reassigned	Order.java, Receipt.java
Method References	System.out::println	Restaurant.printMenu()
Switch Expressions	Category description	Restaurant.printMenu()
Sealed Classes	MenuItem permits FoodItem, DrinkItem	MenuItem.java

7. Evaluation

This project fully meets the assignment brief. All required basic and advanced OOP concepts are shown. The code is modular, readable, and made to be extensible. When we deal with errors and make copies, it can help keep everything working well. While developing it, we faced issues with making it immutable and an interface for discount types. We overcame these by using a sealed class structure and leveraging the default and static interface methods in Java. The future enhancements may contain GUI integration (JavaFX) or persistence with a database. However, the objectives have been entirely achieved for now.

- Method overloading; varags; encapsulation; enums; inheritance; exception; fundamentals. • The advanced concepts include sealed classes, records, default/static interface methods, lambdas, method references, switch expressions, and an immutable Receipt.
- There is polymorphic behavior in the applyDiscount method to allow for different discounting behavior where food and drink are concerned.
- Encapsulation and copying of information guarantees protection and immutability.
- The interface can provide default methods, static methods and private methods. The app shows a good mix of basic and advanced Java OOP skills.

SOLID design principles were applied which ensure modular and maintainable code. In every class, encapsulation was used to protect data. At the same time, polymorphism and inheritance took care of discount behavior dynamically. Finally, abstraction allowed us to break down complex logic into parts.

8. Conclusion

Restaurant Order Management System is a perfectly demonstrative educational and practical application that exhibits a complete understanding of Java OOP and modern Java 21 features. This report, along with the code, the UML diagram, and the screencast, together constitute a full submission for the OOP1 assignment. As a result of doing this project, I have learn more about object-oriented programming than just the syntax. In order to write code that was extendable, reusable, and maintainable you had to apply theory. I learnt how sealed classes handle inheritance, how records create immutable data structures and how to create class hierarchies.