DELIVERING SYSTEM ”DELIVO”

1. Application Overview

Food Ordering System is a **console-based Java application** that simulates the process of ordering food in an online restaurant.

The system supports **two roles**:

- **User** – can browse menu, create orders, customize food, and view order history.
- **Admin** – can manage and update order statuses.

The project focuses on **object-oriented programming** and the practical use of **design patterns**.

2. Core Features

User Features

- View food menu
- Preview food description and price
- Customize food with extras (cheese, spicy, large size)
- Add food to cart
- Place orders
- View personal order history
- Receive notifications when order status changes

Admin Features

- View all orders
- Update order status (Placed, Cooking, Ready, Delivered)

3. Technologies Used

- Java
- Console-based interface
- `ArrayList` and `HashMap`
- Object-Oriented Programming (OOP)
- Design Patterns

4. Design Patterns Used

4.1 Factory Method Pattern

Where used:
`FoodFactory` and `SimpleFoodFactory`

Description:
The Factory Method pattern is used to create different food objects such as:

- Pizza
- Burger
- Fries
- Cola
- Coffee

The main program does not know how food objects are created.
It only calls `createFood(type)`.

Why used:

- Makes the system easy to extend
- New food types can be added without changing existing logic
- Follows the Open/Closed Principle

4.2 Decorator Pattern

Where used:
`FoodDecorator`, `CheeseDecorator`, `SpicyDecorator`, `LargeDecorator`

Description:
The Decorator pattern allows food items to be customized dynamically at runtime.

Extras:

- Cheese
- Spicy
- Large size

Each decorator:

- Wraps a `Food` object
- Modifies description and price

Why used:

- Avoids creating many subclasses for every food-extra combination
- Provides flexible customization
- Keeps code clean and scalable

4.3 Observer Pattern

Where used:
`OrderObserver`, `User`, `OrderManager`

Description:
The Observer pattern is used for **order notifications**.

- Users subscribe to `OrderManager`
- When an order is created or its status changes, all subscribed users receive a notification

Why used:

- Loose coupling between orders and users
- Automatic notification system
- Easy to add more observers in the future

5. SOLID Principles

- **Single Responsibility Principle**
  Each class has one responsibility (Food, Order, Cart, OrderManager).
- **Open/Closed Principle**
  New foods or extras can be added without modifying existing code.
- **Liskov Substitution Principle**
  Decorated food objects can replace base food objects.
- **Interface Segregation Principle**
  Small interfaces like `Food` and `OrderObserver` are used.
- **Dependency Inversion Principle**
  High-level logic depends on interfaces, not concrete classes.

6. System Architecture

Main Components

- **Main** – application entry point and menu navigation
- **FoodFactory** – creates food objects

- **Food** – base interface for all food items
- **Decorators** – extend food functionality
- **Cart** – stores selected food items
- **Order** – represents a user order
- **OrderManager** – manages orders and notifications
- **User** – represents customer and observer
- **Admin** – manages order status

7. User Guide

How to Run

1. Compile: `javac Main.java`
2. Run: `java Main`

How to Use

1. Choose role (User or Admin)
2. User:
    - Browse menu
    - Create order
    - Add extras
    - Checkout
    - View orders
3. Admin:
    - View all orders
    - Update order status

8. Conclusion

Food Ordering System demonstrates how **Factory Method**, **Decorator**, and **Observer** patterns can be used together in a real-world scenario.

The system is:

- Easy to understand
- Easy to extend
- Well-structured
- Suitable for educational purposes