CSCI-C 322 Object-Oriented Software Methods

Members: **Heer Soni, Joey Myers, Sindhur Bansal, Dylan Drotts**

Food Ordering and Delivery System

Abstract

This project presents a comprehensive Java-based Food Ordering and Delivery System designed using multiple object-oriented design patterns. The system provides users the ability to place customized food orders, select a preferred delivery method (Bike, Car, Drone), and track the entire order lifecycle in real time. Emphasis was placed on scalability, maintainability, flexibility, and user interaction. Through the integration of Command, Strategy, Factory, Decorator, and State patterns, the project showcases a robust application of core object-oriented principles. A graphical user interface (GUI) was implemented using Java Swing, which added an intuitive and interactive front-end layer to the project, earning bonus points according to the project requirements.

Introduction

Modern food delivery systems require dynamic customization, efficient order processing, and reliable delivery methods. Our project aims to simulate such a system by utilizing key object-oriented design patterns to structure the application in a modular and extensible way. Users can create complex food orders by combining pizzas, salads, and drinks with customizable toppings and sizes. Delivery strategies offer flexible delivery options. Furthermore, orders transition through a complete lifecycle, simulating real-world food delivery experiences.

The system is designed to be scalable, allowing for the addition of new food types, toppings, and delivery strategies with minimal changes to existing code. This approach significantly improves the maintainability of the application.

System Design Overview

- **GUI (Swing)**: Built using Java Swing, it captures user inputs such as food specification, drop-off location, and delivery method. It displays output including order summaries, estimated delivery times, and costs.
- **Order Management**: Utilizing the Command pattern, the system handles placing, canceling, reordering, and undoing orders effectively.
- **Food Factory & Decorators**: The Factory pattern is used to create basic food types (Pizza, Salad, Drink). The Decorator pattern enhances these food items with additional toppings, modifications, or size adjustments.
- **Delivery Strategies**: Strategy pattern implementation allows dynamic selection between Bike, Car, and Drone delivery, each with unique cost and ETA computations.
- **State Management**: Orders transition sequentially through Confirmed, Picking Up, En Route, and Delivered states, showcasing the State pattern's effectiveness in managing object state transitions.

The interaction between these components ensures the software is loosely coupled and highly cohesive, allowing future scalability.

Patterns Implemented

- **Command Pattern**: Centralized order-related actions (place, cancel, reorder) with easy rollback capabilities using PlaceOrderCommand, CancelOrderCommand, ReorderCommand, and ProcessOrderCommand.
- **Strategy Pattern**: Enabled flexible delivery methods by implementing BikeDeliveryStrategy, CarDeliveryStrategy, and DroneDeliveryStrategy.
- **Decorator Pattern**: Allowed users to dynamically customize food items by layering toppings, size adjustments, and special modifications.
- **Factory Pattern**: FoodFactory centralized object creation, abstracting the instantiation and decoration of food items based on user input.
- **State Pattern** (Bonus): Managed order status transitions through states such as Confirmed, Picking Up, En Route, and Delivered, enhancing clarity and responsibility delegation in order tracking.

Challenges Encountered

- **Pattern Integration**: Combining multiple design patterns without creating tight coupling posed significant challenges. We had to redesign certain classes to adhere to solid principles (especially Dependency Inversion and Single Responsibility).
- **Undo Functionality**: Supporting undo for complex operations, especially in ProcessOrderCommand which aggregates multiple steps (order placement, delivery

execution, and state transitions), required maintaining an accurate command history and rollback logic.

- **GUI Synchronization**: Keeping the GUI state synchronized with backend operations, particularly when orders were canceled or undone, demanded careful event handling and Swing thread management.
- **Testing and Debugging**: Verifying the correct behavior of composite decorated food items, especially under invalid or unexpected user inputs, led to rigorous edge-case testing and improved input validation.

Conclusion

The Food Ordering and Delivery System successfully achieves the project objectives of applying multiple object-oriented design patterns in a cohesive, real-world application. The modular structure and flexible architecture ensure easy extensibility for future enhancements, such as real-time GPS tracking, additional food categories, or new delivery strategies.

Moreover, implementing a fully functioning Java Swing GUI elevated the project's usability and made the system more intuitive and engaging for users. This project provided invaluable hands-on experience in designing scalable systems, integrating multiple design patterns, and adhering to industry-level coding standards.

Contributions

- **Heer Soni**: Developed Factory Pattern, contributed significantly to the report and presentation slides, worked on UMLs.
- **Joey Myers**: Implemented Decorator Pattern, and integrated it with factory pattern to ensure they work together.
- **Sindhur Bansal**: Developed Strategy and State Patterns, Edited Order class to support the stages of order cycle, Updated GUI.
- **Dylan Drotts**:: Developed the Command Pattern for placing, canceling, reordering and undoing food orders, Worked on GUI