Cairo University
Faculty of Computers and Artificial Intelligence

Advanced Software Engineering

Documentation file

Team Members

ID	Name	Email	Mobile
20230542	Ahmed sheref sayed	20230542@stud.fci-cu.edu.eg	01091575793
20231142	Mohammed Sheref Abd-Alazim	20231142@stud.fci-cu.edu.eg	01150600775

1. Overall Architecture

The application is organized into cohesive subsystems: menu creation, add-ons, discounts, payments, kitchen notifications, and a facade that coordinates all components. Each subsystem follows a clear design pattern to keep responsibilities separated and avoid unintended changes across the system.

---

2. Abstract Factory for Menus

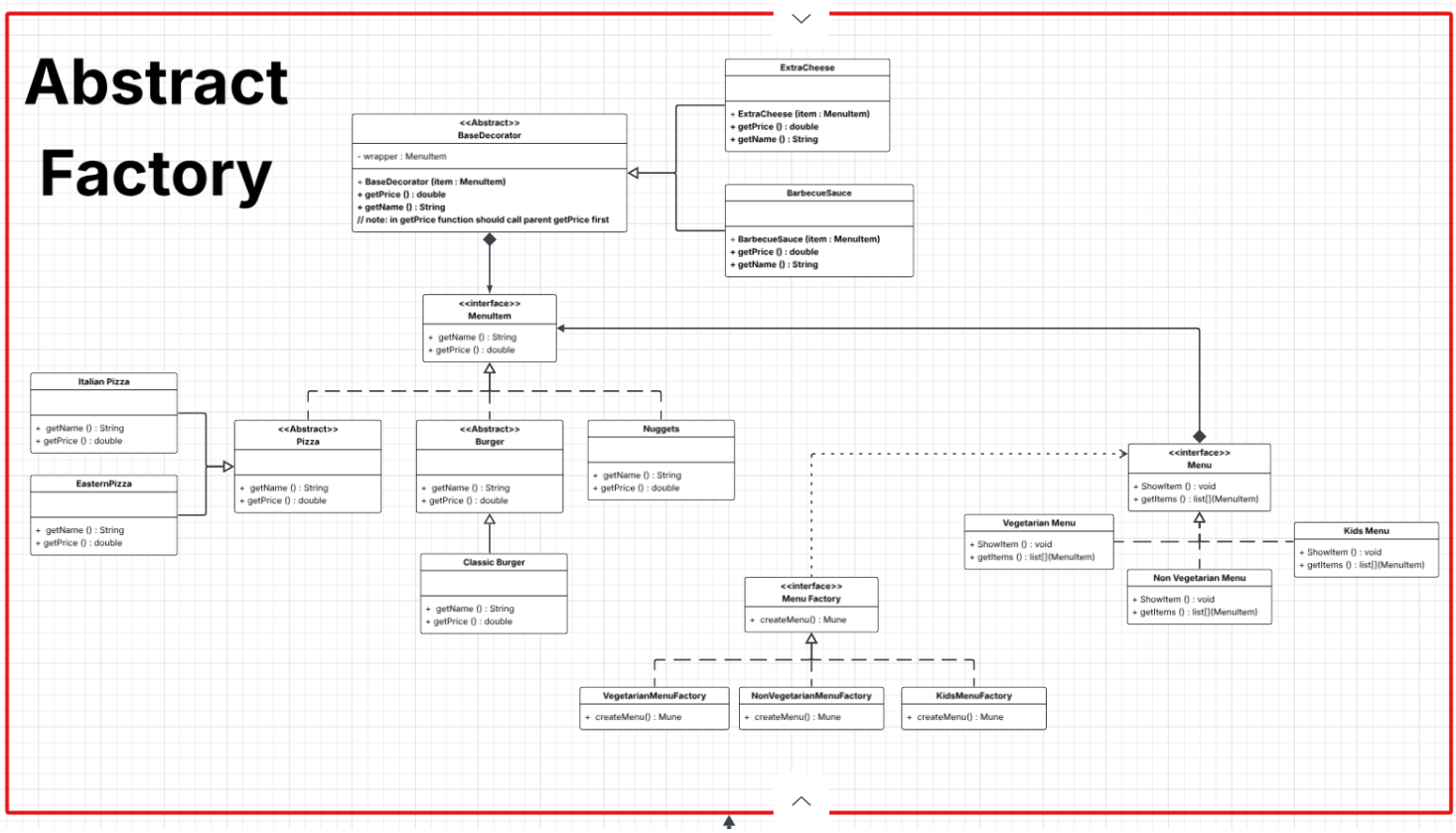
- **Problem:** The system must support multiple menu categories (Vegetarian, Non-Vegetarian, Kids), each containing specific items. Adding new menus should not require modifying existing logic.
- **Decision:** Use the **Abstract Factory** pattern.
- **Implementation:**
    - MenuFactory defines createMenu().
    - VegetarianMenuFactory, NonVegetarianMenuFactory, and KidsMenuFactory each produce their own menu.
- **Benefit:** The main() function only interacts with Menu and MenuItem interfaces. New menus can be added easily without editing existing code.

---

3. Decorator for Add-Ons

- **Problem:** Items may include multiple customizable add-ons (e.g., Extra Cheese, Barbecue Sauce). Combinations should be flexible without creating many subclasses.
- **Decision:** Use the **Decorator** pattern.
- **Implementation:**
    - BaseDecorator wraps a MenuItem.
    - Decorator classes modify behavior of getName() and getPrice().
- **Benefit:** Add-ons can be chained in any order. No need for subclasses like Pizza With Cheese And BBQ.

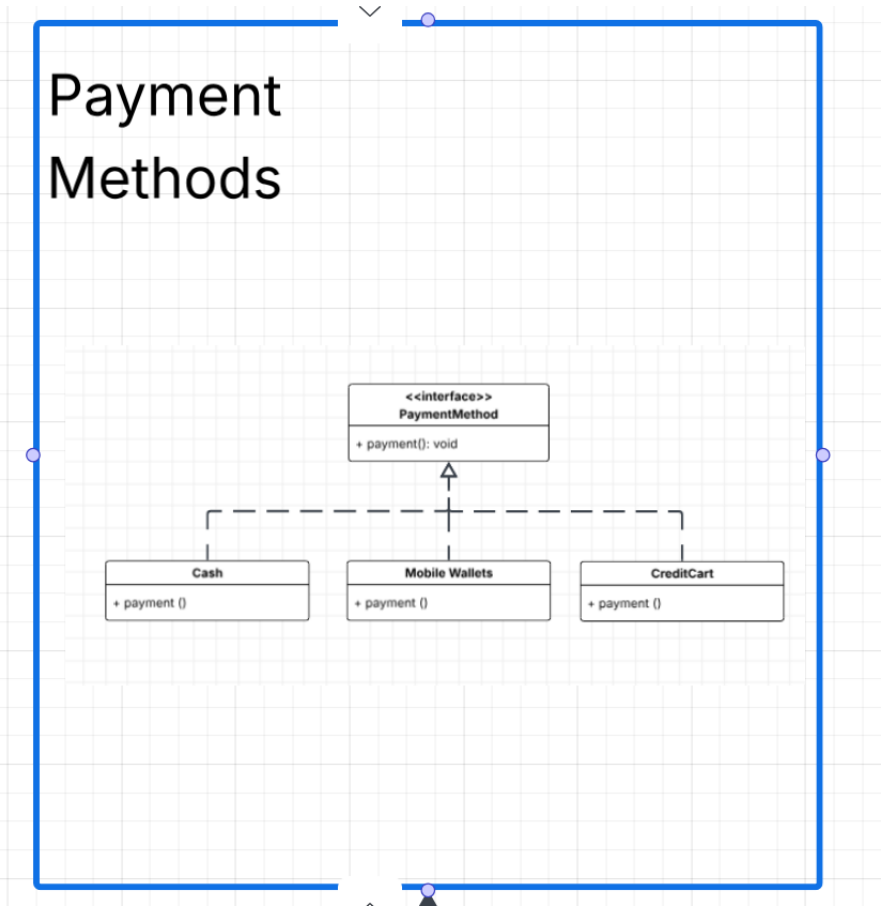
First Subsystem UML (Abstract Factory and Decorator):



4. Strategy for Payments

- **Problem:** Each order can be paid using different methods (cash, wallet, credit card). The order should not be tightly coupled to payment types.
- **Decision:** Use the **Strategy** pattern.
- **Implementation:**
    - PaymentMethod interface defines payment(amount).
    - Concrete classes implement the behavior.
- **Benefit:** Order.pay() doesn't need to change when adding new payment methods.

  Second Subsystem UML (Payment Strategy) :

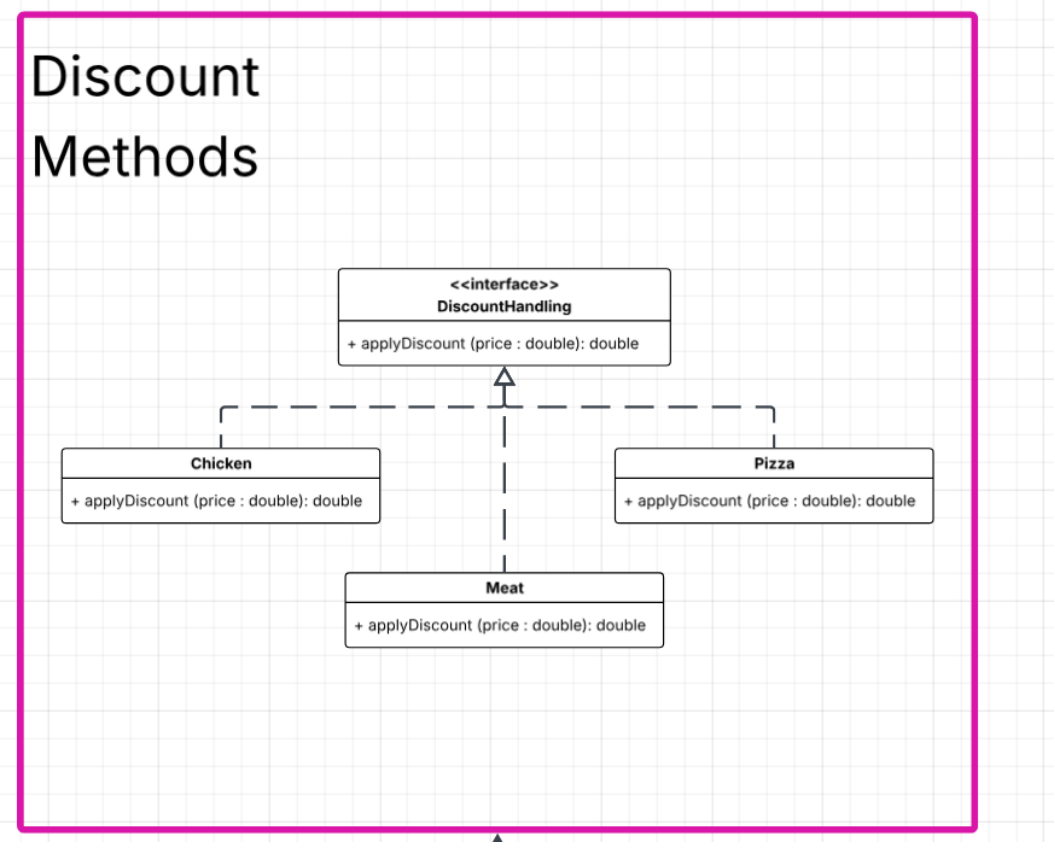


Note: A payment context is unnecessary in this design, as the Order class interacts directly with the selected payment strategy.

5. Strategy for Discounts (Per-Item)

- **Problem:** Different food categories require different discount rules. Users should not choose discounts manually. Orders may contain mixed categories.
- **Decision:** Use the **Strategy** pattern + automatic discount selection.
- **Implementation:**
    - DiscountHandling interface defines applyDiscount(priceOfItem).
    - DiscountSelector.fromItem(item) returns the discount strategy based on the item's category.
    - Order.pay() applies discounts per item.
- **Benefit:** Every item receives its correct discount without user involvement.

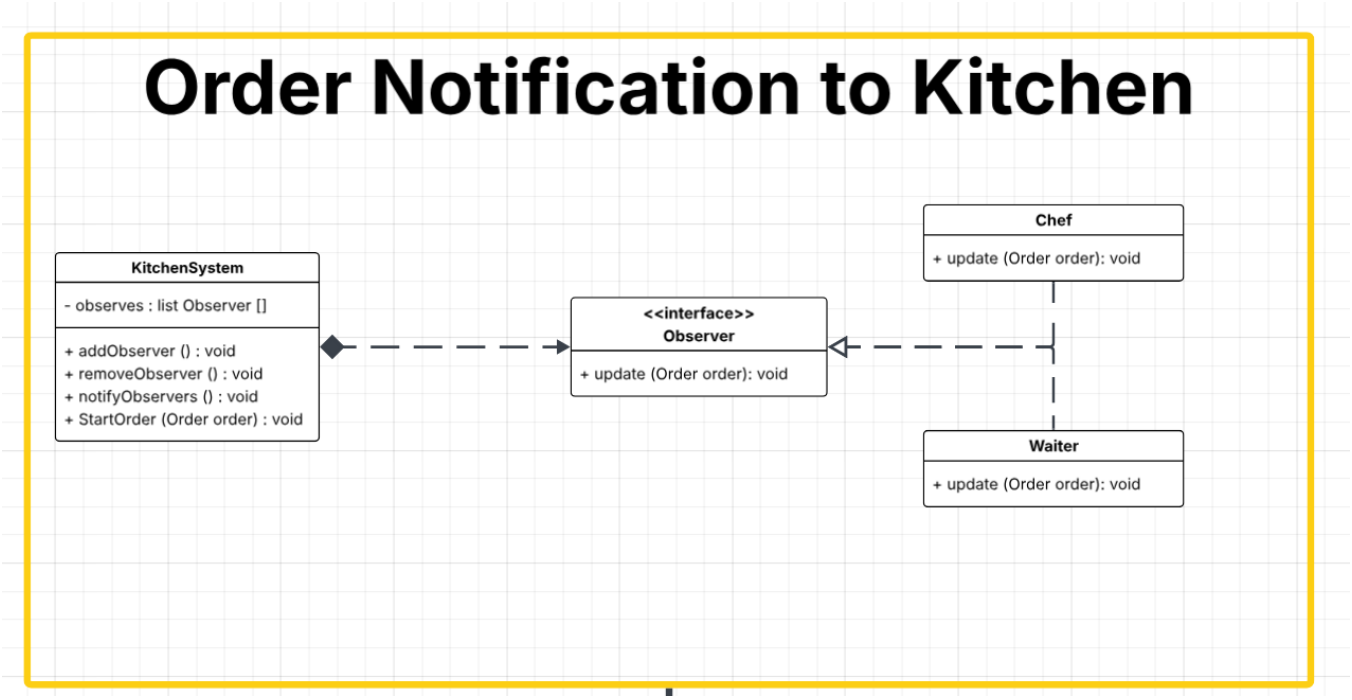
  Third Subsystem UML (Discounts Strategy) :



6. Observer for Kitchen Notifications

- **Problem:** The system must notify different roles (chef, waiter) when an order starts.
- **Decision:** Use the **Observer** pattern.
- **Implementation:**
  - KitchenSystem holds observers.
  - Chef and Waiter implement update(Order).
- **Benefit:** New observers can be added (e.g., delivery staff) without modifying existing code.

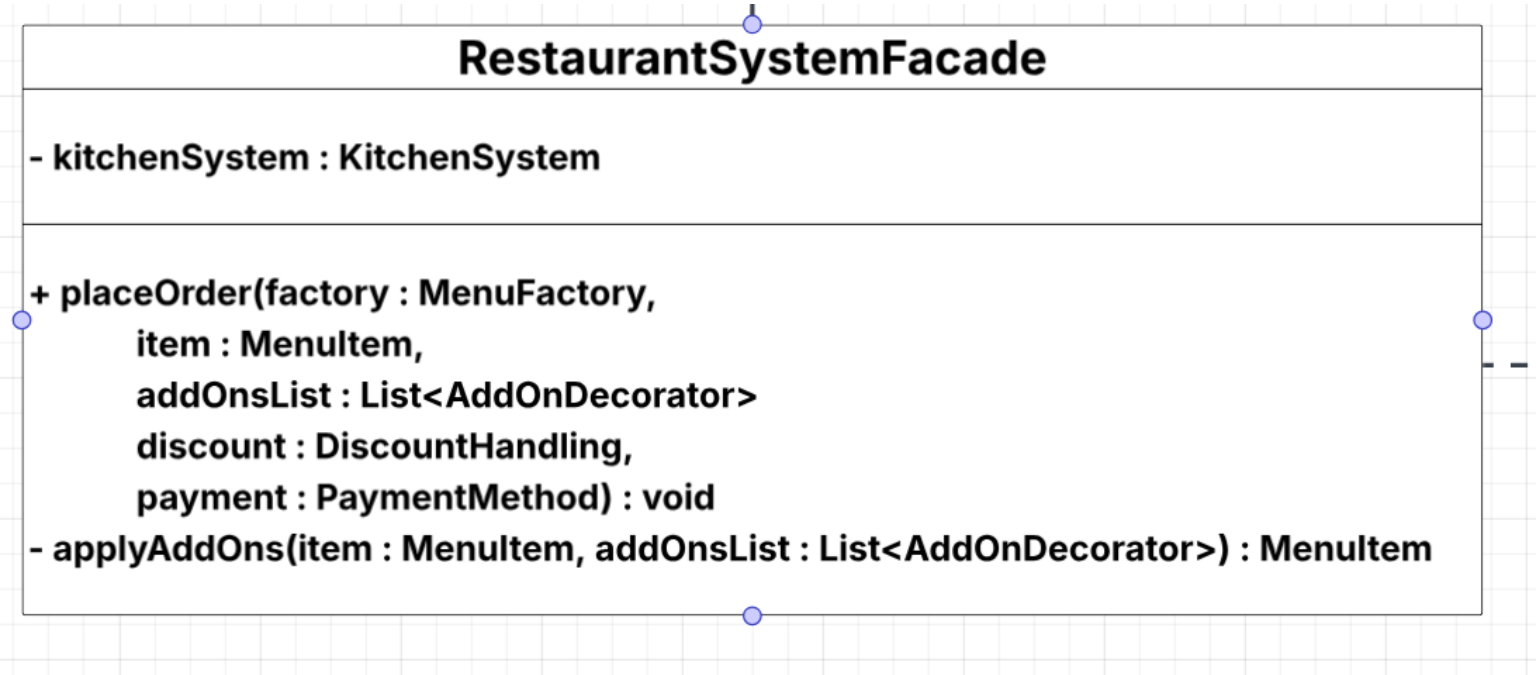
Fourth Subsystem UML (**Observer for Kitchen Notifications**)



7. Facade to Coordinate the Flow

- **Problem:** The main() function should not handle all internal steps like applying add-ons, selecting payment, notifying kitchen, and calculating the bill.
- **Decision:** Use the **Facade** pattern.
- **Implementation:**
    - RestaurantSystemFacade provides placeOrder() and simplifies interactions.
- **Benefit:** The application logic stays clean and readable.

Fifth Subsystem UML (Façade) :



8. Number-Based User Input

- **Problem:** Allowing users to type item names causes errors.
- **Decision:** Use numeric selection for menus and add-ons.
- **Benefit:** Prevents mistakes and improves usability.

The all System UML: