Lab 7 - Design Patterns

Course: Object-Oriented Analysis and Design (Code: IT090IU)
HCMIU-CSE, Fall 2025
Instructors: Le Thi Ngoc Hanh

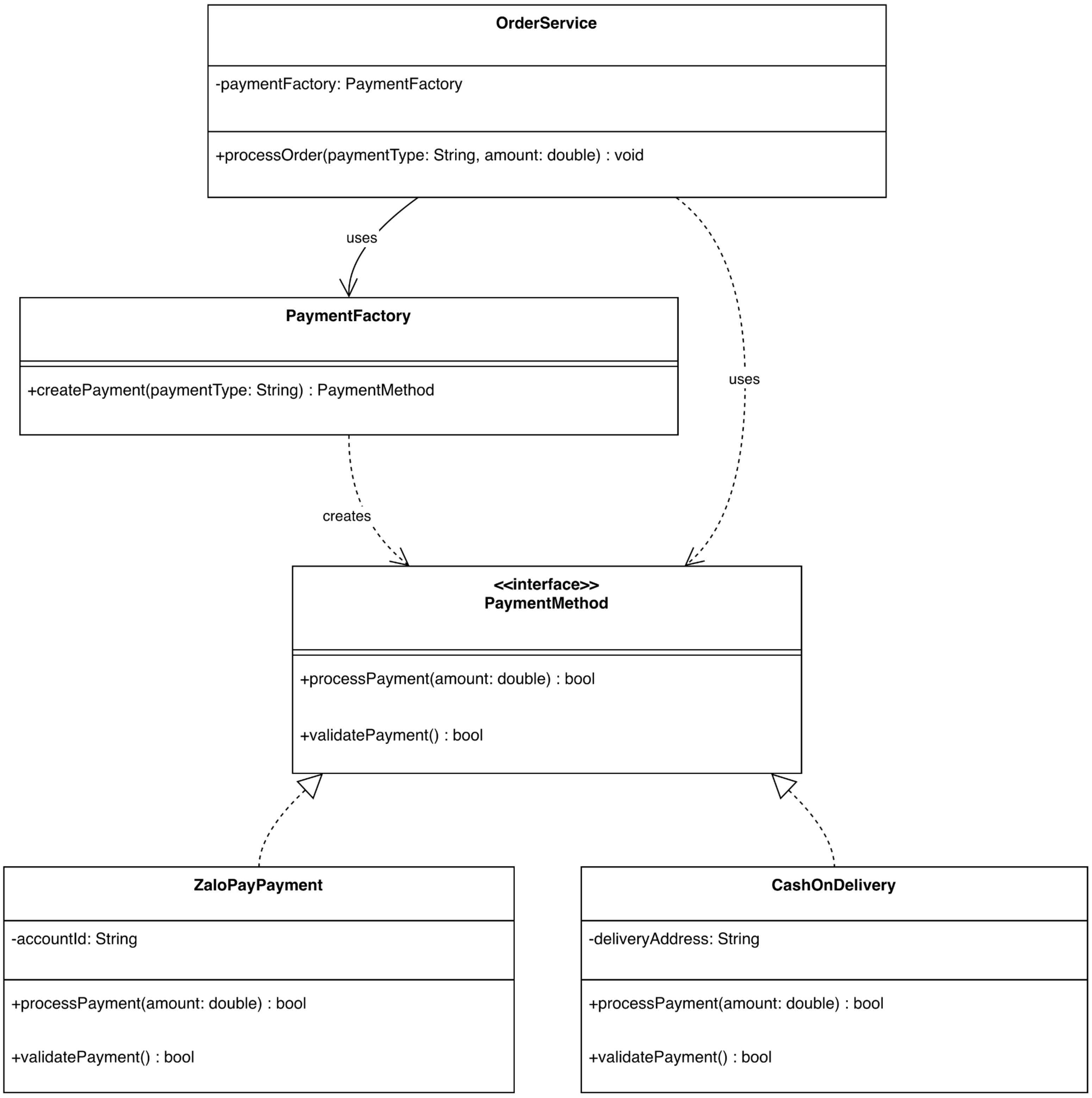
Student name: Nguyễn Đức Thành Công Student ID: ITCSIU22023
Student name: Phạm Tuấn Kiệt Student ID: ITCSIU22235

The food delivery app has the description as follows.
In a food delivery app, customers can search for restaurants by cuisine, place orders, and track their delivery in real-time. The system supports multiple payment options, promotions, and customer reviews. Restaurants can update their menus, monitor order statuses, and communicate with shippers. Shippers receive delivery assignments, navigate to pickup locations, and update delivery statuses. Additionally, the system manages failed deliveries (e.g., when a customer is unavailable) and processes refunds or credits for canceled orders.
Consider the process of placing and delivering a food order in this food delivery app. After placing an order, the system checks if the selected restaurant is currently accepting orders. If the restaurant is closed, the customer is notified and asked to either change the time or cancel the order. Once the order is accepted, the payment is processed. The system concurrently handles the preparation of the order at the restaurant and assigns a shipper. After both are complete, the order is dispatched to the customer.

Hypothetical scenario:
The food delivery app now supports multiple payment methods, including ZaloPay, cash on delivery (COD), and potentially more in the future (e.g., e-wallets). To accommodate this, the system must be designed to instantiate the appropriate payment method dynamically, based on the customer's selection during checkout. You are tasked with refactoring the design using the Factory Design Pattern to support this extensibility.

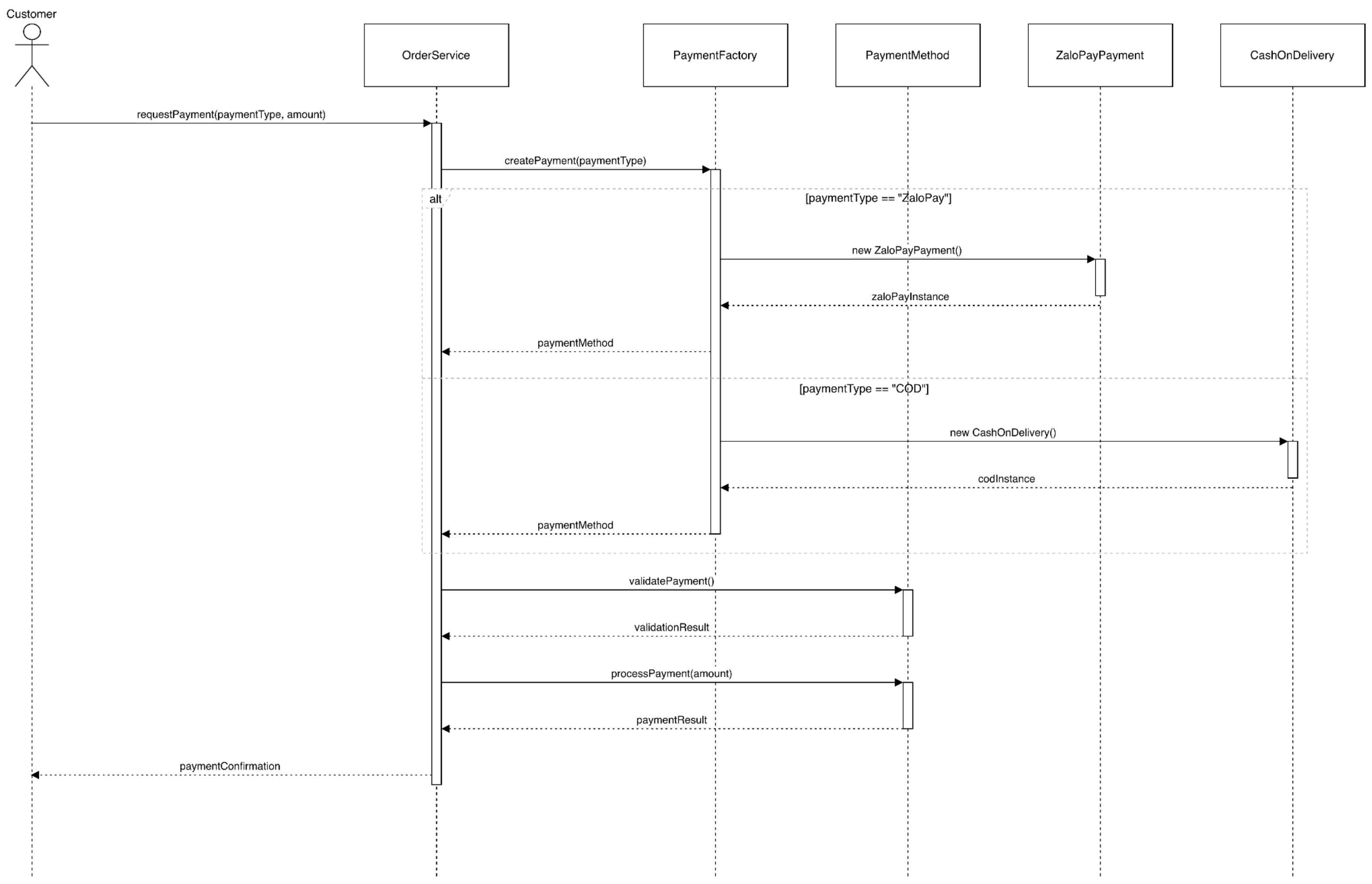
Q1.

- **PaymentMethod** interface with common payment operations
- Two concrete implementations: **ZaloPayPayment** and **CashOnDelivery**
- **PaymentFactory** class that creates the appropriate payment method
- **OrderService** that uses the factory to process payments


OrderService
-paymentFactory: PaymentFactory
+processOrder(paymentType: String, amount: double) : void
uses
uses
PaymentFactory
+createPayment(paymentType: String) : PaymentMethod
creates
<<interface>>
PaymentMethod
+processPayment(amount: double) : bool
+validatePayment() : bool
ZaloPayPayment
-accountId: String
+processPayment(amount: double) : bool
+validatePayment() : bool
CashOnDelivery
-deliveryAddress: String
+processPayment(amount: double) : bool
+validatePayment() : bool

Q2.

1. Customer requests payment through **OrderService**
2. **OrderService** calls **PaymentFactory** to create the correct payment method
3. Factory instantiates either **ZaloPayPayment** or **CashOnDelivery** based on the type
4. The selected **PaymentMethod** validates and processes the payment
5. Result is returned to the customer


Customer
OrderService
PaymentFactory
PaymentMethod
ZaloPayPayment
CashOnDelivery
requestPayment(paymentType, amount)
createPayment(paymentType)
alt
[paymentType == "ZaloPay"]
new ZaloPayPayment()
zaloPayInstance
paymentMethod
[paymentType == "COD"]
new CashOnDelivery()
codInstance
paymentMethod
validatePayment()
validationResult
processPayment(amount)
paymentResult
paymentConfirmation