**เรื่อง LuggagePricingSystem:**

* การรวม LuggagePricingSystem เข้ากับ Luggage อาจไม่เหมาะสมตามหลัก Single Responsibility Principle (SRP) เพราะ:
* Luggage ควรรับผิดชอบเฉพาะข้อมูลกระเป๋า
* LuggagePricingSystem รับผิดชอบการคำนวณราคา ซึ่งอาจมีกฎเกณฑ์ที่ซับซ้อนและเปลี่ยนแปลงได้

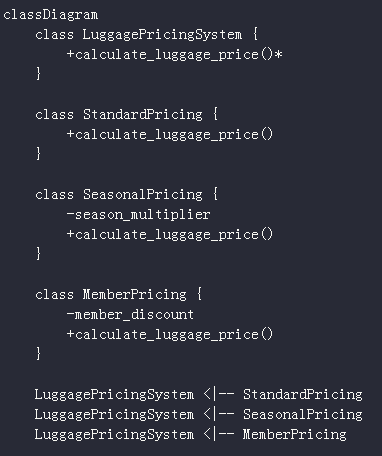
ทำไมไม่ควรรวม LuggagePricingSystem เข้ากับ Luggage ตามหลัก SOLID Principles:

1. **Single Responsibility Principle (SRP)**

* **Luggage Class**:
* รับผิดชอบจัดการข้อมูลพื้นฐานของกระเป๋า เช่น น้ำหนัก, ขนาด
* เป็น Domain Object ที่เก็บ state ของกระเป๋า
* ไม่ควรมีความรับผิดชอบในการคำนวณราคา เพราะไม่ใช่หน้าที่หลักของมัน
* **LuggagePricingSystem Class**:
* รับผิดชอบเฉพาะการคำนวณราคากระเป๋า
* มีความซับซ้อนในการคำนวณที่อาจเปลี่ยนแปลงได้ในอนาคต
* สามารถรองรับกฎการคิดราคาที่หลากหลาย

1. ข้ออื่นๆ

* รองรับการเปลี่ยนแปลงในอนาคตได้ดี



* ทดสอบและบำรุงรักษาได้ง่าย (แก้ไขส่วนการคำนวณราคาได้โดยไม่กระทบ Luggage, Debug ง่าย)
* "มีประโยชน์อย่างไรในการพัฒนาต่อ?"
* เพิ่มวิธีคำนวณราคาใหม่ได้ง่าย
* ไม่ต้องแก้ไข code เดิม
* รองรับความต้องการทางธุรกิจที่อาจเปลี่ยนแปลง
* ลดความเสี่ยงในการแก้ไข code ผิดพลาด

Abstract Base Class

1. **เพื่อบังคับให้ subclass ต้อง implement method ที่จำเป็น**:

* PaymentMethod บังคับให้ทุก payment method ต้องมี processPayment()
* LuggagePricingSystem บังคับให้ทุกระบบคำนวณราคาต้องมี calculateLuggagePrice()

1. **เพื่อสร้างโครงสร้างที่ชัดเจนและขยายได้ในอนาคต**:

* สามารถเพิ่มวิธีการชำระเงินใหม่ได้ง่าย
* สามารถเพิ่มวิธีคำนวณราคากระเป๋าใหม่ได้ง่าย

1. **เพื่อให้เป็นไปตามหลัก OOP**:

* Abstraction: ซ่อนรายละเอียดการทำงานภายใน
* Inheritance: สืบทอดคุณสมบัติจาก parent class
* Polymorphism: object ต่างชนิดสามารถตอบสนองต่อ method เดียวกันได้

1. **เพื่อให้โค้ดมีความยืดหยุ่นและบำรุงรักษาง่าย**:

* แก้ไขเฉพาะส่วนที่ต้องการโดยไม่กระทบส่วนอื่น
* เพิ่มฟีเจอร์ใหม่ได้โดยไม่ต้องแก้ไขโค้ดเดิม

**1. เรื่อง SOLID Principles**

1. "ทำไมต้องแยก LuggagePricingSystem ออกจาก Luggage Class?"

ตอบ: เพื่อทำตามหลัก Single Responsibility Principle โดย:

* Luggage รับผิดชอบเรื่องข้อมูลกระเป๋า
* LuggagePricingSystem รับผิดชอบเรื่องการคำนวณราคา
* ทำให้สามารถเปลี่ยนแปลงวิธีคำนวณราคาได้โดยไม่กระทบ Luggage

1. "อธิบายการใช้ Abstract Class ในระบบ"

ตอบ: มีการใช้ Abstract Class 2 ที่:

* PaymentMethod: เป็น abstract base class สำหรับวิธีการชำระเงิน
* LuggagePricingSystem: เป็น abstract base class สำหรับการคำนวณราคากระเป๋า

1. "ทำไมต้องมี UI Class แยกออกมาต่างหาก?"

ตอบ: เพื่อทำตามหลัก MVC Pattern:

* UI (View): รับผิดชอบการแสดงผล
* Controller: จัดการ logic
* Model: จัดการข้อมูล

**2. เรื่อง Inheritance & Polymorphism**

1. "อธิบายความแตกต่างระหว่าง DebitCard และ CreditCard"

ตอบ:

* ทั้งคู่สืบทอดจาก ATMCard
* DebitCard มี fee 7% และ method calculate\_fee()
* CreditCard มี interest\_rate และ method calculate\_interest()

1. "ทำไมต้องมี ATMCard เป็น parent class?"

ตอบ: เพราะ DebitCard และ CreditCard มีคุณสมบัติร่วมกัน:

* มี card\_number, CVV, EXP
* ต้องมีการ validate\_card()

**3. เรื่อง Design Patterns**

1. "ระบบนี้ใช้ Design Pattern อะไรบ้าง?"

ตอบ:

* MVC Pattern: แยก UI, Controller, และ Model
* Strategy Pattern: ในการคำนวณราคากระเป๋า
* Factory Pattern: ในการสร้าง Payment objects

**4. เรื่อง Sequence Diagram**

1. "ทำไม PassengerPayment.plantuml ต้องมี loop ในการ Update Booking Status?"

ตอบ: เพราะต้องอัพเดทสถานะที่นั่งของผู้โดยสารทุกคนในการจอง

1. "อธิบายการทำงานของ alt ใน PassengerPayment.plantuml"

ตอบ: แสดงทางเลือกในการชำระเงิน 3 แบบ:

* DebitCard (มี fee 7%)
* CreditCard (มี interest)

**5. เรื่อง Class Diagram**

1. "ทำไมต้องใช้ aggregation ระหว่าง Controller กับ Airport?"

* ตอบ: เพราะ Controller เป็นผู้จัดการ Airport objects แต่ Airport สามารถมีอยู่ได้โดยไม่ต้องมี Controller

1. "อธิบายความสัมพันธ์ระหว่าง Account กับ UserDetail"

* ตอบ: เป็นแบบ composition (1-to-1) เพราะ UserDetail เป็นส่วนประกอบของ Account และไม่สามารถมีอยู่ได้โดยไม่มี Account

**6. เรื่อง Implementation**

1. "ทำไมต้องใช้ private attributes ใน Promocode Class?"

* ตอบ: เพื่อ encapsulation:
* ป้องกันการแก้ไขข้อมูลโดยตรง
* ควบคุมการเข้าถึงผ่าน getter/setter

1. "อธิบายการทำงานของ calculate\_price() ใน Luggage Class"

* ตอบ: คำนวณราคาตามน้ำหนัก:
* น้ำหนัก >= 30kg: คิดเพิ่ม 2.25/kg
* น้ำหนัก >= 20kg: คิดเพิ่ม 1.75/kg

**7. เรื่อง API Design**

1. "ทำไมต้องแยก API endpoints ออกจาก Class?"

* ตอบ: เพื่อแยกส่วน presentation layer ออกจาก business logic:
* API รับผิดชอบการรับ-ส่งข้อมูล
* Class รับผิดชอบการประมวลผล

1. "อธิบายการทำงานของ route ใน app.py"

* ตอบ: ใช้ FastAPI routing system:
* กำหนด endpoint URL
* เชื่อมต่อกับ function handlers
* จัดการ HTTP methods

**8. เรื่อง Testing & Maintenance**

1. "ระบบนี้จะทดสอบอย่างไร?"

* ตอบ: สามารถทดสอบได้หลายระดับ:
* Unit Test: ทดสอบแต่ละ class
* Integration Test: ทดสอบการทำงานร่วมกัน
* API Test: ทดสอบ endpoints

1. "หากต้องเพิ่มวิธีการชำระเงินใหม่ จะทำอย่างไร?"

* ตอบ: สร้าง class ใหม่ที่สืบทอดจาก PaymentMethod:
* implement process\_payment()
* เพิ่ม logic เฉพาะของวิธีการชำระเงินนั้น
* ไม่ต้องแก้ไขโค้ดเดิม