Câu hỏi trắc nghiệm

Câu hỏi 1: Thuộc tính của một lớp nên được thiết kế như thế nào để đảm bảo tính đóng gói dữ liệu?

A. Public

B. Private

C. Protected

D. Internal

Câu hỏi 2: Nguyên lý nào đảm bảo rằng dữ liệu chỉ có thể được truy cập thông qua các phương thức công khai của lớp?

A. Hướng trách nhiệm

B. Đóng gói dữ liệu

C. Thực hiện lời gọi nhiều lần

D. Kế thừa

Câu hỏi 3: Thẻ CRC bao gồm những thành phần nào?

B. Class, Responsibility, Collaboration

Câu hỏi 4: Quan hệ nào giữa các lớp thể hiện việc một lớp là thành phần của lớp khác và không thể tồn tại độc lập?

C. Composition

Câu hỏi 5: Sơ đồ FSM mô tả điều gì?

A. Cấu trúc dữ liệu của hệ thống

B. Các trạng thái và chuyển đổi giữa các trạng thái của đối tượng

C. Các lớp và quan hệ giữa các lớp

D. Cách người dùng tương tác với hệ thống

Câu hỏi 6: Nguyên lý nào khuyến khích việc thiết kế phương thức để tái sử dụng nhiều lần trong các ngữ cảnh khác nhau?

A. Hướng trách nhiệm

B. Thực hiện lời gọi nhiều lần

C. Đóng gói dữ liệu

D. Phân lớp

Câu hỏi 7: Chuẩn hóa quan hệ giữa các bảng về dạng 3-NF nhằm mục đích gì?

A. Tăng tốc độ xử lý dữ liệu

B. Giảm thiểu dư thừa dữ liệu và đảm bảo tính nhất quán

C. Đảm bảo các bảng chứa đầy đủ dữ liệu

D. Tăng khả năng mở rộng của hệ thống

Câu hỏi 8: Lớp nào trong hệ thống thường chứa các thuộc tính lưu trữ dữ liệu và không chứa nhiều logic xử lý?

A. Lớp điều khiển

B. Lớp biên

C. Lớp thực thể

D. Lớp giao diện

Câu hỏi 9: Khi xây dựng sơ đồ lớp chi tiết, điều gì cần được bổ sung?

A. Thêm các quan hệ giữa các lớp

B. Thêm các thuộc tính và phương thức đầy đủ cho từng lớp

C. Thêm giao diện người dùng

D. Thêm các sơ đồ tuần tự

Câu hỏi 10: Trong mô hình cơ sở dữ liệu quan hệ, một thuộc tính của bảng được gọi là:

A. Field

B. Row

C. Record

D. Table

Câu hỏi ngắn

1.Tại sao cần đóng gói dữ liệu khi thiết kế lớp?

Đóng gói dữ liệu giúp bảo vệ và kiểm soát truy cập vào thuộc tính của lớp, đảm bảo tính nhất quán và toàn vẹn dữ liệu. Nó ngăn chặn truy cập trực tiếp, chỉ cho phép thao tác thông qua các phương thức (getter, setter), giúp dễ bảo trì và mở rộng hệ thống. Đây là nguyên tắc quan trọng trong lập trình hướng đối tượng để ẩn chi tiết triển khai và tăng tính an toàn cho dữ liệu.

2.Nguyên lý hướng trách nhiệm trong thiết kế phương thức là gì?

Nguyên lý **hướng trách nhiệm** trong thiết kế phương thức nghĩa là mỗi phương thức chỉ nên thực hiện **một nhiệm vụ duy nhất** và có **trách nhiệm rõ ràng**. Điều này giúp mã nguồn dễ hiểu, dễ bảo trì và mở rộng.

3.Thẻ CRC là gì?

Thẻ CRC (Class - Responsibility - Collaboration) là một công cụ được sử dụng trong lập trình hướng

đối tượng để giúp thiết kế và tổ chức các lớp trong hệ thống.

•Class (Lớp): Đại diện cho một thực thể hoặc đối tượng trong hệ thống.

•Responsibility (Trách nhiệm): Mô tả các nhiệm vụ hoặc chức năng mà lớp đảm nhận.

•Collaboration (Cộng tác): Các lớp khác mà lớp này tương tác để thực hiện chức năng của nó.

4.Quan hệ Aggregation và Composition khác nhau như thế nào?

- **Aggregation (Kết tập)**: Mối quan hệ "HAS-A" yếu. Đối tượng con có thể tồn tại độc lập với đối tượng chứa nó. Khi đối tượng chứa bị hủy, đối tượng con vẫn còn tồn tại.
- **Composition (Hợp thành)**: Mối quan hệ "HAS-A" mạnh. Đối tượng con phụ thuộc hoàn toàn vào đối tượng chứa nó. Khi đối tượng chứa bị hủy, đối tượng con cũng bị hủy theo.

5.FSM (Finite State Machine) là gì?

FSM (Finite State Machine - Máy trạng thái hữu hạn) là một mô hình tính toán dùng để mô tả hành vi của một hệ thống thông qua các trạng thái và chuyển đổi giữa các trạng thái. Một FSM bao gồm:

- Tập hợp các trạng thái (states).
- Trạng thái ban đầu (initial state).
- Các điều kiện hoặc sự kiện (transitions) làm thay đổi trạng thái.
- Trạng thái kết thúc (nếu có).

FSM được ứng dụng rộng rãi trong thiết kế hệ thống nhúng, lập trình game, kiểm tra bảo mật, xử lý chuỗi ký tự, và các hệ thống phản ứng theo sự kiện.

6.Mục tiêu của việc chuẩn hóa cơ sở dữ liệu là gì?

Chuẩn hóa cơ sở dữ liệu (Database Normalization) nhằm tối ưu cấu trúc dữ liệu, tránh dư thừa và đảm bảo tính toàn vẹn dữ liệu. Các mục tiêu chính gồm:

1) Giảm dư thừa dữ liệu → Tránh lưu trữ dữ liệu lặp lại không cần thiết.
2) Loại bỏ bất thường trong cập nhật, chèn và xóa dữ liệu → Giảm rủi ro mất mát hoặc xung đột dữ liệu.
3) Cải thiện tính nhất quán và toàn vẹn dữ liệu → Đảm bảo dữ liệu luôn đúng và chính xác.
4) Tăng cường hiệu suất truy vấn → Giúp truy vấn dữ liệu nhanh hơn và dễ bảo trì hơn.

Chuẩn hóa thường được thực hiện thông qua các dạng chuẩn (Normal Forms - NF) như 1NF, 2NF, 3NF, BCNF, v.v.

7.Lớp điều khiển có vai trò gì trong hệ thống?

Lớp điều khiển có vai trò xử lý luồng nghiệp vụ của hệ thống, tiếp nhận yêu cầu từ lớp giao diện, gọi các phương thức thích hợp của lớp thực thể hoặc dịch vụ, và điều phối dữ liệu giữa các lớp.

8.Nguyên lý thực hiện lời gọi nhiều lần giúp ích gì trong thiết kế phương thức?

Nguyên lý thực hiện lời gọi nhiều lần giúp thiết kế phương thức dễ bảo trì, tái sử dụng và mở rộng, giảm trùng lặp code bằng cách gom nhóm các logic chung vào một phương thức có thể gọi lại nhiều lần.

9.Sơ đồ lớp là gì?

Sơ đồ lớp (Class Diagram) là một loại sơ đồ trong **UML (Unified Modeling Language)** dùng để mô tả **cấu trúc** của hệ thống thông qua các **lớp (classes)**, **thuộc tính (attributes)**, **phương thức (methods)** và **mối quan hệ giữa các lớp**.

🔹 **Các thành phần chính của sơ đồ lớp:**

- **Lớp (Class):** Đại diện cho một thực thể trong hệ thống.
- **Thuộc tính (Attributes):** Biểu diễn dữ liệu hoặc trạng thái của lớp.
- **Phương thức (Methods):** Các hành vi hoặc chức năng của lớp.
- **Quan hệ giữa các lớp:** Bao gồm **kế thừa (inheritance), kết hợp (association), phụ thuộc (dependency), tổng hợp (aggregation)**,...

10.Dạng chuẩn 3-NF của cơ sở dữ liệu là gì?

Dạng chuẩn 3 (Third Normal Form - 3NF) là một quy tắc trong **chuẩn hóa cơ sở dữ liệu**, giúp loại bỏ **sự dư thừa dữ liệu** và **đảm bảo tính toàn vẹn của dữ liệu**.

📌 **Điều kiện để một bảng đạt 3NF:**

1. **Phải ở dạng chuẩn 2 (2NF)** (không có phụ thuộc từng phần vào khóa chính).
2. **Không có phụ thuộc bắc cầu** (một cột không khóa không được phụ thuộc vào một cột không khóa khác).

🔹 **Ví dụ bảng chưa đạt 3NF:**

MaSV	TenSV	MaLop	TenLop
SV01	An	L01	CNTT
SV02	Binh	L01	CNTT

- Ở đây, **TenLop** phụ thuộc vào **MaLop**, chứ không phụ thuộc trực tiếp vào **MaSV** (khóa chính).
- Điều này vi phạm 3NF vì có phụ thuộc bắc cầu: **MaSV → MaLop → TenLop**.

🔹 **Cách chuẩn hóa lên 3NF:**

Tách thành hai bảng:

1. **Bảng SinhVien:**

MaSV	TenSV	MaLop
SV01	An	L01
SV02	Binh	L01

Bảng Lop

MaLop	TenLop
L01	CNTT

CÂU HỎI THẢO LUẬN NHÓM

1.Thảo luận về vai trò của việc đóng gói dữ liệu khi thiết kế lớp.

Bảo vệ tính toàn vẹn của dữ liệu

- Giúp ngăn chặn việc truy cập và sửa đổi dữ liệu trực tiếp từ bên ngoài lớp, giảm thiểu nguy cơ lỗi hoặc hành vi không mong muốn.

- Ví dụ, nếu một lớp có thuộc tính `balance` của tài khoản ngân hàng, việc đóng gói đảm bảo rằng chỉ có các phương thức như `deposit()` hoặc `withdraw()` có thể thay đổi giá trị này, tránh trường hợp `balance` bị gán giá trị không hợp lệ.

Kiểm soát quyền truy cập

- Dữ liệu có thể được ẩn đi bằng cách sử dụng các phạm vi truy cập như `private` hoặc `protected`, giúp giới hạn phạm vi mà dữ liệu có thể bị thay đổi.

- Cho phép xác định chính xác những dữ liệu nào có thể đọc, ghi hoặc thay đổi từ bên ngoài thông qua các phương thức `getter` và `setter`.

Dễ dàng bảo trì và mở rộng

- Khi dữ liệu được đóng gói, việc thay đổi cấu trúc hoặc cách thức xử lý dữ liệu chỉ ảnh hưởng đến bên trong lớp mà không làm thay đổi các phần khác của chương trình.

- Điều này giúp mã nguồn dễ bảo trì hơn và tránh các lỗi do thay đổi trực tiếp trên dữ liệu.

Hỗ trợ tính trừu tượng và che giấu thông tin

- Giúp người dùng chỉ quan tâm đến cách sử dụng lớp thay vì cách lớp hoạt động bên trong.

- Ví dụ, một lớp `Car` có thể có một phương thức `startEngine()`, nhưng người dùng không cần biết chi tiết về cách động cơ hoạt động.

Tăng tính bảo mật

- Đóng gói giúp bảo vệ dữ liệu quan trọng khỏi sự truy cập trái phép hoặc sai sót từ các phần khác của chương trình.

- Điều này đặc biệt quan trọng trong các ứng dụng yêu cầu bảo mật cao như hệ thống ngân hàng, quản lý danh tính,...

2.So sánh giữa Aggregation và Composition trong thiết kế lớp.

Aggregation và Composition đều thể hiện mối quan hệ **"has-a"** trong lập trình hướng đối tượng (OOP), nhưng khác nhau về mức độ gắn kết giữa các đối tượng.

Mức độ gắn kết

- **Aggregation:** Mối quan hệ lỏng lẻo, đối tượng con có thể tồn tại độc lập với đối tượng cha.

- **Composition:** Mối quan hệ chặt chẽ, đối tượng con không thể tồn tại nếu đối tượng cha bị hủy.

Sự phụ thuộc

- **Aggregation:** Đối tượng con không phụ thuộc vào đối tượng cha. Nếu đối tượng cha bị hủy, đối tượng con vẫn tồn tại.

- **Composition:** Đối tượng con phụ thuộc hoàn toàn vào đối tượng cha. Nếu đối tượng cha bị hủy, đối tượng con cũng bị hủy theo.

Mối quan hệ

- **Aggregation:** Lớp cha "có" lớp con nhưng không sở hữu hoàn toàn.

- **Composition:** Lớp cha "có" và sở hữu hoàn toàn lớp con.

Tính tái sử dụng

- **Aggregation:** Đối tượng con có thể được dùng lại ở nhiều nơi khác.
- **Composition:** Đối tượng con bị ràng buộc với đối tượng cha, khó tái sử dụng hơn.

Ví dụ thực tế

- **Aggregation:** Một trường học có nhiều giáo viên, nhưng giáo viên vẫn tồn tại ngay cả khi trường học đóng cửa.
- **Composition:** Một ngôi nhà có các phòng, nếu ngôi nhà bị phá hủy, các phòng cũng không còn.

Khi nào sử dụng?

- Dùng **Aggregation** khi đối tượng con có thể tồn tại độc lập với đối tượng cha.
- Dùng **Composition** khi đối tượng con cần phụ thuộc hoàn toàn vào đối tượng cha.

3.Thảo luận về cách xây dựng thẻ CRC hiệu quả cho hệ thống.

Xác định các lớp (Classes)

- **Mục tiêu**: Xác định các đối tượng chính trong hệ thống.
- **Cách thực hiện**: Dựa trên các thực thể trong miền vấn đề (domain) hoặc các thành phần logic của hệ thống để liệt kê các lớp tiềm năng.
- **Ví dụ**: Trong một hệ thống quản lý thư viện, các lớp có thể bao gồm Book (Sách), Member (Thành viên), Loan (Phiếu mượn), và Library (Thư viện).
- **Lưu ý**: Tập trung vào các đối tượng có ý nghĩa rõ ràng trong bối cảnh hệ thống, tránh tạo quá nhiều lớp không cần thiết.

Xác định trách nhiệm (Responsibilities)

- **Mục tiêu**: Xác định những gì mỗi lớp phải biết (thuộc tính) và phải làm (phương thức).

- **Cách thực hiện**: Mô tả trách nhiệm một cách ngắn gọn, cụ thể và tập trung vào chức năng cốt lõi của lớp.
- **Ví dụ**:
    - Lớp Book:
        - Biết: Lưu trữ thông tin về tiêu đề, tác giả, và ISBN.
        - Làm: Cung cấp thông tin về tình trạng mượn (có sẵn hay đã được mượn).
- **Lưu ý**: Trách nhiệm nên phản ánh vai trò chính của lớp, tránh chồng chéo hoặc quá tải trách nhiệm lên một lớp.

Xác định mối quan hệ hợp tác (Collaborations)

- **Mục tiêu**: Hiểu rõ cách các lớp tương tác với nhau để hoàn thành trách nhiệm.
- **Cách thực hiện**: Liệt kê các lớp khác mà một lớp cần hợp tác để thực hiện chức năng của mình.
- **Ví dụ**: Lớp Loan cần hợp tác với:
    - Book: Để kiểm tra tình trạng sách.
    - Member: Để xác định người mượn.
- **Lưu ý**: Việc xác định hợp tác giúp làm rõ sự phụ thuộc giữa các lớp, từ đó tối ưu hóa thiết kế.

Sử dụng thẻ CRC

- **Mục tiêu**: Tạo một biểu diễn trực quan cho mỗi lớp.
- **Cách thực hiện**: Mỗi thẻ CRC bao gồm ba phần:
    - **Tên lớp**: Đặt ở đầu thẻ.
    - **Trách nhiệm**: Liệt kê các trách nhiệm chính.
    - **Hợp tác**: Ghi lại các lớp khác liên quan.
- **Lưu ý**: Thẻ có thể được viết tay trên giấy hoặc sử dụng công cụ phần mềm để dễ dàng chỉnh sửa.

Kiểm tra và tinh chỉnh

- **Mục tiêu**: Đảm bảo thiết kế hoạt động tốt trong thực tế.
- **Cách thực hiện**: Sử dụng các kịch bản hoặc trường hợp sử dụng (use cases) để mô phỏng cách các lớp tương tác. Điều chỉnh thẻ CRC nếu phát hiện:
    - Trách nhiệm chưa được phân bổ hợp lý.
    - Một lớp có quá nhiều hoặc quá ít trách nhiệm.

- **Ví dụ**: Nếu lớp Library phải xử lý cả quản lý sách và thành viên, có thể cần tách thành hai lớp riêng biệt để tránh quá tải.

Tính đến nguyên tắc thiết kế

- **Mục tiêu**: Đảm bảo thiết kế linh hoạt và dễ bảo trì.
- **Cách thực hiện**: Áp dụng các nguyên tắc SOLID:
    - **Single Responsibility**: Mỗi lớp chỉ nên có một trách nhiệm duy nhất.
    - **Open/Closed**: Lớp nên dễ mở rộng nhưng không cần sửa đổi.
    - **Liskov Substitution, Interface Segregation, Dependency Inversion**: Đảm bảo tính linh hoạt và giảm phụ thuộc không cần thiết.
- **Ví dụ**: Nếu lớp Book bắt đầu xử lý cả việc mượn sách, hãy chuyển trách nhiệm đó sang lớp Loan.

Tương tác nhóm

- **Mục tiêu**: Tối ưu hóa thiết kế thông qua thảo luận.
- **Cách thực hiện**: Sử dụng thẻ CRC trong các buổi brainstorming hoặc thiết kế nhóm. Các thành viên có thể di chuyển, bổ sung hoặc chỉnh sửa thẻ để tìm ra thiết kế tốt nhất.
- **Lợi ích**: Tăng tính trực quan và khuyến khích sự tham gia của tất cả thành viên.

Tài liệu hóa

- **Mục tiêu**: Lưu giữ thông tin thiết kế cho các giai đoạn sau.
- **Cách thực hiện**: Sau khi hoàn thiện, sử dụng thẻ CRC như một phần của tài liệu thiết kế để ghi lại cấu trúc và hành vi của hệ thống.
- **Ví dụ**: Chuyển các thẻ CRC thành một phần của tài liệu kỹ thuật hoặc sơ đồ lớp UML.

4.Tại sao cần xây dựng sơ đồ FSM cho các lớp trong hệ thống?

Việc xây dựng Sơ đồ Máy trạng thái hữu hạn (FSM - Finite State Machine) cho các lớp trong hệ thống có những lợi ích quan trọng:

> **Mô hình hóa hành vi**: FSM giúp mô tả rõ ràng các trạng thái mà một đối tượng có thể tồn tại và các chuyển tiếp giữa các trạng thái đó.

Quản lý độ phức tạp: Phân tách hành vi phức tạp thành các trạng thái riêng biệt, dễ hiểu hơn.

Xác định đầy đủ các tình huống: Giúp xác định tất cả các trạng thái và sự kiện có thể xảy ra, tránh bỏ sót các trường hợp trong thiết kế.

Tăng tính ổn định: Xác định rõ các chuyển tiếp trạng thái hợp lệ, ngăn chặn các chuyển tiếp không mong muốn.

Làm rõ tương tác: Giúp hiểu rõ cách đối tượng phản ứng với các sự kiện bên ngoài.

Tạo cơ sở cho việc triển khai: Cung cấp khuôn khổ rõ ràng để chuyển thiết kế thành mã nguồn.

Hỗ trợ kiểm thử: Dễ dàng tạo các trường hợp kiểm thử dựa trên các trạng thái và chuyển tiếp.

Tài liệu hóa: Cung cấp tài liệu trực quan về hành vi của lớp cho các nhà phát triển.

5.Thảo luận về các bước chuẩn hóa cơ sở dữ liệu và vai trò của từng bước.

Bước 1: Dạng chuẩn thứ nhất (1NF - First Normal Form)

✔ Điều kiện:

- Mỗi cột trong bảng chỉ chứa **một giá trị đơn** (*atomic value*).
- Không có danh sách hoặc nhóm lặp trong một cột.

✔ Vai trò:

- Đảm bảo dữ liệu được lưu trữ dưới dạng bảng hợp lệ.
- Tránh trùng lặp dữ liệu theo kiểu danh sách trong một ô.

Bước 2: Dạng chuẩn thứ hai (2NF - Second Normal Form)

✔ Điều kiện:

- Đạt **1NF**.

- Mỗi thuộc tính không khóa phải **phụ thuộc hoàn toàn** vào khóa chính, không phải phụ thuộc một phần.
  X→A(A phụ thuộc hoa`n toa`n va`o X, kho^ng phải một pha^`n của X)X → A \quad \text{(A phụ thuộc hoàn toàn vào X, không phải một phần của X)}X→A(A phụ thuộc hoa`n toa`n va`o X, kho^ng phải một pha^`n của X)

✔ Vai trò:

- Loại bỏ **phụ thuộc một phần** (*partial dependency*).
- Tránh dữ liệu dư thừa do chỉ phụ thuộc vào một phần của khóa chính (trong bảng có khóa chính tổng hợp).

Bước 3: Dạng chuẩn thứ ba (3NF - Third Normal Form)

✔ Điều kiện:

- Đạt **2NF**.
- Không có **phụ thuộc bắc cầu** (*transitive dependency*), tức là mỗi cột không khóa chỉ phụ thuộc vào khóa chính, không phụ thuộc vào cột không khóa khác.
  X→Y,Y→Z⇒X→ZX → Y, Y → Z \Rightarrow X → ZX→Y,Y→Z⇒X→Z
  Nếu YYY không phải là khóa chính, thì có phụ thuộc bắc cầu.

✔ Vai trò:

- Ngăn chặn các dữ liệu có thể bị **sai lệch khi cập nhật**.
- Loại bỏ sự dư thừa thông tin trong các cột không phải khóa.

Bước 4: Dạng chuẩn Boyce-Codd (BCNF - Boyce-Codd Normal Form)

✔ Điều kiện:

- Đạt **3NF**.
- Mọi phụ thuộc hàm **X → Y** đều có **X là siêu khóa** (*superkey*).

✔ Vai trò:

- Xử lý một số trường hợp ngoại lệ mà **3NF chưa tối ưu hoàn toàn**.
- Loại bỏ trường hợp một cột không khóa phụ thuộc vào một tập hợp con của khóa chính.

6.Làm thế nào để đảm bảo rằng các phương thức của lớp tuân thủ nguyên lý hướng trách nhiệm?

1. Áp dụng nguyên lý Single Responsibility Principle (SRP)

Mỗi phương thức chỉ thực hiện một nhiệm vụ duy nhất, tránh tình trạng một phương thức xử lý nhiều công việc khác nhau.

Ví dụ:

· Sai: Một phương thức xử lý cả dữ liệu đầu vào, tính toán và ghi log.

· Đúng: Chia thành các phương thức riêng biệt:

• validateInput() – Kiểm tra đầu vào.

• calculateResult() – Tính toán kết quả.

• logActivity() – Ghi log hoạt động.

2. Tránh lặp code bằng nguyên lý DRY (Don't Repeat Yourself)

Không viết lặp lại cùng một logic trong nhiều phương thức hoặc lớp khác nhau.

Nếu một đoạn code xuất hiện nhiều lần, nên đưa nó vào một phương thức hoặc lớp riêng để dễ bảo trì.

Ví dụ:

· Sai: Lặp lại đoạn kiểm tra đầu vào trong nhiều phương thức.

· Đúng: Tạo một phương thức chung validateInput() và gọi lại khi cần.

3. Tuân theo nguyên lý Open/Closed Principle (OCP)

Thiết kế phương thức sao cho có thể mở rộng mà không cần sửa đổi mã nguồn gốc.

Sử dụng kế thừa (inheritance) và đa hình (polymorphism) để dễ mở rộng.

Ví dụ:

· Sai:

```
if (shape == "circle") calculateCircleArea();

else if (shape == "square") calculateSquareArea();
```

· Đúng:

```
abstract class Shape { abstract float getArea(); }
```

Các lớp con Circle, Square sẽ kế thừa và định nghĩa riêng phương thức getArea().

4. Giảm phụ thuộc giữa các lớp (Loose Coupling)

Giảm sự phụ thuộc giữa các lớp bằng cách sử dụng interface hoặc Dependency Injection (DI).

Tránh việc một lớp phải biết quá nhiều về lớp khác, giúp dễ dàng thay đổi hoặc mở rộng hệ thống.

Ví dụ:

· Sai:

```
class PaymentService {
    PayPalService payPal = new PayPalService();
    void processPayment() { payPal.pay(); }
}
```

· Đúng:

```
interface PaymentGateway { void processPayment(); }
```

Các lớp PayPalService, StripeService có thể triển khai PaymentGateway, giúp linh hoạt khi thay đổi phương thức thanh toán.

7.Thảo luận về ưu và nhược điểm của việc sử dụng sơ đồ lớp chi tiết trong thiết kế hệ thống.

Ưu và nhược điểm của sơ đồ lớp chi tiết trong thiết kế hệ thống:

- **Ưu điểm**:
    - Giúp hiểu rõ cấu trúc và mối quan hệ giữa các lớp trong hệ thống.
    - Hỗ trợ lập trình viên trong việc triển khai mã nguồn dễ dàng hơn.
    - Giúp phát hiện sớm các vấn đề thiết kế và đảm bảo tính nhất quán của hệ thống.
    - Tạo tài liệu hữu ích cho bảo trì và mở rộng hệ thống sau này.
- **Nhược điểm**:
    - Tốn nhiều thời gian để xây dựng và cập nhật khi hệ thống thay đổi.
    - Có thể trở nên phức tạp và khó đọc đối với hệ thống lớn.
    - Nếu không được cập nhật thường xuyên, sơ đồ có thể lỗi thời so với mã nguồn thực tế.

8.Tại sao cần chuẩn hóa cơ sở dữ liệu đến dạng 3-NF?

7. **Chuẩn hóa cơ sở dữ liệu đến dạng 3-NF giúp**:
    - Loại bỏ sự dư thừa dữ liệu, giúp tiết kiệm không gian lưu trữ.
    - Đảm bảo tính toàn vẹn và nhất quán của dữ liệu.
    - Giảm nguy cơ lỗi khi cập nhật, chèn hoặc xóa dữ liệu.
    - Cải thiện hiệu suất truy vấn bằng cách giảm sự trùng lặp và tối ưu hóa các mối quan hệ giữa các bảng.

9.Thảo luận về tầm quan trọng của lớp điều khiển trong mô hình MVC.

Trong mô hình **MVC (Model-View-Controller)**, **lớp điều khiển (Controller)** đóng vai trò trung gian giữa **Model (Dữ liệu)** và **View (Giao diện người dùng)**.

🔹 **Tầm quan trọng của lớp điều khiển:**

1. **Tách biệt logic nghiệp vụ và giao diện** → Giúp hệ thống dễ bảo trì và mở rộng.

2. **Xử lý yêu cầu từ người dùng** → Nhận dữ liệu đầu vào từ giao diện, xử lý và gửi đến Model.

3. **Cập nhật View** → Sau khi xử lý, Controller yêu cầu View hiển thị dữ liệu mới.

4. **Cải thiện khả năng kiểm thử** → Dễ dàng kiểm tra logic mà không ảnh hưởng đến giao diện.

5. **Hỗ trợ mở rộng** → Có thể thay đổi View hoặc Model mà không làm ảnh hưởng đến Controller.

📌 Ví dụ trong Java với Servlet (Web MVC)

- **Model:** Lớp xử lý dữ liệu (`User.java`).

- **View:** JSP hiển thị giao diện (`user.jsp`).

- **Controller:** Servlet nhận request từ client, xử lý và trả kết quả (`UserController.java`).

java

CopyEdit

```
@WebServlet("/UserController")

public class UserController extends HttpServlet {

    protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {

        User user = new User("John", "john@example.com");

        request.setAttribute("user", user);
```

```
request.getRequestDispatcher("user.jsp").forward(request,
response);
    }
}
```

💡 **Kết luận:** Controller giúp mô hình MVC hoạt động hiệu quả, dễ quản lý và mở rộng.

10.Làm thế nào để xây dựng sơ đồ lớp chi tiết cho một hệ thống lớn và phức tạp?

🔹 **Bước 1: Xác định yêu cầu hệ thống**

- Xác định các thực thể chính trong hệ thống (**User, Product, Order,...**).
- Hiểu cách các thực thể này tương tác với nhau.

🔹 **Bước 2: Xác định các lớp chính**

- Phân loại theo mô hình **MVC**:
    - **Model (Dữ liệu):** Các lớp đại diện cho thực thể chính.
    - **Controller (Xử lý):** Điều khiển luồng dữ liệu.
    - **View (Giao diện):** Hiển thị thông tin cho người dùng.

🔹 **Bước 3: Thêm các thuộc tính và phương thức**

- Mỗi lớp cần có các **thuộc tính** (biến) và **phương thức** (hành động).
- Xác định **mức độ truy cập** (`private`, `public`, `protected`).

🔹 **Bước 4: Xác định mối quan hệ giữa các lớp**

- **Kế thừa (`extends`)**: Một lớp con kế thừa lớp cha.

- **Quan hệ kết hợp (`association`)**: Một lớp chứa tham chiếu đến lớp khác.

- **Tổng hợp & thành phần (`aggregation`, `composition`)**: Xác định mức độ phụ thuộc giữa các lớp.

🔹 **Bước 5: Vẽ sơ đồ lớp bằng UML**

- Dùng các công cụ như **Draw.io, StarUML, Visual Paradigm** để trực quan hóa sơ đồ.

CÂU HỎI TÌNH HUỐNG

1. Trong quá trình thiết kế hệ thống quản lý sinh viên, bạn nhận ra rằng nhiều lớp có các thuộc tính giống nhau. Bạn sẽ xử lý vấn đề này như thế nào?

Sử dụng Kế thừa (Inheritance)

Kế thừa giúp tái sử dụng thuộc tính và phương thức chung giữa các lớp. Khi nhiều lớp có các thuộc tính giống nhau, ta có thể tạo một lớp cha chứa các thuộc tính chung, và các lớp con sẽ kế thừa từ lớp cha này. Điều này giúp giảm lặp mã và tổ chức mã nguồn gọn gàng hơn.

Lợi ích:

- Giảm thiểu sự lặp lại mã nguồn.

- Dễ dàng bảo trì và thay đổi thông tin chung ở một nơi duy nhất.

Sử dụng Interface

Nếu các lớp có hành vi giống nhau nhưng không có thuộc tính chung, có thể sử dụng interface để định nghĩa các hành vi cần thiết. Các lớp có thể triển khai các hành vi này mà không cần kế thừa thuộc tính từ lớp cha.

Lợi ích:

- Cung cấp sự linh hoạt cao cho các lớp thực hiện hành vi chung mà không phải kế thừa thuộc tính.

- Các lớp có thể kết hợp nhiều interface khác nhau.

Sử dụng Composition

Composition là cách kết hợp các đối tượng vào trong nhau thay vì kế thừa. Nếu nhiều lớp có các thuộc tính giống nhau nhưng không muốn tạo mối quan hệ kế thừa, ta có thể tách các thuộc tính chung thành một lớp riêng biệt và sau đó sử dụng lớp đó trong các lớp khác.

Lợi ích:

- Tăng tính linh hoạt trong thiết kế vì không cần phải tuân thủ quan hệ kế thừa.
- Dễ dàng mở rộng hệ thống mà không làm thay đổi quá nhiều các lớp hiện tại.

2. Sau khi hoàn thành sơ đồ lớp, nhóm phát triển phát hiện thiếu một số phương thức cần thiết. Làm thế nào để bổ sung vào sơ đồ lớp?

Xác định các phương thức cần thiết

Trước hết, nhóm phát triển cần xác định rõ các phương thức thiếu. Điều này có thể dựa trên yêu cầu chức năng của hệ thống hoặc các tình huống sử dụng cụ thể mà hệ thống cần hỗ trợ.

Cập nhật các lớp có liên quan

Sau khi xác định được các phương thức thiếu, nhóm cần thêm chúng vào **các lớp** phù hợp trong sơ đồ lớp. Đảm bảo rằng các phương thức bổ sung này phù hợp với chức năng của lớp và không vi phạm nguyên lý đóng gói (encapsulation).

Chỉnh sửa mối quan hệ giữa các lớp nếu cần

Nếu các phương thức bổ sung yêu cầu thay đổi mối quan hệ giữa các lớp (ví dụ, thêm quan hệ giữa các lớp qua phương thức gọi, sử dụng kế thừa, hoặc tạo thêm các lớp hỗ trợ), thì sơ đồ lớp cần được điều chỉnh lại để phản ánh sự thay đổi này.

Đảm bảo tính nhất quán trong sơ đồ

Các phương thức mới cần được bổ sung một cách có hệ thống vào các lớp, bao gồm:

- **Tên phương thức:** Đảm bảo tên phương thức rõ ràng, dễ hiểu và phù hợp với nhiệm vụ.

- **Tham số:** Xác định rõ các tham số cần thiết cho phương thức (nếu có).

- **Kiểu trả về:** Cập nhật kiểu dữ liệu trả về cho phương thức.

Rà soát lại toàn bộ sơ đồ lớp

Sau khi đã bổ sung phương thức, nhóm phát triển cần rà soát lại toàn bộ sơ đồ lớp để đảm bảo tính **nhất quán** và **đúng đắn** về mặt logic. Kiểm tra xem các phương thức có tương thích với các lớp và mối quan hệ giữa chúng hay không, đồng thời xác định xem có cần bổ sung hoặc thay đổi gì khác để hỗ trợ chức năng của phương thức mới.

Cập nhật tài liệu và thông báo cho các thành viên

Cuối cùng, sau khi chỉnh sửa sơ đồ lớp, nhóm cần cập nhật **tài liệu hệ thống** để đảm bảo rằng tất cả các thành viên trong nhóm phát triển đều nắm rõ những thay đổi và có thể thực hiện chúng khi triển khai.

3. Một nhóm phát triển gặp khó khăn trong việc phân biệt giữa Aggregation và Composition khi thiết kế sơ đồ lớp. Làm thế nào để giúp nhóm giải quyết vấn đề này?

Đặt câu hỏi về mối quan hệ sở hữu và vòng đời

Hướng dẫn họ đặt câu hỏi cụ thể:

- "Khi đối tượng chứa bị hủy, đối tượng được chứa có nên bị hủy theo không?"
    - Nếu có → Composition
    - Nếu không → Aggregation

Kiểm tra tính phụ thuộc

- **Composition**: Đối tượng con không tồn tại nếu không có đối tượng chủ
- **Aggregation**: Đối tượng con có thể tồn tại độc lập, không phụ thuộc vào đối tượng chủ

Sử dụng ví dụ thực tế để minh họa

- **Composition**:
    - Xe hơi và động cơ (động cơ là một phần không thể tách rời của xe)
    - Con người và tim (tim không tồn tại độc lập ngoài cơ thể)
- **Aggregation**:
    - Trường học và học sinh (học sinh vẫn tồn tại khi rời trường)

    - Công ty và nhân viên (nhân viên có thể chuyển việc)

Xác định kỹ thuật triển khai trong mã nguồn

- **Composition**: Thường được triển khai bằng cách tạo/hủy đối tượng con trong constructor/destructor của đối tượng chứa
- **Aggregation**: Thường được triển khai bằng cách truyền tham chiếu hoặc con trỏ đến đối tượng được tạo từ bên ngoài

Sử dụng ký hiệu UML chính xác

- **Composition**: Mũi tên có đầu hình thoi được tô đen (◆)
- **Aggregation**: Mũi tên có đầu hình thoi rỗng (◇)

4. Trong quá trình xây dựng thẻ CRC, nhóm phát triển phát hiện nhiều lớp có trách nhiệm trùng lặp. Làm thế nào để xử lý tình huống này?

Xem xét lại việc phân chia trách nhiệm:

- Kiểm tra từng lớp để đảm bảo rằng mỗi lớp chỉ đảm nhiệm một nhiệm vụ chính theo nguyên tắc **Single Responsibility Principle (SRP)**.
- Xác định xem sự trùng lặp có do việc phân chia nhiệm vụ không hợp lý hay không.

Tái cấu trúc các lớp:

- **Gộp các lớp có chức năng tương tự:** Nếu hai hay nhiều lớp thực hiện cùng một chức năng, cân nhắc hợp nhất chúng thành một lớp duy nhất để tránh dư thừa.
- **Tách các chức năng chung:** Nếu chức năng chung xuất hiện ở nhiều lớp, có thể tách ra thành một lớp độc lập hoặc một module chung để các lớp khác cùng sử dụng, áp dụng nguyên tắc **DRY (Don't Repeat Yourself)**.

Sử dụng kế thừa hoặc giao diện:

- Tạo ra một lớp cha hoặc một giao diện chứa các phương thức chung, sau đó các lớp có trách nhiệm tương tự có thể kế thừa từ lớp cha hoặc triển khai giao diện này. Điều này giúp duy trì tính mở rộng và dễ bảo trì.

Đánh giá lại mối quan hệ giữa các lớp:

- Xem xét các mối quan hệ trong thẻ CRC, nếu thấy có sự phụ thuộc quá mức giữa các lớp, hãy điều chỉnh lại để mỗi lớp hoạt động độc lập nhất có thể.

5. Khách hàng yêu cầu thêm một chức năng mới sau khi sơ đồ lớp đã được hoàn thiện. Nhóm phát triển cần làm gì để cập nhật sơ đồ lớp?

Khi khách hàng yêu cầu thêm một chức năng mới, nhóm phát triển cần làm các bước sau để cập nhật sơ đồ lớp (UML Class Diagram):

Bước 1: Xác định phạm vi thay đổi

- Chức năng mới có ảnh hưởng đến lớp hiện có hay cần tạo lớp mới?
- Xác định các thuộc tính và phương thức mới cần thêm.

Bước 2: Kiểm tra nguyên lý thiết kế OOP

- Đảm bảo các thay đổi tuân thủ nguyên lý SOLID để không ảnh hưởng đến mã nguồn hiện có:
    - OCP (Open/Closed Principle): Mở rộng nhưng không sửa đổi lớp cũ.
    - SRP (Single Responsibility Principle): Tránh làm một lớp có quá nhiều nhiệm vụ.

Bước 3: Cập nhật sơ đồ lớp

- Nếu chức năng mở rộng lớp hiện có → Thêm thuộc tính và phương thức mới vào lớp đó.
- Nếu cần lớp mới → Tạo lớp mới và xác định mối quan hệ (association, inheritance, aggregation, composition) với các lớp khác.
- Kiểm tra lại các ràng buộc (constraints) và quy tắc nghiệp vụ (business rules).

Bước 4: Cập nhật tài liệu & mã nguồn

- Cập nhật tài liệu thiết kế và đảm bảo rằng sơ đồ mới phù hợp với code.
- Viết lại mã nguồn nếu cần, nhưng phải kiểm thử lại hệ thống sau khi chỉnh sửa.

6. Khi xây dựng sơ đồ FSM cho lớp DonHang, nhóm phát triển gặp khó khăn trong việc xác định các trạng thái và sự kiện. Bạn sẽ hướng dẫn nhóm như thế nào?

Khi nhóm phát triển gặp khó khăn trong việc xác định trạng thái (states) và sự kiện (events) trong sơ đồ FSM (Finite State Machine) của lớp `DonHang`, có thể làm theo các bước sau:

Bước 1: Xác định các trạng thái chính của `DonHang`

- Xác định các giai đoạn chính mà đơn hàng có thể trải qua trong vòng đời của nó.
- Ví dụ về các trạng thái của `DonHang`:
    - `TaoMoi` (Đơn hàng mới tạo)
    - `ChoXacNhan` (Chờ xác nhận)
    - `DaXacNhan` (Đã xác nhận)
    - `DangVanChuyen` (Đang vận chuyển)
    - `HoanThanh` (Đã giao hàng thành công)
    - `DaHuy` (Đã hủy)

Bước 2: Xác định các sự kiện kích hoạt trạng thái

- Mỗi trạng thái thay đổi khi có một sự kiện cụ thể xảy ra.
- Ví dụ về các sự kiện thay đổi trạng thái của `DonHang`:
    - `taoDon()` → `TaoMoi`
    - `xacNhanDon()` → `ChoXacNhan` → `DaXacNhan`
    - `huyDon()` → `DaHuy`

- `batDauGiaoHang()` → `DangVanChuyen`
- `hoanThanhGiaoHang()` → `HoanThanh`

Bước 3: Vẽ sơ đồ FSM

- Vẽ các trạng thái của đơn hàng dưới dạng hình tròn hoặc hình chữ nhật.
- Vẽ các mũi tên thể hiện sự chuyển đổi giữa các trạng thái khi có sự kiện xảy ra.
- Ghi rõ sự kiện kích hoạt trên từng đường chuyển trạng thái.

Bước 4: Kiểm tra logic và tối ưu sơ đồ

- Đảm bảo rằng không có trạng thái nào bị cô lập (không thể tới hoặc thoát ra).
- Kiểm tra xem có xảy ra vòng lặp vô hạn hoặc trạng thái không hợp lệ không.
- Đảm bảo mọi trạng thái đầu vào và đầu ra đều hợp lý.

Ví dụ sơ đồ FSM cho lớp `DonHang`

```
(TaoMoi) --[xacNhanDon()]--> (ChoXacNhan)
--[xacNhan()]--> (DaXacNhan)

(DaXacNhan) --[batDauGiaoHang()]--> (DangVanChuyen)

(DangVanChuyen) --[hoanThanhGiaoHang()]--> (HoanThanh)

(TaoMoi) --[huyDon()]--> (DaHuy)

(ChoXacNhan) --[huyDon()]--> (DaHuy)

(DaXacNhan) --[huyDon()]--> (DaHuy) [Chỉ khi chưa giao
hàng]
```

7. Trong quá trình chuẩn hóa cơ sở dữ liệu, nhóm phát triển gặp vấn đề khi các bảng chứa nhiều dữ liệu dư thừa. Làm thế nào để xử lý vấn đề này?

Để xử lý vấn đề dư thừa dữ liệu trong quá trình chuẩn hóa cơ sở dữ liệu, nhóm phát triển có thể thực hiện các bước sau:

- **Phân tích các thuộc tính và xác định khóa chính** để đảm bảo mỗi bảng chỉ chứa thông tin liên quan.
- **Tách bảng chứa dữ liệu dư thừa thành các bảng nhỏ hơn** theo chuẩn 1-NF, 2-NF, và 3-NF.
- **Sử dụng khóa ngoại** để liên kết các bảng nhằm loại bỏ dữ liệu trùng lặp.
- **Kiểm tra và tối ưu hóa mô hình quan hệ** để đảm bảo dữ liệu không bị mất trong quá trình chuẩn hóa.

8. Một nhóm phát triển gặp khó khăn khi thiết kế các phương thức cho lớp vì chưa hiểu rõ nguyên lý hướng trách nhiệm. Làm thế nào để hướng dẫn nhóm áp dụng nguyên lý này?

Để hướng dẫn nhóm áp dụng **nguyên lý hướng trách nhiệm**, có thể thực hiện các bước sau:

1. **Giải thích nguyên lý SRP (Single Responsibility Principle)**:
    - Mỗi lớp hoặc phương thức chỉ nên có **một lý do để thay đổi**.
    - Nếu một phương thức xử lý nhiều trách nhiệm, hãy **tách thành nhiều phương thức nhỏ hơn**.
2. **Phân tích yêu cầu và chia nhỏ trách nhiệm**:
    - Xác định từng chức năng cụ thể của hệ thống.
    - Nhóm các hành vi liên quan vào một lớp, tránh tình trạng một lớp xử lý quá nhiều thứ.
3. **Áp dụng mô hình kiến trúc phù hợp (VD: MVC, Service-Oriented)**:
    - **Lớp thực thể (Entity)**: Chỉ chứa dữ liệu.
    - **Lớp dịch vụ (Service)**: Chứa logic xử lý nghiệp vụ.

- **Lớp điều khiển (Controller)**: Điều phối luồng dữ liệu giữa UI và Service.

4. **Sử dụng ví dụ thực tế**:
    - Đưa ra một bài toán thực tế, thiết kế một lớp sai nguyên lý, sau đó hướng dẫn nhóm cách cải tiến theo SRP.
5. **Kiểm tra và phản hồi code**:
    - Review code định kỳ để đảm bảo các phương thức tuân theo nguyên lý SRP.
    - Khuyến khích nhóm viết code rõ ràng, dễ bảo trì.

9. Trong quá trình xây dựng sơ đồ lớp chi tiết, khách hàng yêu cầu thay đổi một số yêu cầu đã thống nhất trước đó.

Nhóm phát triển nên xử lý như thế nào?

Khi khách hàng yêu cầu thay đổi các yêu cầu đã thống nhất trong quá trình phát triển sơ đồ lớp chi tiết, nhóm phát triển cần xử lý một cách chuyên nghiệp và có hệ thống để đảm bảo dự án không bị ảnh hưởng tiêu cực. Dưới đây là các bước nhóm có thể thực hiện:

1. **Ghi nhận và đánh giá yêu cầu thay đổi**:
    - Lắng nghe khách hàng và ghi lại chi tiết các yêu cầu thay đổi.
    - Đánh giá mức độ ảnh hưởng của thay đổi đến sơ đồ lớp hiện tại, bao gồm các lớp, quan hệ, và chức năng liên quan.
2. **Thảo luận nội bộ trong nhóm**:
    - Tổ chức một cuộc họp ngắn với nhóm phát triển để phân tích tác động của thay đổi lên thiết kế hiện tại, mã nguồn (nếu đã triển khai), và tiến độ dự án.
    - Xác định xem thay đổi có khả thi không và cần bao nhiêu thời gian, công sức để thực hiện.
3. **Trao đổi lại với khách hàng**:
    - Trình bày cho khách hàng về tác động của thay đổi (ví dụ: thay đổi sơ đồ lớp có thể kéo dài thời gian phát triển hoặc tăng chi phí).
    - Đề xuất các giải pháp thay thế nếu thay đổi quá phức tạp hoặc không phù hợp với giai đoạn hiện tại.
4. **Cập nhật tài liệu và kế hoạch**:

    - Nếu cả hai bên đồng ý với thay đổi, cập nhật sơ đồ lớp chi tiết, tài liệu yêu cầu, và kế hoạch dự án (nếu cần).
    - Đảm bảo tất cả thành viên trong nhóm được thông báo về thay đổi này.
5. **Áp dụng quy trình quản lý thay đổi (Change Management)**:
    - Nếu dự án có quy trình quản lý thay đổi chính thức, hãy tuân thủ (ví dụ: lập yêu cầu thay đổi - Change Request, phê duyệt, và triển khai).
    - Điều này giúp kiểm soát rủi ro và duy trì tính minh bạch với khách hàng.
6. **Thực hiện và kiểm tra**:
    - Điều chỉnh sơ đồ lớp theo yêu cầu mới và kiểm tra lại để đảm bảo tính nhất quán và đúng đắn.
    - Tiếp tục theo dõi tiến độ để đảm bảo không bị chậm trễ.

Lưu ý: Sự linh hoạt là cần thiết, nhưng nhóm cũng nên nhấn mạnh tầm quan trọng của việc thống nhất yêu cầu ban đầu để tránh thay đổi liên tục, gây gián đoạn dự án.

10.Khi kiểm tra lại sơ đồ lớp, nhóm phát triển nhận thấy một số quan hệ giữa các lớp bị sai. Làm thế nào để sửa lại sơ

đồ lớp mà không ảnh hưởng đến tiến độ dự án?

Việc phát hiện sai sót trong quan hệ giữa các lớp là điều bình thường trong quá trình phát triển phần mềm. Để sửa lại sơ đồ lớp mà không làm ảnh hưởng đến tiến độ dự án, nhóm phát triển có thể làm theo các bước sau:

1. **Xác định và phân loại lỗi**:
    - Xác định cụ thể các quan hệ sai (ví dụ: quan hệ kế thừa, tổng hợp, hay liên kết bị nhầm lẫn).
    - Đánh giá mức độ nghiêm trọng của lỗi (ảnh hưởng đến toàn bộ hệ thống hay chỉ một phần nhỏ).
2. **Ưu tiên sửa chữa**:
    - Tập trung sửa các lỗi lớn có thể ảnh hưởng đến logic chính của hệ thống trước.
    - Các lỗi nhỏ hoặc ít tác động có thể được xử lý sau nếu thời gian gấp rút.
3. **Sửa đổi từng bước**:
    - Thay vì sửa toàn bộ sơ đồ cùng lúc, hãy sửa từng quan hệ sai và kiểm tra ngay sau mỗi lần sửa để đảm bảo không tạo ra lỗi mới.
    - Sử dụng công cụ mô hình hóa (như UML tools) để tự động cập nhật và kiểm tra tính nhất quán.
4. **Kiểm tra tác động**:

    - Xem xét các lớp liên quan đến quan hệ bị sai và đảm bảo rằng việc sửa đổi không làm thay đổi chức năng đã triển khai.
    - Nếu mã nguồn đã được viết dựa trên sơ đồ lớp cũ, kiểm tra xem mã có cần điều chỉnh không.
5. **Tối ưu hóa thời gian**:
    - Phân công thành viên có kinh nghiệm xử lý việc sửa đổi để giảm thiểu thời gian thực hiện.
    - Nếu cần, làm việc song song: một số thành viên sửa sơ đồ lớp, trong khi những người khác tiếp tục các nhiệm vụ khác không bị ảnh hưởng.
6. **Cập nhật tài liệu và thông báo**:
    - Sau khi sửa xong, cập nhật sơ đồ lớp và các tài liệu liên quan.
    - Thông báo cho nhóm về thay đổi để mọi người đồng bộ thông tin.
7. **Ngăn ngừa lỗi tương lai**:
    - Tổ chức một buổi review nhanh sau khi sửa để rút kinh nghiệm, đảm bảo các quan hệ trong sơ đồ lớp được kiểm tra kỹ hơn trong tương lai.

Mẹo: Nếu dự án đang ở giai đoạn gấp rút, có thể tạm thời ghi chú các lỗi nhỏ trong sơ đồ và sửa sau khi hoàn thành các nhiệm vụ quan trọng, miễn là lỗi không ảnh hưởng đến chức năng chính.

Cả hai tình huống trên đều yêu cầu sự phối hợp chặt chẽ trong nhóm và giao tiếp hiệu quả với khách hàng (nếu cần). Việc xử lý linh hoạt nhưng có tổ chức sẽ giúp duy trì tiến độ và chất lượng dự án.