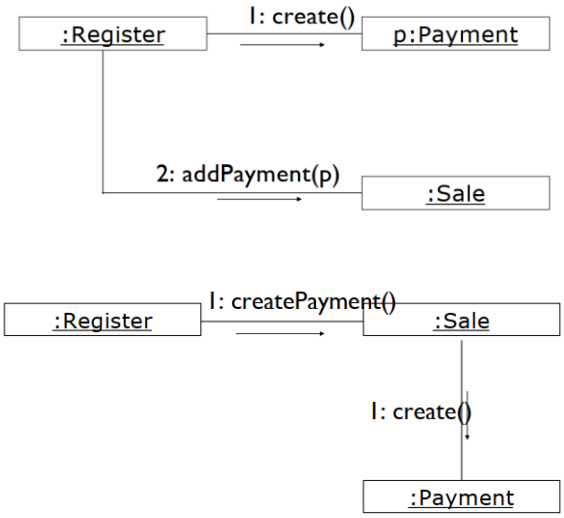
Sinh viên: Nguyễn Chế Định

MSSV:102220223

1. **Low coupling**
   - Coupling:
     - Là số quan hệ giữa các đối tượng/subsystem.
     - Thể hiện độ mạnh yếu về sự liên kết giữa 2 thành phần của hệ thống.
     - Một thành phần có liên kết yếu thì không phụ thuộc vào quá nhiều thành phần khác.
   - 2 có liên kết với nhau nếu:
     - Thành phần này có quan hệ cộng tác aggreation/composition với thành phần kia.
     - Thành phần này triển khai (implement) thành phần kia.
   - Một class liên kết với nhiều class khác thì có nhiều vấn đề sau:
     - Phải thay đổi vì class được liên kết thay đổi.
     - Khó hiểu nếu được xét riêng lẻ (chỉ đọc class này mà không đọc class khác).
     - Khó để tái sử dụng vì cần phải có tạo thêm các class để class này liên kết vào.
   - Giải pháp:
     - Gán trách nhiệm để liên kết với các class khác thấp.
     - Giảm thiểu các mối quan hệ phụ thuộc với nhiều class.
   - Ví dụ:
     - Class Register liên kết với cả 2 class là Payment và Sale. Chúng ta có thể chỉ liên kết Register với Sale, và để Sale thực hiện tạo Payment, như vậy sẽ giúp đảm bảo low coupling.



2. **High cohesion:**

   Đây là nguyên tắc thiết kế nhằm tạo ra các thành phần có sự tập trung chức năng nhằm hướng tới mục đích cụ thể.

Để thực hiện được việc này cần xác định mục đích cụ thể cho từng phần tử, sau đó tập hợp các chức năng, trách nhiệm liên quan vào một phần tử theo mục đích đã xác định. Nhờ vậy mỗi thành phần chỉ cần quản lý một số ít các chức năng cụ thể.

Lợi ích của việc này:

- Giúp code dễ hiểu, dễ bảo trì
- Dễ tái sử dụng mã, phát huy ưu điểm của hướng đối tượng: Do mỗi thành phần thực hiện một chức năng cụ thể và có mối quan hệ chặt chẽ với các thành phần khác, việc tái sử dụng mã trở nên dễ dàng hơn
- Kết hợp thấp, phân tán thấp: (Low Coupling): các thành phần không phụ thuộc quá nhiều vào nhau, giúp cho việc thay đổi hoặc mở rộng hệ thống trở nên dễ dàng hơn.

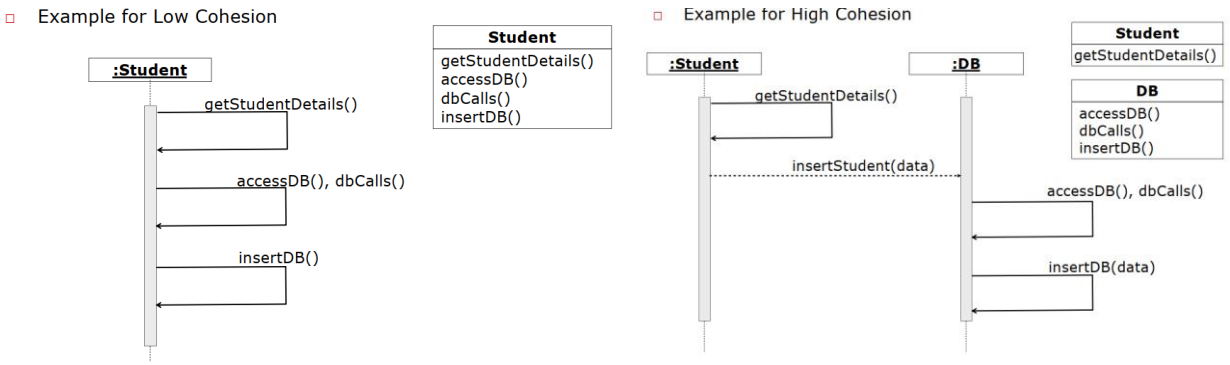
Nếu thực hiện high cohesion không tốt, Một lớp thực hiện nhiều việc không liên quan hoặc làm quá nhiều công việc (low cohesion) thì sẽ dẫn đến nhiều vấn đề:

- Khó hiểu
- Khó tái sử dụng
- Khó bảo trì
- Bị ảnh hưởng bởi sự thay đổi, thay đổi phần tử này ảnh hưởng nhiều đến phần tử khác

Để high cohesion một class phải:

- Có ít phương thức
- Có code ngắn gọn, ít dòng.
- Không làm quá nhiều công việc, không có quá nhiều phương thức
- Có mức độ liên quan cao trong code

Ví dụ:



Thay vì để Student thực hiện hết các phương thức, thì ta sẽ xác định chức năng của nó là thực hiện các việc liên quan đến sinh viên: getStudentDetails(), còn các chức năng không liên quan sẽ được thực hiện bởi 1 lớp khác. Ở đây các phương thức accessDB(), dbCalls(), insertDB() liên quan đến data nên ta cho lớp DB thực hiện.

3. Creator

Khái niệm

Creator pattern không phải là một mẫu thiết kế độc lập như Factory Method hay Abstract Factory mà là một nguyên tắc hướng dẫn về việc phân công trách nhiệm trong quá trình tạo đối tượng. Nó được mô tả trong phương pháp GRASP (General Responsibility Assignment Software Patterns) do Craig Larman giới thiệu.

Nguyên tắc

Nguyên tắc Creator trong GRASP khuyên rằng một lớp nên chịu trách nhiệm tạo một đối tượng nếu một hoặc nhiều điều kiện sau đây đúng:

Lớp chứa hoặc tổng hợp đối tượng cần được tạo ra.

Lớp sử dụng đối tượng cần được tạo ra.

Lớp có thông tin để khởi tạo đối tượng.

Lớp giữ các đối tượng trong một tập hợp và các đối tượng đó cần được khởi tạo.

Chức năng

Hướng dẫn giao phó trách nhiệm tạo đối tượng

Giúp tìm ra class chịu trách nhiệm cho việc tạo đối tượng

Lợi ích

Hỗ trợ low coupling giữa các class

Giảm mức độ phụ thuộc và tăng tính tái sử dụng

Ví dụ: Do Sale chứa SalasLineItem nên Sale nên có trách nhiệm tạo các object của SalesLineItem

Phươnng thức “makeLineItem(quantity)” sẽ được giới thiệu trong Sale class

4. Controller

Khái niệm

Trong một hệ thống, Bộ điều khiển (Controller) là thành phần đầu tiên bên ngoài lớp giao diện người dùng (UI) chịu trách nhiệm tiếp nhận và xử lý các tác vụ của hệ thống.

Nguyên tắc

Bộ điều khiển sẽ ủy thác công việc cho các đối tượng khác. Nó chỉ nhận các yêu cầu nhưng không trực tiếp giải quyết chúng.

Mô hình Bộ điều khiển (Controller pattern) có thể được áp dụng cho tất cả các hệ thống cần xử lý các sự kiện bên ngoài. Một lớp Controller được chọn để xử lý các sự kiện này.

Có 2 hướng có thể đi cho việc sử dụng Controller:

- Dùng 1 controller
- Dùng nhiều controller

Ví dụ

Khi tạo một lịch hẹn thì Controller sẽ nhận input nút add new appointment của người dùng. Controller sẽ giao lại công việc khởi tạo một lịch hẹn mới cho lớp Appointment và việc lưu trữ dữ liệu vào database cho các lớp khác như theo mô hình ba lớp thì giao lại cho lớp DAL, DTO

Ưu điểm:

- Là một mô hình phân quyền đơn giản: Giao diện người dùng (UI) không nên chứa logic ứng dụng. Điều này giúp cho code dễ hiểu, dễ bảo trì và tránh các lỗi do logic ứng dụng bị nhúng vào trong UI.
- Tăng khả năng tái sử dụng và các giao diện có thể cắm (pluggable interfaces): Các logic xử lý trong bộ điều khiển có thể được tái sử dụng cho các sự kiện khác nhau, giảm thiểu mã cần viết. Đồng thời, mô hình này cho phép dễ dàng thay thế các thành phần xử lý logic bằng các thành phần khác, tăng tính linh hoạt của hệ thống.
- Tạo cơ hội để suy luận về trạng thái của một ngữ cảnh sử dụng (use-case): Bộ điều khiển có thể đảm bảo các hành động được thực hiện theo đúng thứ tự hợp lệ. Ví dụ, trong quá trình tạo tài khoản, bộ điều khiển có thể đảm bảo người dùng nhập thông tin đăng ký trước khi tạo mật khẩu.

Nhược điểm:

- Bộ điều khiển phình to (Bloated controllers): Nếu một bộ điều khiển đơn lẻ nhận tất cả các sự kiện của hệ thống và xử lý quá nhiều logic, nó có thể trở nên phức tạp và khó bảo trì.
- Các vấn đề:
  - Một bộ điều khiển xử lý quá nhiều sự kiện.
  - Bộ điều khiển chứa quá nhiều thuộc tính (có thể trùng lặp với thông tin ở các nơi khác trong hệ thống).

Cách khắc phục:

- Thêm nhiều bộ điều khiển: Chia nhỏ logic xử lý thành các bộ điều khiển con phụ trách các sự kiện riêng biệt, giúp giảm tải cho bộ điều khiển chính và dễ dàng quản lý hơn.
- Thiết kế bộ điều khiển để chủ yếu ủy thác việc thực hiện các tác vụ của hệ thống cho các đối tượng khác: Bộ điều khiển sẽ tập trung vào việc điều phối và xử lý luồng các sự kiện, phân công các tác vụ cụ thể cho các đối tượng chuyên biệt khác xử lý.

5. Information Expert

Khái niệm:

Nguyên tắc Chuyên gia Thông tin là một nguyên tắc thiết kế cơ bản trong phát triển phần mềm hướng đối tượng, hướng dẫn việc gán trách nhiệm cho các lớp. Nguyên tắc này khuyên rằng một trách nhiệm nên được giao cho lớp sở hữu thông tin và kiến thức liên quan nhất để thực hiện trách nhiệm đó.

Nguyên tắc:

Nguyên tắc Chuyên gia Thông tin khuyến khích bạn xác định các lớp chứa dữ liệu và phương thức có giá trị liên quan đến một nhiệm vụ cụ thể.

Giải pháp:

Giao trách nhiệm cho lớp có thông tin cần thiết để hoàn thành nhiệm vụ đó.

Ví dụ:

Hệ thống quản lí thư viện

Khi một người dùng muốn mượn sách, trách nhiệm kiểm tra xem sách có sẵn hay không nên thuộc về lớp Book. Lớp Book chứa thông tin về tình trạng sẵn có của nó và có thể thực hiện các kiểm tra cần thiết mà không cần phụ thuộc vào các lớp khác. Điều này thúc đẩy sự gắn kết cao và giảm sự phụ thuộc giữa các lớp.