Java Design Pattern overview - Mẫu thiết kế trong Java

Design Pattern là giải pháp kỹ thuật thiết kế (hay còn gọi là mẫu thiết kế) chương trình để giải quyết tối ưu các vấn đề chung, thường gặp trong lập trình.

1. Design Pattern là gì

Design patterns là các giải pháp đã được tối ưu hóa, được tái sử dụng cho các vấn đề lập trình mà chúng ta gặp phải hàng ngày. Nó là một khuôn mẫu có thể được áp dụng vào mỗi trường hợp cụ thể.

Các vấn đề mà bạn gặp phải có thể bạn sẽ tự nghĩ ra cách giải quyết nhưng có thể nó chưa phải là tối ưu. Design Pattern giúp bạn giải quyết vấn đề một cách tối ưu nhất, cung cấp cho bạn các giải pháp trong lập trình OOP. Nó không phải là ngôn ngữ cụ thể nào cả. Design patterns có thể thực hiện được ở phần lớn các ngôn ngữ lập trình. Ta thường gặp nó nhất trong lập trình OOP.

Để xây dựng và áp dụng Design Pattern, cần nắm rõ các kiến thức sau:

- Bốn đặc tính của OOP: Thừa kế, Đa hình, Trừu tượng, Bao đóng.
- Khái niệm interface và abstract. Cái này cực kỳ quan trọng, để hiểu và áp dụng 2 khái niệm này có thể sẽ mất một thời gian, nhưng khi bạn nắm chắc nó bạn sẽ thấy nó thực sự cần thiết.

2. Lợi ích của Design Pattern

- Các mẫu thiết kế đã được xác định và cung cấp cách tiếp cận chuẩn theo ngành để giải quyết vấn đề lặp lại, vì vậy sẽ tiết kiệm được thời gian nếu chúng ta sử dụng mẫu thiết kế hợp lý.
- Sử dụng các mẫu thiết kế thúc đẩy khả năng tái sử dụng dẫn đến phát triển phần mềm nhanh hơn. Nó giúp giảm tổng chi phí của sản phẩm phần mềm.
- Khi áp dụng 1 mẫu thiết kế, nó làm cho code dễ hiểu và gỡ lỗi.
- Design Pattern giúp bạn tái sử dụng mã lệnh và dễ dàng mở rộng.
- Nó là tập hơn những giải pháp đã được tối ưu hóa, đã được kiểm chứng để giải quyết các vấn đề trong software engineering. Vậy khi bạn gặp bất kỳ khó khăn gì, design patterns là kim chỉ nam giúp bạn giải quyết vấn đề thay vì tự tìm kiếm giải pháp cho một vấn đề đã được chứng minh.
- Design pattern cung cấp giải pháp ở dạng tổng quát, giúp tăng tốc độ phát triển phần mềm bằng cách đưa ra các mô hình test, mô hình phát triển đã qua kiểm nghiệm.
- Dùng lại các design pattern giúp tránh được các vấn đề tiềm ẩn có thể gây ra những lỗi lớn, dễ dàng nâng cấp, bảo trì về sau.
- Giúp cho các lập trình viên có thể hiểu code của người khác 1 cách nhanh chóng (có thể hiểu là tính communicate).
- Mọi thành viên trong team có thể dễ dàng trao đổi với nhau để cùng xây dựng dự án mà k mất quá nhiều thời gian.

3. Các loại Design Pattern

Năm 1994, bốn tác giả **Erich Gamma, Richard Helm, Ralph Johnson và John Vlissides** đã cho xuất bản một cuốn sách với tiêu đề **Design Patterns – Elements of Reusable Object-Oriented Software**, đây là khởi nguồn của khái niệm design pattern trong lập trình phần mềm.

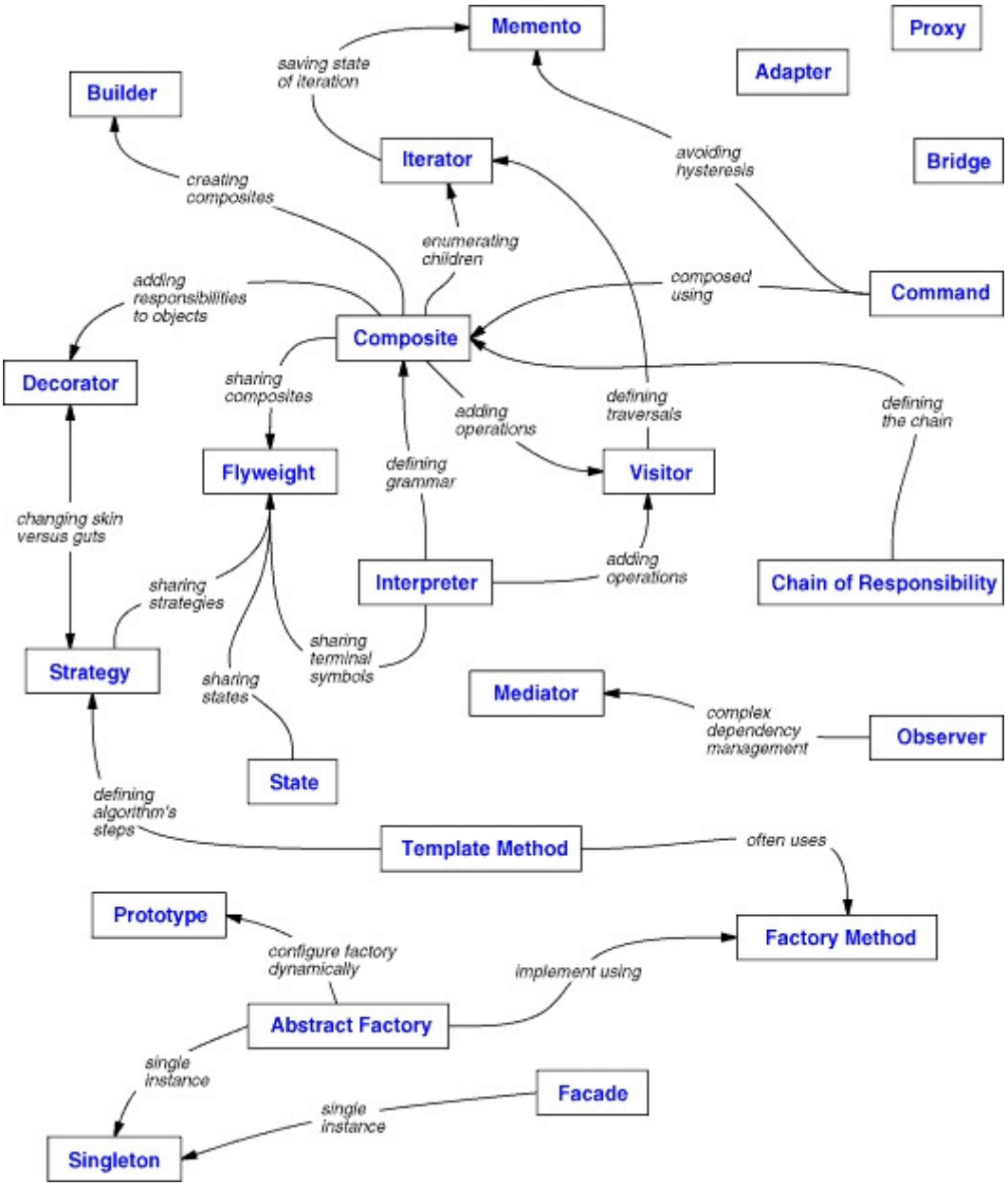
Bốn tác giả trên được biết đến rộng rãi dưới tên Gang of Four (bộ tứ). Theo quan điểm của bốn người, design pattern chủ yếu được dựa theo những quy tắc sau đây về thiết kế hướng đối tượng.

Có 4 loại Design Pattern sau:

TT	Design Pattern	Mô tả	Các loại

1	**Creational Patterns**	Nhóm này cung cấp phương pháp tạo ra các đối tượng một cách linh hoạt hơn. Nghĩa là quyết định đối tượng nào được tạo ra tuỳ thuộc vào trường hợp sử dụng nhất định.	5 mẫu: Factory Method, Abstract Factory, Builder, Prototype, Singleton
2	**Structural Pattern**	Nhóm này liên quan đến sự kết hợp giữa các đối tượng với nhau	7 mẫu: Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy
3	**Behavioral Patterns**	Mẫu thiết kế này trình bày phương pháp thiết kế liên quan đến hành vi của các đối tượng.	11 mẫu: Interpreter, Template Method, Chain of Responsibility, Command, Iterator, Mediator, Memento, Observer, State, Strategy, Visitor
4	**J2EE Pattern**	Nhóm này cung cấp phương pháp thiết kế chương trình theo mô hình nhiều tầng (multiple tier)	

Hình dưới là mối quan hệ giữa 23 Design Pattern cơ bản:



3.1. Nhóm Creational (nhóm khởi tạo)

Nhóm này cung cấp phương pháp tạo ra các đối tượng một cách linh hoạt hơn. Nghĩa là quyết định đối tượng nào được tạo ra tuỳ thuộc vào trường hợp sử dụng nhất định.

- **Singleton**:
  - Đảm bảo 1 class chỉ có 1 instance và cung cấp 1 điểm truy xuất toàn cục đến nó.
  - Tần suất sử dụng: cao trung bình.
- **Abstract Factory**:
  - Cung cấp một interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần qui định lớp khi hay xác định lớp cụ thể (concrete) tạo mỗi đối tượng.
  - Tần suất sử dụng: cao.
- **Factory Method**:
  - Định nghĩa Interface để sinh ra đối tượng nhưng để cho lớp con quyết định lớp nào được dùng để sinh ra đối tượng Factory method cho phép một lớp chuyển quá trình

khởi tạo đối tượng cho lớp con.
    - Tần suất sử dụng: cao.
- **Builder**:
    - Tách rời việc xây dựng (construction) một đối tượng phức tạp khỏi biểu diễn của nó sao cho cùng một tiến trình xây dựng có thể tạo được các biểu diễn khác nhau.
    - Tần suất sử dụng: trung bình thấp.
- **Prototype**:
    - Qui định loại của các đối tượng cần tạo bằng cách dùng một đối tượng mẫu, tạo mới nhờ vào sao chép đối tượng mẫu này.
    - Tần suất sử dụng: trung bình.

3.2. Nhóm Structural (nhóm cấu trúc)

Nhóm này liên quan đến sự kết hợp giữa các đối tượng với nhau.

- **Adapter**:
    - Do vấn đề tương thích, thay đổi interface của một lớp thành một interface khác phù hợp với yêu cầu người sử dụng lớp.
    - Tần suất sử dụng: cao trung bình.
- **Bridge**:
    - Tách rời ngữ nghĩa của một vấn đề khỏi việc cài đặt, mục đích để cả hai bộ phận (ngữ nghĩa và cài đặt) có thể thay đổi độc lập nhau.
    - Tần suất sử dụng: trung bình.
- **Composite**:
    - Tổ chức các đối tượng theo cấu trúc phân cấp dạng cây. Tất cả các đối tượng trong cấu trúc được thao tác theo một cách thuần nhất như nhau.
      Tạo quan hệ thứ bậc bao gộp giữa các đối tượng. Client có thể xem đối tượng bao gộp và bị bao gộp như nhau -> khả năng tổng quát hoá trong code của client -> dễ phát triển, nâng cấp, bảo trì.
    - Tần suất sử dụng: cao trung bình.
- **Decorator**:
    - Gán thêm trách nhiệm cho đối tượng (mở rộng chức năng) vào lúc chạy (dynamically).
    - Tần suất sử dụng:trung bình.
- **Facade**:
    - Cung cấp một interface thuần nhất cho một tập hợp các interface trong một “hệ thống con” (subsystem). Nó định nghĩa 1 interface cao hơn các interface có sẵn để làm cho hệ thống con dễ sử dụng hơn.
    - Tần suất sử dụng: cao.
- **Flyweight**:
    - Sử dụng việc chia sẻ để thao tác hiệu quả trên một số lượng lớn đối tượng “cở nhỏ” (chẳng hạn paragraph, dòng, cột, ký tự...).
    - Tần suất sử dụng: thấp.
- **Proxy**:
    - Cung cấp đối tượng đại diện cho một đối tượng khác để hỗ trợ hoặc kiểm soát quá trình truy xuất đối tượng đó. Đối tượng thay thế gọi là proxy.

- Tần suất sử dụng: cao trung bình.

3.3. Nhóm Behavioral (nhóm hành vi/ tương tác)

Mẫu thiết kế này trình bày phương pháp thiết kế liên quan đến hành vi của các đối tượng.

- **Chain of Responsibility:**
  - Khắc phục việc ghép cặp giữa bộ gởi và bộ nhận thông điệp. Các đối tượng nhận thông điệp được kết nối thành một chuỗi và thông điệp được chuyển dọc theo chuỗi nầy đến khi gặp được đối tượng xử lý nó. Tránh việc gắn kết cứng giữa phần tử gởi request với phần tử nhận và xử lý request bằng cách cho phép hơn 1 đối tượng có có cơ hội xử lý request. Liên kết các đối tượng nhận request thành 1 dây chuyền rồi gửi request xuyên qua từng đối tượng xử lý đến khi gặp đối tượng xử lý cụ thể.
  - Tần suất sử dụng: trung bình thấp.
- **Command:**
  - Mỗi yêu cầu (thực hiện một thao tác nào đó) được bao bọc thành một đối tượng. Các yêu cầu sẽ được lưu trữ và gởi đi như các đối tượng.Đóng gói request vào trong một Object, nhờ đó có thể nthông số hoá chương trình nhận request và thực hiện các thao tác trên request: sắp xếp, log, undo…
  - Tần suất sử dụng: cao trung bình.
- **Interpreter:**
  - Hỗ trợ việc định nghĩa biểu diễn văn phạm và bộ thông dịch cho một ngôn ngữ.
  - Tần suất sử dụng: thấp.
- **Iterator:**
  - Truy xuất các phần tử của đối tượng dạng tập hợp tuần tự (list, array, …) mà không phụ thuộc vào biểu diễn bên trong của các phần tử.
  - Tần suất sử dụng: cao.
- **Mediator:**
  - Định nghĩa một đối tượng để bao bọc việc giao tiếp giữa một số đối tượng với nhau.
  - Tần suất sử dụng: trung bình thấp.
- **Memento:**
  - Hiệu chỉnh và trả lại như cũ trạng thái bên trong của đối tượng mà vẫn không vi phạm việc bao bọc dữ liệu.
  - Tần suất sử dụng: thấp.
- **Observer:**
  - Định nghĩa sự phụ thuộc một-nhiều giữa các đối tượng sao cho khi một đối tượng thay đổi trạng thái thì tất cả các đối tượng phụ thuộc nó cũng thay đổi theo.
  - Tần suất sử dụng: cao.
- **State:**
  - Cho phép một đối tượng thay đổi hành vi khi trạng thái bên trong của nó thay đổi, ta có cảm giác như class của đối tượng bị thay đổi.
  - Tần suất sử dụng: trung bình.
- **Strategy:**
  - Bao bọc một họ các thuật toán bằng các lớp đối tượng để thuật toán có thể thay đổi độc lập đối với chương trình sử dụng thuật toán.Cung cấp một họ giải thuật cho phép client chọn lựa linh động một giải thuật cụ thể khi sử dụng.
  - Tần suất sử dụng: cao trung bình.

- **Template method:**
  - Định nghĩa phần khung của một thuật toán, tức là một thuật toán tổng quát gọi đến một số phương thức chưa được cài đặt trong lớp cơ sở; việc cài đặt các phương thức được ủy nhiệm cho các lớp kế thừa.
  - Tần suất sử dụng: trung bình.
- **Visitor:**
  - Cho phép định nghĩa thêm phép toán mới tác động lên các phần tử của một cấu trúc đối tượng mà không cần thay đổi các lớp định nghĩa cấu trúc đó.
  - Tần suất sử dụng: thấp.

Revision #3
Created 2 November 2019 04:51:34 by Laptrinh.vn
Updated 14 August 2021 08:08:31 by Laptrinh.vn