

# Nguyên lý Thiết kế và Kiến trúc Phần mềm

## Chương 3: Mẫu thiết kế trong hướng đối tượng

# Giới thiệu về Design Pattern

---

- Design Pattern là một kỹ thuật lập trình được sử dụng để giải quyết các vấn đề phức tạp trong phát triển phần mềm.
- Design Pattern giúp lập trình viên xây dựng các giải pháp có tính tái sử dụng, dễ bảo trì và mở rộng.
- Design Pattern không phải là một giải pháp tổng quát cho tất cả các vấn đề trong lập trình, mà là các mẫu thiết kế đã được kiểm chứng và sử dụng rộng rãi trong cộng đồng lập trình.

# Lịch sử phát triển Design Pattern

---

- Khái niệm Design Pattern được đưa ra bởi ba tác giả Erich Gamma, Richard Helm, Ralph Johnson và John Vlissides vào năm 1994 trong cuốn sách "Design Patterns: Elements of Reusable Object-Oriented Software".
- Cuốn sách này là một tài liệu quan trọng trong lĩnh vực phát triển phần mềm và đã trở thành cẩm nang cho các lập trình viên trong việc sử dụng Design Pattern.
- Sau này, các Design Pattern khác nhau đã được phát triển và được sử dụng rộng rãi trong cộng đồng lập trình viên trên toàn thế giới.

# Khái niệm Design Pattern

- Design Pattern là một mô hình giải quyết các vấn đề phát triển phần mềm được phát triển và sử dụng bởi cộng đồng lập trình viên.
- Mỗi Design Pattern mô tả một giải pháp cụ thể cho một vấn đề được gặp phải trong quá trình phát triển phần mềm.
- Các Design Pattern được đóng gói thành các mẫu thiết kế (design templates) để dễ dàng sử dụng và áp dụng trong các dự án phát triển phần mềm.

# Loại Design Pattern

Creational Patterns

Structural Patterns

Behavioral Patterns

# Lợi ích của Design Pattern

- Design Pattern giúp tạo ra các giải pháp phát triển phần mềm có tính chất đồng nhất và dễ bảo trì.
- Sử dụng Design Pattern giúp giảm thiểu sự trùng lặp mã và tăng tính tái sử dụng của mã.
- Design Pattern giúp lập trình viên giải quyết các vấn đề phức tạp trong phát triển phần mềm một cách hiệu quả và nhanh chóng hơn.



# Lợi ích của Design Pattern


- Khi sử dụng Design Pattern, chúng ta có thể tạo ra các giải pháp phát triển phần mềm có tính đồng nhất và dễ bảo trì hơn.
- Ví dụ, khi áp dụng Design Pattern "Singleton", ta có thể đảm bảo rằng chỉ có một instance duy nhất của một đối tượng được tạo ra trong suốt quá trình thực thi của ứng dụng. Điều này giúp giảm thiểu sự trùng lặp và tăng tính tái sử dụng của mã, đồng thời giảm thiểu khả năng phát sinh lỗi trong quá trình phát triển và bảo trì.

# Hạn chế của Design Pattern: Design Pattern là một công cụ hữu ích trong phát triển phần mềm, nhưng nó cũng có những hạn chế nhất định.

Khó áp dụng đúng cách: Các Design Pattern cần phải được sử dụng đúng cách để đem lại lợi ích. Nếu áp dụng sai cách, chúng có thể dẫn đến các vấn đề về hiệu suất và khó bảo trì.

Khó hiểu: Design Pattern đòi hỏi người dùng có kiến thức chuyên môn về lập trình và thiết kế phần mềm để hiểu và sử dụng chúng. Điều này có thể gây khó khăn cho người mới bắt đầu trong lĩnh vực này.





# Design Pattern là một công cụ hữu ích trong phát triển phần mềm, nhưng nó cũng có những hạn chế nhất định.

Tăng độ phức tạp của mã:  
Sử dụng quá nhiều Design Pattern có thể dẫn đến mã phức tạp hơn, khó hiểu hơn và khó bảo trì hơn.

Khó phát hiện: Các Design Pattern có thể không phù hợp với mọi tình huống và không phải lúc nào cũng dễ phát hiện được khi nào cần sử dụng chúng.

Design Pattern là một công cụ hữu ích trong phát triển phần mềm, nhưng nó cũng có những hạn chế nhất định.

- Không phù hợp cho tất cả các dự án: Các Design Pattern không phải lúc nào cũng phù hợp cho tất cả các loại dự án. Chúng có thể không phù hợp cho các dự án đơn giản hoặc quá phức tạp.

# Các trường hợp sử dụng Design Pattern

- Sử dụng Design Pattern trong các dự án phát triển lớn và phức tạp để giúp tăng tính tái sử dụng và dễ bảo trì của mã.
- Sử dụng Design Pattern trong các dự án có yêu cầu về độ tin cậy và khả năng mở rộng cao.
- Sử dụng Design Pattern trong các dự án phát triển phần mềm đòi hỏi sự linh hoạt cao và có thể dễ dàng thay đổi.

# Những điều cần lưu ý khi sử dụng Design Pattern

- Không nên sử dụng Design Pattern một cách cố định, mà nên tùy chỉnh và điều chỉnh để phù hợp với yêu cầu cụ thể của dự án.
- Sử dụng quá nhiều Design Pattern có thể dẫn đến sự phức tạp và khó bảo trì.
- Cần đảm bảo rằng các Design Pattern được sử dụng đúng cách và đúng mục đích để tránh tình trạng "over-engineering".



# Những điều cần lưu ý khi sử dụng Design Pattern

- Chúng ta cần đảm bảo rằng các Design Pattern được sử dụng đúng cách và đúng mục đích để tránh tình trạng "over-engineering".
  - Ví dụ, nếu chúng ta sử dụng Design Pattern "Decorator" để thêm các tính năng mới vào một đối tượng, nhưng lại sử dụng quá nhiều decorator và tạo ra một đối tượng quá phức tạp, thì điều này có thể làm cho mã khó bảo trì và gây ra hiệu suất thấp. Do đó, khi sử dụng Design Pattern, chúng ta cần đảm bảo rằng chúng được áp dụng một cách hợp lý và phù hợp với yêu cầu cụ thể của dự án.
-



# Thực hành sử dụng Design Pattern

- Để sử dụng thành thạo Design Pattern, cần phải đọc và nghiên cứu các tài liệu về các Design Pattern và cách sử dụng chúng.
  - Cần phải thực hành sử dụng Design Pattern trong các dự án thực tế để có thể nắm vững và hiểu rõ cách sử dụng chúng.
  - Cần phải thường xuyên cập nhật kiến thức về Design Pattern để áp dụng những phương pháp và kỹ thuật mới nhất trong lập trình.
-

## Phân loại mẫu thiết kế

Mẫu thiết kế (design pattern) có thể được phân loại thành các nhóm dựa trên mục đích sử dụng hoặc cách triển khai.



Dưới đây là một số cách phân loại mẫu thiết kế:

Phân loại dựa  
trên mục đích  
sử dụng

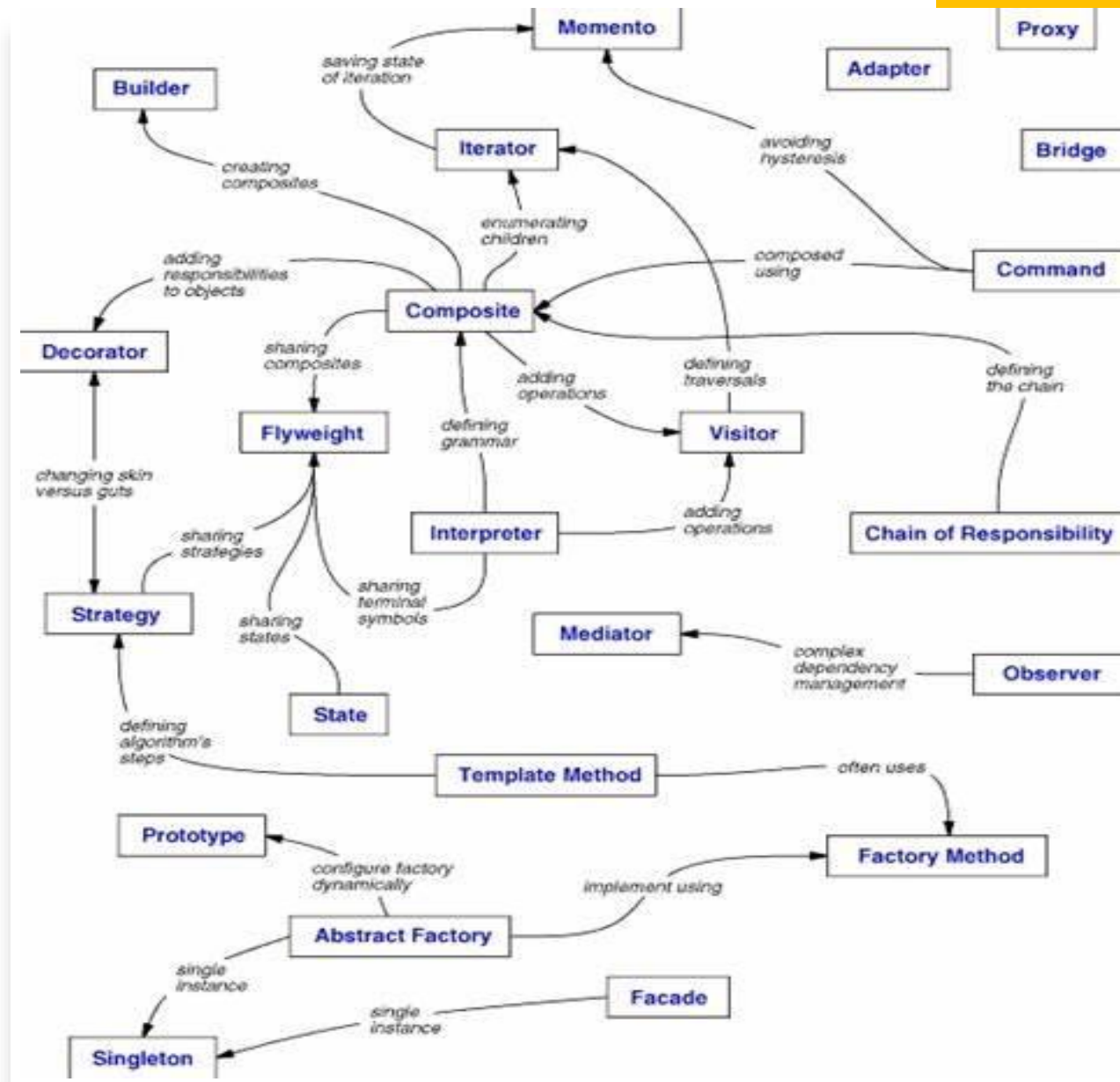
Phân loại dựa  
trên cách triển  
khai

Phân loại dựa  
trên số lượng  
đối tượng

Phân loại dựa  
trên phạm vi  
áp dụng

# Các loại mẫu thiết kế cổ điển

- Có nhiều mẫu design pattern cổ điển (**23 mẫu thiết kế**) đã được giới thiệu trong cuốn sách "Design Patterns: Elements of Reusable Object-Oriented Software" của Erich Gamma, Richard Helm, Ralph Johnson và John Vlissides.







Phân loại dựa  
trên mục đích sử  
dụng:

Creational (tạo lập): tập trung vào quá trình khởi tạo đối tượng.

Structural (cấu trúc): tập trung vào cách tổ chức các đối tượng và lớp thành các cấu trúc lớn hơn.

Behavioral (hành vi): tập trung vào cách các đối tượng tương tác và hoạt động với nhau.

# Phân loại dựa trên cách triển khai:

---

Class-based (dựa trên lớp): dựa trên việc sử dụng lớp và kế thừa để triển khai các mẫu thiết kế.

---

Object-based (dựa trên đối tượng): dựa trên việc sử dụng đối tượng để triển khai các mẫu thiết kế.

---

Functional (hàm): dựa trên việc sử dụng các hàm để triển khai các mẫu thiết kế.

# Phân loại dựa trên số lượng đối tượng:

Single-instance  
(đơn vị): chỉ có  
một đối tượng  
được tạo ra.

Multiple-instance  
(nhiều đơn vị):  
có thể tạo nhiều  
đối tượng.

# Phân loại dựa trên phạm vi áp dụng:

Global (toàn cục):  
áp dụng cho toàn  
bộ hệ thống.


Local (địa  
phương): áp dụng  
cho một phần cụ  
thể của hệ thống.

## Mẫu thiết kế (Design Pattern) dưới góc nhìn các hệ thống phi tập trung

Các hệ thống phi tập trung (hệ thống phân tán) là những hệ thống gồm nhiều nút xử lý độc lập, có thể giao tiếp với nhau qua mạng.

Các hệ thống phân tán đòi hỏi các giải pháp đặc biệt để đảm bảo tính nhất quán, khả năng mở rộng, hiệu năng và an toàn của dữ liệu.

05 mẫu thiết kế phổ biến nhất khi thiết kế các hệ thống phân tán quy mô lớn: Saga, CQRS, Event Sourcing, Leader và Followers, Sharding



# Mẫu thiết kế Saga

- Mẫu thiết kế Saga: Một mẫu thiết kế cho phép xử lý các yêu cầu song song và đảm bảo tính nhất quán của dữ liệu trong trường hợp có lỗi xảy ra.
- Mỗi yêu cầu được chia thành nhiều bước nhỏ, và nếu một bước nào đó thất bại, các bước trước đó sẽ được hoàn tác

# Mẫu thiết kế CQRS

Mẫu thiết kế CQRS: Một mẫu thiết kế cho phép tách biệt các hoạt động đọc và ghi dữ liệu, để tăng cường khả năng mở rộng và hiệu năng của hệ thống.

CQRS viết tắt của  
Command Query  
Responsibility  
Segregation.



# Mẫu thiết kế Event Sourcing

- Mẫu thiết kế Event Sourcing: Một mẫu thiết kế cho phép lưu trữ các sự kiện thay đổi trạng thái của hệ thống, thay vì lưu trữ trạng thái hiện tại.
- Điều này cho phép tái hiện lại lịch sử của hệ thống, cũng như hỗ trợ các tính năng như undo/redo, audit log hay snapshot



# Mẫu thiết kế Leader and Followers

- Mẫu thiết kế Leader and Followers: Một mẫu thiết kế cho phép sao chép dữ liệu từ một nút chính (leader) sang các nút phụ (followers), để cải thiện khả năng chịu lỗi và khả năng đọc dữ liệu của hệ thống. Các yêu cầu ghi dữ liệu sẽ được gửi đến leader, rồi được lan truyền sang các followers

# Mẫu thiết kế Sharding

- Mẫu thiết kế Sharding:  
Một mẫu thiết kế cho phép chia nhỏ dữ liệu thành các phân vùng (shards), và phân bổ chúng trên các nút khác nhau, để giảm thiểu áp lực lên một nút duy nhất. Sharding giúp tăng khả năng mở rộng và hiệu năng của hệ thống