**TRƯỜNG ĐẠI HỌC SƯ PHẠM TP. HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**

---□□---

Logo

Description automatically generated

**ĐỒ ÁN MÔN HỌC**

**LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG**

**PROTOTYPE DESSIGN PATTERN**

Giáo viên hướng dẫn : GV.Trần Quang Huy

Nhóm SV thực hiện : 47.01.104.200 – Hồ Ngọc Hưng Thịnh

: 47.01.104.231 – Phạm Anh Tùng

Mã lớp học phần : COMP130402

TP. HCM, tháng 5 năm 2023

**TRƯỜNG ĐẠI HỌC SƯ PHẠM TP.HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

---□□---

Logo

Description automatically generated

**ĐỒ ÁN MÔN HỌC**

**LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG**

**PROTOTYPE DESIGN PATTERN**

Giáo viên hướng dẫn : GV.Trần Quang Huy

Nhóm SV thực hiện : 47.01.104.200 – Hồ Ngọc Hưng Thịnh

: 47.01.104.231 – Phạm Anh Tùng

Mã lớp học phần : COMP130402

TP. HCM, tháng 5 năm 2023

LỜI CẢM ƠN

Chúng em xin gửi lời cảm ơn chân thành đến nhóm giảng viên và bạn bè đã hỗ trợ và đồng hành cùng chúng em trong quá trình thực hiện đồ án về Prototype Design Pattern này.

Đầu tiên, chúng em xin chân thành cảm ơn các giảng viên, những người đã cung cấp kiến thức nền tảng vững chắc và hướng dẫn tận tình giúp chúng em hiểu rõ hơn về các khái niệm thiết kế phần mềm và Design Pattern. Sự nhiệt tình và động viên của các giảng viên đã giúp chúng em vượt qua những khó khăn và hoàn thành đồ án một cách thành công.

Chúng em cũng xin bày tỏ lòng biết ơn đến bạn bè trong lớp đã chia sẻ kinh nghiệm và hỗ trợ trong quá trình thực hiện đồ án này. Những lời khuyên và đóng góp của các bạn đã góp phần làm cho đồ án của chúng em hoàn thiện hơn và đạt được kết quả tốt.

Cuối cùng, chúng em muốn cảm ơn gia đình và người thân đã luôn động viên, ủng hộ và chia sẻ niềm vui cùng chúng em trong suốt thời gian thực hiện đồ án. Sự hỗ trợ và yêu thương của gia đình là động lực vô cùng quan trọng giúp chúng em vượt qua mọi khó khăn.

Lời cảm ơn chân thành này dành riêng cho tất cả mọi người, nhờ có sự đóng góp và hỗ trợ của các bạn, đồ án của chúng em đã trở nên hoàn thiện hơn và đạt được kết quả mà chúng em hằng mong đợi.

Xin chân thành cảm ơn!

ĐÁNH GIÁ NHẬN XÉT CỦA GIẢNG VIÊN

Điểm số:

Điểm chữ:

TP.Hồ Chí Minh, Ngày tháng năm 2023

DANH MỤC HÌNH ẢNH

[[Hình 1. Các thành phần chính. 6](#_Toc141380987)](#_Toc141553249)

[[Hình 2. class DongVatPrototype(). 12](#_Toc141380987)](#_Toc141553250)

[[Hình 3. class Meo(). 13](#_Toc141380987)](#_Toc141553251)

[[Hình 4. class Cho(). 14](#_Toc141380987)](#_Toc141553252)

[[Hình 5. class QuanLyDongVatPrototype() 15](#_Toc141380987)](#_Toc141553253)

[[Hình 6. void SuaThongTin() 16](#_Toc141380987)](#_Toc141553254)

[[Hình 7. Hàm Main() 18](#_Toc141380987)](#_Toc141553255)

[13](#_Toc141380987)

**Mục lục**

[I. Giới thiệu. 1](#_Toc141553294)

[1. Mô tả đồ án. 1](#_Toc141553295)

[2. Mục tiêu. 1](#_Toc141553296)

[3. Khái niệm về Design pattern. 2](#_Toc141553297)

[4. Giới thiệu về Prototype Design Pattern. 2](#_Toc141553298)

[II. NGUYÊN LÝ HOẠT ĐỘNG CỦA PROTOTYPE DESIGN PATTERN. 3](#_Toc141553299)

[1. Tại sao cần sử dụng Prototype Design Pattern? 3](#_Toc141553300)

[2. Cách hoạt động của Prototype Design Pattern. 4](#_Toc141553301)

[3. So sánh với các design pattern khác. 5](#_Toc141553302)

[III. Các thành phần chính. 6](#_Toc141553303)

[1. Prototype Interface/Abstract class. 6](#_Toc141553304)

[2. Concrete Prototypes. 7](#_Toc141553305)

[3. Client. 7](#_Toc141553306)

[IV. ƯU ĐIỂM VÀ NHƯỢC ĐIỂM CỦA PROTOTYPE DESIGN PATTERN. 7](#_Toc141553307)

[1. Ưu điểm. 7](#_Toc141553308)

[2. Nhược điểm. 8](#_Toc141553309)

[V. CÁC TRƯỜNG HỢP SỬ DỤNG 9](#_Toc141553310)

[1. Khi nào nên sử dụng Prototype Design Pattern? 9](#_Toc141553311)

[2. Khi nào nên tránh sử dụng Prototype Design Pattern? 10](#_Toc141553312)

[VI. VÍ DỤ MINH HỌA. 11](#_Toc141553313)

[VII. KẾT LUẬN 19](#_Toc141553314)

[1. Tổng kết về Prototype Design Pattern. 19](#_Toc141553315)

[2. Tầm quan trọng và ứng dụng trong thực tế. 19](#_Toc141553316)

[VIII. TÀI LIỆU THAM KHẢO 20](#_Toc141553317)

# Giới thiệu.

## Mô tả đồ án.

Đồ án này tập trung nghiên cứu và trình bày về "Prototype Design Pattern" - một trong những mẫu thiết kế phần mềm quan trọng trong lập trình hướng đối tượng. Đây là một trong những mẫu thiết kế tạo ra các đối tượng mới bằng cách sao chép các đối tượng đã tồn tại, giúp tránh việc tạo mới từng đối tượng thủ công.

## Mục tiêu.

* Hiểu rõ về Prototype Design Pattern: Nắm vững nguyên lý hoạt động, cơ chế sao chép đối tượng, và cách Prototype Design Pattern giúp tạo mới đối tượng hiệu quả.
* Thành thạo cách triển khai Prototype Design Pattern: Đảm bảo có khả năng xây dựng và sử dụng mẫu thiết kế Prototype trong các dự án phần mềm thực tế bằng cách hiện thực mã nguồn trong các ngôn ngữ lập trình khác nhau.
* So sánh và phân tích ưu điểm, nhược điểm của Prototype Design Pattern: Đánh giá độ hiệu quả, tính linh hoạt và hạn chế của mẫu thiết kế này so với các phương pháp tạo đối tượng truyền thống.
* Xây dựng các ví dụ minh họa chân thực: Tạo ra các ví dụ cụ thể và sử dụng chúng để minh họa cách Prototype Design Pattern giúp tối ưu hóa việc tạo mới đối tượng và giảm thiểu tải cho hệ thống.
* Hiểu ứng dụng thực tế của Prototype Design Pattern: Khám phá các trường hợp sử dụng phổ biến của mẫu thiết kế này trong thế giới thực và hiểu rõ lợi ích của việc áp dụng nó vào các dự án phần mềm.
* Cung cấp tài liệu tham khảo đáng tin cậy: Tìm hiểu từ các nguồn uy tín và đáng tin cậy để cung cấp dữ liệu hỗ trợ cho nghiên cứu và phân tích trong đồ án.
* Trình bày một cách logic và rõ ràng: Đảm bảo cách trình bày đồ án được cấu trúc mạch lạc, dễ hiểu, và phù hợp với đối tượng độc giả có kiến thức cơ bản về lập trình hướng đối tượng.

## Khái niệm về Design pattern.

Design pattern (mẫu thiết kế) là một giải pháp tổng quát được phát triển và cố định cho một vấn đề thường gặp trong lĩnh vực thiết kế phần mềm. Nó là kết quả của kinh nghiệm tích lũy và các nguyên tắc thiết kế tốt đã được chứng minh trong thực tế, giúp cải thiện tính mở rộng, tái sử dụng và hiệu quả của mã nguồn.

Một design pattern không phải là một mô hình chương trình cụ thể hay một mã nguồn hoàn chỉnh, mà là một hướng dẫn thông qua việc cung cấp một cấu trúc tổ chức và mô tả các quy tắc, cách thức sử dụng các lớp, đối tượng và các thành phần trong một tình huống cụ thể. Design pattern giúp các nhà phát triển giải quyết các vấn đề phức tạp bằng cách cung cấp một cách tiếp cận chuẩn hóa và dễ hiểu.

Design pattern chia thành ba loại chính:

* Cấu trúc (Structural Patterns): Tập trung vào cách các lớp và đối tượng được tổ chức để hỗ trợ sự tương tác giữa chúng một cách linh hoạt. Ví dụ: Adapter, Composite, Decorator, Proxy...
* Hành vi (Behavioral Patterns): Tập trung vào cách các đối tượng tương tác và phân chia trách nhiệm của chúng một cách hiệu quả. Ví dụ: Observer, Strategy, Command, Iterator...
* Tạo (Creational Patterns): Tập trung vào cách tạo đối tượng và ẩn thông tin về việc tạo đối tượng, giúp giảm sự phụ thuộc và tăng tính mở rộng của hệ thống. Ví dụ: Singleton, Factory Method, Prototype, Builder...

## Giới thiệu về Prototype Design Pattern.

Prototype Design Pattern (Mẫu thiết kế Nguyên mẫu) là một trong số các mẫu thiết kế sáng tạo (Creational Design Pattern) trong lập trình hướng đối tượng. Mục tiêu chính của Prototype là tạo ra các đối tượng mới bằng cách sao chép (clone) các đối tượng đã tồn tại thay vì tạo mới từng đối tượng thủ công.

Ý tưởng chính của Prototype Design Pattern là sử dụng một đối tượng ban đầu (prototype) để tạo ra các bản sao của nó bằng cách sao chép. Các bản sao này có thể thay đổi dữ liệu của mình mà không ảnh hưởng đến prototype ban đầu. Việc sao chép này có thể được thực hiện theo hai cách: sao chép sâu (deep copy) hoặc sao chép hướng tham chiếu (shallow copy) tùy thuộc vào yêu cầu và đặc tính của đối tượng.

Prototype Design Pattern thường được sử dụng trong các trường hợp khi việc tạo mới đối tượng có chi phí cao hoặc cần tái sử dụng các đối tượng đã có. Nó cũng là một công cụ hữu ích để tạo ra các bản sao của đối tượng trong quá trình thử nghiệm và phát triển.

# NGUYÊN LÝ HOẠT ĐỘNG CỦA PROTOTYPE DESIGN PATTERN.

## Tại sao cần sử dụng Prototype Design Pattern?

Giảm tải và tối ưu hóa hiệu năng: Việc tạo mới các đối tượng phức tạp có thể tốn nhiều thời gian và tài nguyên. Prototype cho phép sao chép (clone) các đối tượng đã tồn tại thay vì tạo mới từng đối tượng, giúp giảm tải cho hệ thống và cải thiện hiệu năng chương trình.

Tái sử dụng đối tượng: Prototype giúp tái sử dụng các đối tượng đã có một cách dễ dàng. Thay vì việc viết lại mã để tạo mới đối tượng tương tự, chúng ta có thể sao chép các đối tượng có sẵn và chỉ cần thay đổi dữ liệu cần thiết.

Tối ưu hóa xử lý tạo đối tượng: Trong một số trường hợp, việc khởi tạo đối tượng có chi phí cao, chẳng hạn như đọc dữ liệu từ cơ sở dữ liệu hoặc kết nối mạng. Prototype giúp tiết kiệm thời gian và tài nguyên bằng cách tạo ra các bản sao của đối tượng thay vì tạo mới từ đầu.

Đảm bảo tính linh hoạt và mở rộng: Prototype giúp tạo ra các biến thể mới của đối tượng dễ dàng. Điều này làm cho chúng ta có thể mở rộng chức năng của đối tượng một cách linh hoạt mà không cần thay đổi cấu trúc gốc.

Tiếp cận tạo đối tượng thống nhất: Prototype cung cấp một cách tiếp cận chung và đơn giản trong việc tạo đối tượng. Điều này giúp giảm sự phức tạp và đảm bảo sự nhất quán trong mã nguồn.

Sử dụng trong quá trình thử nghiệm và phát triển: Prototype rất hữu ích khi cần tạo ra các bản sao của đối tượng để thử nghiệm và phát triển mà không làm ảnh hưởng đến đối tượng gốc.

## Cách hoạt động của Prototype Design Pattern.

Prototype Design Pattern hoạt động bằng cách sử dụng một đối tượng ban đầu (prototype) để tạo ra các bản sao của nó thông qua quá trình sao chép (clone). Điều này cho phép chúng ta tạo mới các đối tượng một cách hiệu quả mà không cần tạo mới từng đối tượng thủ công. Các bản sao này sau đó có thể thay đổi dữ liệu của mình mà không ảnh hưởng đến đối tượng gốc.

Quá trình hoạt động của Prototype Design Pattern bao gồm các bước sau:

* Xác định giao diện hoặc lớp Prototype: Đầu tiên, chúng ta xác định một giao diện hoặc lớp prototype, đại diện cho các đối tượng cần được sao chép. Giao diện này thường bao gồm một phương thức "clone" để thực hiện việc sao chép.
* Hiện thực lớp Concrete Prototype: Sau đó, chúng ta hiện thực các lớp cụ thể (Concrete Prototype) từ giao diện hoặc lớp Prototype. Mỗi lớp Concrete Prototype sẽ cung cấp thực thi cho phương thức "clone".
* Tạo và lưu trữ các đối tượng Prototype: Trong quá trình khởi tạo, chúng ta tạo và lưu trữ các đối tượng prototype ban đầu. Các đối tượng này có thể được tạo một lần duy nhất và sau đó được sử dụng để tạo ra các bản sao sau này.
* Tạo bản sao thông qua phương thức "clone": Khi cần tạo mới đối tượng, chúng ta sử dụng phương thức "clone" của đối tượng prototype để sao chép nó và nhận được một bản sao hoàn toàn độc lập từ đối tượng gốc.
* Thay đổi dữ liệu của bản sao (tuỳ chọn): Sau khi nhận được bản sao, chúng ta có thể thay đổi dữ liệu của nó mà không ảnh hưởng đến đối tượng gốc.

## So sánh với các design pattern khác.

* Factory Method Pattern:
* Factory Method tập trung vào việc tạo ra các đối tượng dựa trên một giao diện chung, trong khi Prototype tạo mới các đối tượng bằng cách sao chép các đối tượng đã tồn tại.
* Factory Method sử dụng lớp con để quyết định lớp nào được khởi tạo, trong khi Prototype sử dụng phương thức "clone" để tạo mới các bản sao của đối tượng.
* Singleton Pattern:
* Singleton nhằm mục đích đảm bảo rằng một lớp chỉ có duy nhất một thể hiện, trong khi Prototype cho phép tạo ra nhiều bản sao của một đối tượng.
* Singleton thường được sử dụng khi chúng ta muốn chia sẻ cùng một thể hiện trên toàn ứng dụng, trong khi Prototype hữu ích khi muốn tái sử dụng và thay đổi các đối tượng tùy ý.
* Builder Pattern:
* Builder nhằm mục đích xây dựng các đối tượng phức tạp bước từng bước, trong khi Prototype cho phép sao chép các đối tượng đã có.
* Builder thường được sử dụng khi chúng ta muốn xây dựng đối tượng theo từng bước một, trong khi Prototype được sử dụng để sao chép đối tượng đã tồn tại một cách nhanh chóng và linh hoạt.
* Abstract Factory Pattern:
* Abstract Factory cung cấp một giao diện để tạo ra một họ các đối tượng có liên quan, trong khi Prototype tập trung vào việc sao chép các đối tượng.
* Abstract Factory thường được sử dụng để tạo ra các đối tượng liên quan với nhau, trong khi Prototype giúp tạo mới các đối tượng một cách linh hoạt và hiệu quả.
* Singleton vs Prototype:
* Singleton đảm bảo rằng một lớp chỉ có một thể hiện duy nhất trong toàn ứng dụng, trong khi Prototype cho phép tạo ra nhiều bản sao của một đối tượng.
* Singleton thường được sử dụng khi cần chia sẻ cùng một thể hiện, trong khi Prototype được sử dụng để tái sử dụng và tạo mới các đối tượng linh hoạt.

# Các thành phần chính.

A screenshot of a computer

Description automatically generated

Hình . Các thành phần chính.

## Prototype Interface/Abstract class.

Prototype Interface/Abstract class là giao diện hoặc lớp trừu tượng trong Prototype Design Pattern. Nó đại diện cho các đối tượng cần được sao chép và định nghĩa phương thức "clone" để thực hiện việc sao chép đối tượng. Khi các lớp con thực hiện Prototype Interface/Abstract class, chúng sẽ cung cấp thực thi cụ thể cho phương thức "clone" và quyết định cách thức sao chép đối tượng. Thành phần này không chứa mã cụ thể của sao chép, điều này giúp giữ cho quá trình sao chép linh hoạt và dễ dàng mở rộng theo yêu cầu của ứng dụng.

## Concrete Prototypes.

Các Concrete Prototypes là các lớp cụ thể trong Prototype Design Pattern, được hiện thực từ Prototype Interface hoặc kế thừa từ lớp trừu tượng Prototype Abstract class. Mỗi lớp Concrete Prototype cung cấp thực thi cụ thể cho phương thức "clone", đảm bảo việc sao chép đối tượng của riêng nó. Điều này cho phép mỗi lớp Concrete Prototype thực hiện cách sao chép phù hợp với đặc tính và thuộc tính của lớp đó. Nhờ đó, Prototype Design Pattern cho phép tạo mới các đối tượng một cách linh hoạt và hiệu quả thông qua việc sử dụng các Concrete Prototypes để sao chép các đối tượng đã tồn tại.

## Client.

Client là thành phần quan trọng trong Prototype Design Pattern, đại diện cho lớp hoặc đối tượng sử dụng Prototype Interface/Abstract class để tạo mới các đối tượng. Client không cần biết cụ thể về cách sao chép đối tượng, mà chỉ gửi yêu cầu sao chép tới Prototype. Việc sử dụng Prototype Design Pattern giúp giảm tải cho việc tạo mới đối tượng, thay vì tạo mới từng đối tượng phức tạp, Client có thể tái sử dụng và tạo mới các đối tượng đã có một cách hiệu quả. Nếu cần tạo mới các loại đối tượng mới, Client chỉ cần tạo lớp con mới từ Prototype Interface/Abstract class và triển khai phương thức "clone" theo cách thức mong muốn. Từ đó, Prototype Design Pattern giúp tạo mới các đối tượng một cách linh hoạt và hiệu quả, đồng thời giúp dễ dàng mở rộng hệ thống bằng cách thêm các loại đối tượng mới vào.

# ƯU ĐIỂM VÀ NHƯỢC ĐIỂM CỦA PROTOTYPE DESIGN PATTERN.

## Ưu điểm.

Tạo mới đối tượng hiệu quả: Prototype Design Pattern cho phép sao chép các đối tượng hiện có để tạo mới các đối tượng thay vì tạo mới từng đối tượng từ đầu. Việc sao chép đối tượng là một phép tính nhẹ nhàng hơn so với tạo mới đối tượng phức tạp, giúp tăng hiệu suất của ứng dụng.

Tối ưu hóa bộ nhớ: Khi chúng ta sử dụng Prototype Design Pattern để tạo mới các đối tượng, việc sao chép một đối tượng đã có sẽ giúp tiết kiệm bộ nhớ hơn so với việc tạo mới đối tượng từ đầu. Điều này đặc biệt hữu ích khi ta cần tạo nhiều đối tượng giống nhau hoặc có sự tương đồng cao.

Dễ dàng mở rộng hệ thống: Khi cần thêm các loại đối tượng mới, chúng ta chỉ cần tạo lớp con mới từ giao diện nguyên mẫu (Prototype Interface) và triển khai phương thức clone theo cách thức mong muốn. Việc này giúp dễ dàng mở rộng hệ thống và thêm các đối tượng mới mà không làm ảnh hưởng đến cấu trúc hiện có.

Giảm tải khi tạo mới đối tượng phức tạp: Khi ta có một đối tượng có cấu trúc phức tạp và tạo mới nó đòi hỏi rất nhiều công việc, Prototype Design Pattern giúp chúng ta tiết kiệm thời gian và công sức bằng cách sao chép đối tượng đã có thay vì xây dựng lại từng thành phần.

Tính linh hoạt và tăng cường sự đa dạng: Prototype Design Pattern giúp cho việc tạo mới các đối tượng trở nên linh hoạt hơn và tăng cường sự đa dạng của các đối tượng trong hệ thống. Chúng ta có thể dễ dàng tạo nhiều phiên bản, biến thể của cùng một đối tượng mà không cần thay đổi cấu trúc gốc của nó.

Tránh sự phụ thuộc giữa client và các lớp cụ thể: Client không cần biết cụ thể về cách sao chép đối tượng, mà chỉ gửi yêu cầu sao chép tới Prototype. Điều này giúp giảm sự phụ thuộc giữa client và các lớp cụ thể, tăng tính chất thay đổi và tái sử dụng của hệ thống.

## Nhược điểm.

Khó hiểu: Prototype Design Pattern có thể khá phức tạp và khó hiểu đối với những người mới tiếp xúc với nó. Đặc biệt là khi xử lý các đối tượng có cấu trúc phức tạp và có mối quan hệ phức tạp với nhau, việc triển khai và sử dụng pattern có thể gây khó khăn.

Đối tượng con phải triển khai clone(): Một trong nhược điểm của Prototype Design Pattern là tất cả các lớp con cần phải triển khai phương thức clone(). Nếu quên hoặc không triển khai đúng phương thức clone() trong lớp con, việc tạo mới các đối tượng sẽ không thành công hoặc có thể dẫn đến lỗi.

Sự cần thiết về xây dựng thêm: Prototype Design Pattern đôi khi yêu cầu lớp con cần có một constructor để khởi tạo đối tượng mẫu. Việc này có thể tạo thêm sự ràng buộc và đòi hỏi việc xây dựng đối tượng mẫu phải được hoàn thiện trước khi sử dụng pattern.

Chi phí của việc sao chép: Trong một số trường hợp, việc sao chép đối tượng có thể tốn kém về mặt tài nguyên, đặc biệt là khi các đối tượng có cấu trúc phức tạp. Điều này có thể ảnh hưởng đến hiệu suất của ứng dụng.

Sự đồng nhất của các đối tượng sao chép: Khi sao chép một đối tượng, đôi khi các đối tượng sao chép có thể chia sẻ các tham chiếu đến cùng một đối tượng dữ liệu, dẫn đến tình huống không mong muốn khi các thay đổi được thực hiện trên một đối tượng có thể ảnh hưởng đến các đối tượng khác. Điều này yêu cầu cẩn trọng trong việc sao chép và quản lý dữ liệu của các đối tượng.

# CÁC TRƯỜNG HỢP SỬ DỤNG

## Khi nào nên sử dụng Prototype Design Pattern?

Khi cần tạo mới các đối tượng có cấu trúc phức tạp: Prototype Design Pattern giúp tiết kiệm thời gian và công sức khi sao chép các đối tượng có cấu trúc phức tạp thay vì tạo mới từ đầu. Điều này đặc biệt hữu ích khi các đối tượng có nhiều thành phần phức tạp và tạo mới chúng là công việc tốn kém.

Khi cần tạo nhiều phiên bản, biến thể của một đối tượng: Prototype Design Pattern cho phép dễ dàng sao chép và tạo mới các phiên bản, biến thể của cùng một đối tượng mẫu mà không cần thay đổi cấu trúc gốc của nó. Điều này giúp tăng tính linh hoạt và đa dạng của các đối tượng trong hệ thống.

Khi muốn tạo mới các đối tượng một cách hiệu quả và tối ưu hóa bộ nhớ: Prototype Design Pattern giúp tạo mới các đối tượng một cách hiệu quả bằng cách sao chép các đối tượng đã có thay vì tạo mới từ đầu. Việc sao chép đối tượng là một phép tính nhẹ nhàng hơn, giúp tối ưu hóa bộ nhớ và tăng hiệu suất của ứng dụng.

Khi cần dễ dàng mở rộng hệ thống bằng cách thêm các loại đối tượng mới: Prototype Design Pattern giúp dễ dàng mở rộng hệ thống bằng cách thêm các loại đối tượng mới vào. Chúng ta chỉ cần tạo lớp con mới từ giao diện nguyên mẫu (Prototype Interface) và triển khai phương thức clone theo cách thức mong muốn, mà không ảnh hưởng đến cấu trúc hiện có.

Khi muốn tránh sự phụ thuộc giữa client và các lớp cụ thể: Prototype Design Pattern giúp giảm sự phụ thuộc giữa client và các lớp cụ thể. Client không cần biết cụ thể về cách sao chép đối tượng, mà chỉ gửi yêu cầu sao chép tới Prototype, giúp tăng tính chất thay đổi và tái sử dụng của hệ thống.

Tóm lại, Prototype Design Pattern thường được sử dụng khi cần tạo mới các đối tượng một cách linh hoạt, hiệu quả và dễ dàng mở rộng hệ thống bằng cách thêm các loại đối tượng mới vào. Nó hữu ích trong các tình huống có đối tượng có cấu trúc phức tạp hoặc cần tạo nhiều phiên bản, biến thể của một đối tượng mẫu.

## Khi nào nên tránh sử dụng Prototype Design Pattern?

Khi các đối tượng có cấu trúc đơn giản và không có nhiều thuộc tính: Nếu các đối tượng trong hệ thống có cấu trúc đơn giản và không có nhiều thuộc tính, việc sử dụng Prototype Design Pattern có thể làm cho mã nguồn trở nên phức tạp và không cần thiết. Trong trường hợp này, việc tạo mới các đối tượng trực tiếp thông qua constructor có thể là một cách đơn giản và hiệu quả hơn.

Khi cần tạo mới các đối tượng không liên quan đến một đối tượng mẫu cụ thể: Prototype Design Pattern nên được sử dụng khi cần tạo mới các đối tượng dựa trên một đối tượng mẫu cụ thể. Nếu không có nhu cầu này và các đối tượng không có mối quan hệ hoặc tương tự nhau, việc sử dụng Prototype Design Pattern không hợp lý và có thể làm mã nguồn trở nên phức tạp và không cần thiết.

Khi việc sao chép đối tượng có khả năng gây lỗi hoặc ảnh hưởng đến tính toàn vẹn dữ liệu: Prototype Design Pattern đôi khi yêu cầu việc sao chép đối tượng phải được thực hiện một cách cẩn thận để đảm bảo tính toàn vẹn dữ liệu và tránh lỗi khi sao chép các đối tượng có mối quan hệ phức tạp. Trong những trường hợp này, việc sử dụng Prototype Design Pattern có thể gây khó khăn và tăng nguy cơ gây lỗi.

Khi cần tạo mới các đối tượng với quyền truy cập riêng tư và phạm vi hoạt động riêng biệt: Prototype Design Pattern không phù hợp khi cần tạo mới các đối tượng với quyền truy cập riêng tư hoặc phạm vi hoạt động riêng biệt. Trong trường hợp này, việc tạo mới các đối tượng thông qua constructor có thể giúp duy trì tính riêng tư và phạm vi hoạt động của các đối tượng.

Khi mã nguồn không cần thiết phải tái sử dụng: Prototype Design Pattern được sử dụng để tái sử dụng các đối tượng mẫu và giảm tải việc tạo mới đối tượng. Tuy nhiên, nếu trong mã nguồn không có nhu cầu tái sử dụng các đối tượng mẫu hoặc việc tạo mới các đối tượng không gây tải lớn, việc sử dụng Prototype Design Pattern có thể là không cần thiết và không hợp lý.

Tóm lại, Prototype Design Pattern nên được sử dụng một cách cân nhắc và phù hợp với từng tình huống cụ thể trong việc tạo mới các đối tượng. Nếu không có nhu cầu tái sử dụng đối tượng mẫu hoặc các đối tượng có cấu trúc đơn giản, việc sử dụng Prototype Design Pattern có thể không hợp lý và làm mã nguồn trở nên phức tạp.

# VÍ DỤ MINH HỌA.

Prototype Design Pattern trong ngôn ngữ lập trình C++ cho phép sao chép và tạo mới các đối tượng một cách hiệu quả. Dưới đây là một ví dụ minh họa cụ thể về cách triển khai Prototype Design Pattern trong C++:

1. **DongVatPrototype**: Đây là lớp giao diện (interface) nguyên mẫu (prototype) cho các đối tượng Động vật (DongVat). Nó chứa các phương thức ảo (virtual) clone, tiengKeu, xuatThongTin, nhapThongTin, và suaThongTin, mô tả những hành vi cơ bản mà tất cả các đối tượng Động vật cần triển khai.

A screen shot of a computer code

Description automatically generated

Hình 2. class DongVatPrototype().

1. **Meo**: Là một lớp con (concrete class) triển khai giao diện DongVatPrototype để tạo mới các đối tượng Mèo (Meo). Lớp này có các thuộc tính riêng như ten (tên), tuoi (tuổi), canNang (cân nặng), và gioiTinh (giới tính), cùng với các phương thức triển khai từ giao diện nguyên mẫu.

A screenshot of a computer program

Description automatically generated

Hình 3. class Meo().

1. **Cho**: Là một lớp con khác triển khai giao diện DongVatPrototype để tạo mới các đối tượng Chó (Cho). Tương tự như lớp Meo, lớp này có các thuộc tính riêng và triển khai các phương thức từ giao diện nguyên mẫu.

A screenshot of a computer program

Description automatically generated

Hình 4. class Cho().

1. **QuanLyDongVatPrototype**: Lớp này chịu trách nhiệm quản lý các đối tượng nguyên mẫu (prototypes) của Mèo và Chó thông qua một unordered\_map (bản đồ không thứ tự). Trong hàm khởi tạo, nó tạo các đối tượng nguyên mẫu và lưu chúng vào bản đồ theo các khóa (keys) tương ứng.

A screenshot of a computer program

Description automatically generated

Hình 5. class QuanLyDongVatPrototype()

1. **Hàm suaThongTin()**: trong chương trình cho phép người dùng sửa thông tin của đối tượng động vật (mèo hoặc chó). Đầu tiên, nó sẽ yêu cầu người dùng nhập các thông tin mới bằng cách hỏi tên, tuổi và cân nặng mới. Nếu người dùng nhập 0 ở bất kỳ thông tin nào, nghĩa là muốn giữ nguyên thông tin ban đầu của đối tượng, thì thông tin mới sẽ không được cập nhật và giá trị của thông tin đó sẽ được giữ nguyên. Sau đó, thông tin mới (nếu có) sẽ được cập nhật vào đối tượng.

A screenshot of a computer program

Description automatically generated

Hình 6. void SuaThongTin()

1. **Hàm** **main()**: Trong hàm main, chúng ta tạo một đối tượng QuanLyDongVatPrototype để quản lý các đối tượng nguyên mẫu Mèo và Chó. Sau đó, chúng ta tạo mới các đối tượng conMeo và conCho bằng cách sao chép từ đối tượng nguyên mẫu theo khóa "meo" và "cho". Tiếp theo, chúng ta yêu cầu nhập thông tin cho cả conMeo và conCho thông qua các phương thức nhapThongTin() được triển khai từ giao diện nguyên mẫu.

Tiếp theo, chúng ta xuất thông tin của cả conMeo và conCho bằng cách gọi phương thức xuatThongTin() từ giao diện nguyên mẫu.

Sau đó, chúng ta sử dụng các phương thức suaThongTin() từ giao diện nguyên mẫu để sửa thông tin của conMeo và conCho. Cuối cùng, chúng ta xuất thông tin của cả conMeo và conCho lần nữa sau khi đã được sửa.

A screen shot of a computer

Description automatically generated

Hình 7. Hàm Main()

# KẾT LUẬN

## Tổng kết về Prototype Design Pattern.

Prototype Design Pattern là một mẫu thiết kế quan trọng trong lập trình hướng đối tượng. Nó cho phép sao chép các đối tượng hiện có để tạo mới các đối tượng một cách linh hoạt và hiệu quả. Điều này giúp giảm tải cho việc tạo mới đối tượng và tăng tính linh hoạt của hệ thống.

Giao diện nguyên mẫu (Prototype Interface/Abstract class) xác định phương thức clone để sao chép các đối tượng. Các lớp cụ thể (Concrete Prototypes) triển khai phương thức clone để tạo mới các phiên bản, biến thể của đối tượng mẫu. Client sử dụng Prototype Interface/Abstract class để tạo mới các đối tượng một cách linh hoạt thông qua phương thức clone, không cần biết cụ thể về cách sao chép đối tượng.

Prototype Design Pattern giúp tăng tính linh hoạt và hiệu quả khi cần tạo mới nhiều đối tượng có cấu trúc phức tạp. Nó cũng giúp dễ dàng mở rộng hệ thống bằng cách thêm các loại đối tượng mới vào mà không cần thay đổi cấu trúc hiện có.

Tuy nhiên, cần cẩn thận khi sao chép đối tượng để đảm bảo tính toàn vẹn dữ liệu và tránh lỗi khi sao chép các đối tượng có mối quan hệ phức tạp. Nếu các đối tượng có cấu trúc đơn giản hoặc không có nhu cầu tái sử dụng đối tượng mẫu, việc sử dụng Prototype Design Pattern có thể là không cần thiết và phức tạp.

## Tầm quan trọng và ứng dụng trong thực tế.

Prototype Design Pattern có tầm quan trọng và ứng dụng rộng rãi trong thực tế, đặc biệt là trong lập trình hướng đối tượng và các hệ thống phần mềm lớn. Dưới đây là một số điểm nổi bật về tầm quan trọng và ứng dụng của Prototype Design Pattern:

* Tăng hiệu suất và giảm tải cho việc tạo mới đối tượng: Thay vì tạo mới các đối tượng phức tạp từ đầu, Prototype Design Pattern cho phép sao chép các đối tượng đã có để tạo mới các đối tượng mới. Điều này giúp tiết kiệm thời gian và công sức, giảm tải cho việc tạo mới đối tượng, đồng thời tăng hiệu suất của ứng dụng.
* Tăng tính linh hoạt và tái sử dụng mã nguồn: Prototype Design Pattern giúp tăng tính linh hoạt và dễ dàng mở rộng hệ thống bằng cách thêm các loại đối tượng mới vào. Chúng ta chỉ cần tạo lớp con mới từ giao diện nguyên mẫu (Prototype Interface) và triển khai phương thức clone theo cách thức mong muốn, mà không ảnh hưởng đến cấu trúc hiện có. Điều này giúp tái sử dụng mã nguồn một cách hiệu quả.
* Đơn giản hóa quá trình tạo mới đối tượng: Prototype Design Pattern giúp đơn giản hóa quá trình tạo mới các đối tượng phức tạp. Client chỉ cần gửi yêu cầu sao chép tới Prototype, mà không cần biết cụ thể về cách sao chép đối tượng. Điều này giúp giảm sự phụ thuộc giữa client và các lớp cụ thể, làm mã nguồn trở nên dễ dàng quản lý và bảo trì.
* Sử dụng trong việc tạo mới các đối tượng có cấu trúc phức tạp: Khi các đối tượng có cấu trúc phức tạp với nhiều thành phần và tạo mới chúng là công việc tốn kém, Prototype Design Pattern là lựa chọn lý tưởng. Việc sao chép các đối tượng đã có giúp tạo mới các đối tượng một cách nhanh chóng và hiệu quả hơn.
* Giúp tạo mới các đối tượng với các biến thể và phiên bản khác nhau: Nếu cần tạo mới các phiên bản, biến thể của một đối tượng mẫu, Prototype Design Pattern là một giải pháp tốt. Client có thể sao chép đối tượng mẫu và sau đó điều chỉnh các thuộc tính để tạo ra các phiên bản hoàn toàn khác nhau.

Tóm lại, Prototype Design Pattern có tầm quan trọng cao trong việc tăng tính linh hoạt, hiệu suất và tái sử dụng mã nguồn trong lập trình hướng đối tượng. Việc sử dụng mẫu thiết kế này giúp giảm tải cho việc tạo mới đối tượng, đơn giản hóa quá trình tạo mới đối tượng và tăng khả năng mở rộng hệ thống một cách dễ dàng.

# TÀI LIỆU THAM KHẢO

[1] Refactoring.Guru. <https://refactoring.guru/design-patterns>

[2] Design Patterns for Dummies, Steve Holzner, PhD

[3] Head First, Eric Freeman

[4] Gang of Four Design Patterns 4.0

[5] Dive into Design Pattern