Nhận xét của giáo viên

Nội dung

[1. Tổng quan 6](#_Toc469764215)

[1.1. Giới thiệu 6](#_Toc469764216)

[1.2. Phạm vi báo cáo 6](#_Toc469764217)

[1.3. Các khái niệm OOP được sử dụng 6](#_Toc469764218)

[2. Giới thiệu về mẫu thiết kế 8](#_Toc469764219)

[2.1. Khái niệm mẫu thiết kế: 8](#_Toc469764220)

[2.2. Lịch sử ra đời 8](#_Toc469764221)

[2.3. Phân loại 9](#_Toc469764222)

[2.3.1. Mẫu kiến tạo 9](#_Toc469764223)

[2.3.2. Mẫu cấu trúc 10](#_Toc469764224)

[2.3.3. Mẫu hành vi 11](#_Toc469764225)

[2.4. Lợi ích khi sử dụng mẫu 13](#_Toc469764226)

[2.5. Cấu trúc trình bày mẫu 14](#_Toc469764227)

[2.6. Một số nguyên tắc 14](#_Toc469764228)

[3. Các mẫu Thiết kế GoF 15](#_Toc469764229)

[3.1. Mẫu Singleton 15](#_Toc469764230)

[3.1.1. Tên, phân loại, bí danh 15](#_Toc469764231)

[3.1.2. Mục đích, ý định 15](#_Toc469764232)

[3.1.3. Động lực sử dụng 15](#_Toc469764233)

[3.1.4. Khả năng ứng dụng 15](#_Toc469764234)

[3.1.5. Cấu trúc 15](#_Toc469764235)

[3.1.6. Các thành viên 15](#_Toc469764236)

[3.1.7. Sự công tác 15](#_Toc469764237)

[3.1.8. Các hệ quả mang lại 15](#_Toc469764238)

[3.1.9. Các chú ý liên quan đến cài đặt 15](#_Toc469764239)

[3.1.10. Các ví dụ về hệ thống thực tế 15](#_Toc469764240)

[3.1.11. Các mẫu liên quan 15](#_Toc469764241)

[3.1.12. Mã nguồn minh họa 15](#_Toc469764242)

[3.2. Mẫu Proxy 15](#_Toc469764243)

[3.2.1. Tên, phân loại, bí danh 15](#_Toc469764244)

[3.2.2. Mục đích, ý định 15](#_Toc469764245)

[3.2.3. Động lực sử dụng 15](#_Toc469764246)

[3.2.4. Khả năng ứng dụng 15](#_Toc469764247)

[3.2.5. Cấu trúc 15](#_Toc469764248)

[3.2.6. Các thành viên 15](#_Toc469764249)

[3.2.7. Sự công tác 15](#_Toc469764250)

[3.2.8. Các hệ quả mang lại 15](#_Toc469764251)

[3.2.9. Các chú ý liên quan đến cài đặt 15](#_Toc469764252)

[3.2.10. Các ví dụ về hệ thống thực tế 15](#_Toc469764253)

[3.2.11. Các mẫu liên quan 15](#_Toc469764254)

[3.2.12. Mã nguồn minh họa 15](#_Toc469764255)

[3.3. Mẫu Template 15](#_Toc469764256)

[3.3.1. Tên, phân loại, bí danh 15](#_Toc469764257)

[3.3.2. Mục đích, ý định 15](#_Toc469764258)

[3.3.3. Động lực sử dụng 15](#_Toc469764259)

[3.3.4. Khả năng ứng dụng 15](#_Toc469764260)

[3.3.5. Cấu trúc 15](#_Toc469764261)

[3.3.6. Các thành viên 15](#_Toc469764262)

[3.3.7. Sự công tác 15](#_Toc469764263)

[3.3.8. Các hệ quả mang lại 16](#_Toc469764264)

[3.3.9. Các chú ý liên quan đến cài đặt 16](#_Toc469764265)

[3.3.10. Các ví dụ về hệ thống thực tế 16](#_Toc469764266)

[3.3.11. Các mẫu liên quan 16](#_Toc469764267)

[3.3.12. Mã nguồn minh họa 16](#_Toc469764268)

[3.4. Mẫu Façade 16](#_Toc469764269)

[3.4.1. Tên, phân loại, bí danh 16](#_Toc469764270)

[3.4.2. Mục đích, ý định 16](#_Toc469764271)

[3.4.3. Động lực sử dụng 16](#_Toc469764272)

[3.4.4. Khả năng ứng dụng 16](#_Toc469764273)

[3.4.5. Cấu trúc 16](#_Toc469764274)

[3.4.6. Các thành viên 16](#_Toc469764275)

[3.4.7. Sự công tác 16](#_Toc469764276)

[3.4.8. Các hệ quả mang lại 16](#_Toc469764277)

[3.4.9. Các chú ý liên quan đến cài đặt 16](#_Toc469764278)

[3.4.10. Các ví dụ về hệ thống thực tế 16](#_Toc469764279)

[3.4.11. Các mẫu liên quan 16](#_Toc469764280)

[3.4.12. Mã nguồn minh họa 16](#_Toc469764281)

[3.5. Mẫu Observer 16](#_Toc469764282)

[3.5.1. Tên, phân loại, bí danh 16](#_Toc469764283)

[3.5.2. Mục đích, ý định 16](#_Toc469764284)

[3.5.3. Động lực sử dụng 16](#_Toc469764285)

[3.5.4. Khả năng ứng dụng 16](#_Toc469764286)

[3.5.5. Cấu trúc 16](#_Toc469764287)

[3.5.6. Các thành viên 16](#_Toc469764288)

[3.5.7. Sự công tác 16](#_Toc469764289)

[3.5.8. Các hệ quả mang lại 16](#_Toc469764290)

[3.5.9. Các chú ý liên quan đến cài đặt 16](#_Toc469764291)

[3.5.10. Các ví dụ về hệ thống thực tế 16](#_Toc469764292)

[3.5.11. Các mẫu liên quan 16](#_Toc469764293)

[3.5.12. Mã nguồn minh họa 16](#_Toc469764294)

[4. Tổng kết 17](#_Toc469764295)

[5. Phụ lục 17](#_Toc469764296)

[6. Tài liệu tham khảo 19](#_Toc469764297)

1. **Tổng quan** 
   1. **Giới thiệu**

Quá trình phát triển phần mềm đi theo trình tự

* 1. **Phạm vi báo cáo**

Sử dụng các mẫu của GoF,

Trong phạm vi báo cáo này trình bày về 5 mẫu DP có tần suất sử dụng tương đối cao, được áp dụng trong nhiều lĩnh vực khác nhau, bao gồm : Façade, Template, Observe, Proxy và Singleton.

* 1. **Các khái niệm OOP được sử dụng**

Trong quá trình mô tả các mẫu thiết kế, có khá nhiều các khái niệm trong OOP (lập trình hướng đối tượng) được sử dụng lại. Một số thuật ngữ tương đối khó hiểu và dễ nhầm lẫn khi chuyển đổi từ tiếng Anh sang tiếng Vệt cũng sẽ được nói sơ qua nhằm phục vụ cho nội dung báo cáo.

1. Ngôn ngữ lập trình

Ra đời vào đầu thế kỉ XIX, cùng với sự phát triển của ngành công nghiệp chế tạo máy tính, hiểu biết của con người đối với lập trình cũng ngày càng tăng lên. Từ các chỉ thị máy được đục trên bằng đến các câu lệnh được máy tính biên dịch,

1. Hướng tiếp cận
2. Lập trình hướng đối tượng

* Abstraction:

Trong lập trình hướng đối tượng, đầu tiên ta sẽ gặp các khái niệm về sự trừu tượng và sự hiện thực. Sự trừu tượng hay Abstraction là khái niệm mô tả cách chúng ta nhìn nhận các đối tượng trong thực tế thành các thành phần trong mã nguồn của chúng ta. Ví dụ: một chiếc xe trong thực tế sẽ được chúng ta trừu tượng hóa thành một lớp tên là Xe trong C#. <<mã nguồn>>. Còn sự hiện thực là khái niệm mô tả cách thức một đối tượng sau khi trừu tượng hóa sẽ hoạt động như thế nào. Ví dụ Xe ở trên, nếu ta hiện thực lớp này, nó sẽ có các khả năng run, stop, ….

* Interface
* Implementation
* Inheritance
* Composition
* Is-a, Has-a
* Polymorphism
* Creation of object

1. **Giới thiệu về mẫu thiết kế** 
   1. **Khái niệm mẫu thiết kế:**

Trong quá trình thiết kế phần mềm, chúng ta có thể gặp phải một vài vấn đề khó giải quyết, hoặc cách giải quyết của chúng không phù hợp, hoặc giải quyết vấn đề không trọn vẹn, chúng ta có thể nghĩ đến các mẫu thiết kế phần mềm. Nói chung, mẫu thiết kế là một giải pháp chung được các lập trình viên và các nhà phát triển đi trước tìm ra và tổng hợp lại, chúng thường được áp dụng trong việc giải quyết một số vấn đề thường xảy ra trong quá trình chúng ta thiết kế phần mềm.

Nói một cách khác, ta có thể coi một mẫu thiết kế như một khuôn mẫu mô tả cách thức giải quyết một vấn đề mà ta có thể sử dụng trong nhiều tình huống khác nhau.

* 1. **Lịch sử ra đời**

Khái niệm mẫu thiết kế mà chúng ta đang dùng có nguồn gốc trong lĩnh vực kiến trúc xây dựng. Một kỹ sư người Áo là Christopher Alexander (sinh năm 1936) đã nhận thấy có rằng có nhiều thiết kế được sử dụng nhiều lần trong quá trình thiết kế các công trình kiến trúc. Ông đã xuất bản hai quyển sách có tầm ảnh hưởng lớn là “A Pattern Language : Towns, Buildings, Construction” (xuất bản năm 1977) và “The Timeless Way of Building” (xuất bản năm 1979). Trong các quyển sách này, ông đã mô tả các mẫu thiết kế kiến trúc “Mỗi mẫu mô tả một vấn đề mà chúng xảy ra lặp đi lặp lại trong môi trường của chúng tôi. Đồng thời mô tả giải pháp cốt lõi để giải quyết vấn đề đó, theo một cách mà bạn có thể sử dụng giải pháp này hàng triệu lần mà không cần phải giải quyết một vấn đề giống nhau hai lần.

Gần hai thập kỉ sau, các chuyên gia trong lĩnh vực phần mềm bắt đầu kết hợp các nguyên lý của Alexander và việc tạo ra các tài liệu hướng dẫn đầu tiên về mẫu thiết kế cho các nhà phát triển mới. Vào năm 1994, hội nghị đầu tiên về các mẫu thiết kế được tổ chức. Tên của hội nghị là Pattern Languages of Program Design (PLoP), tạm dịch là “Các ngôn ngữ mô mẫu của thiết kế chương trình”. Không lâu sau đó (1995), cuốn sách có tầm ảnh hưởng lớn nhất đến các mẫu thiết kế phần mềm là *Design Patterns: Elements of Reusable Object-Oriented Software* đã được xuất bản, nó giới thiệu 23 mẫu thiết kế đầu tiên, do bốn người Erich Gamma, Richard Helm, Ralph Johnson và John Vlissides đồng tác giả. Các tác giả sau này thường được nhắc đến với biệt danh *Gang of Four*  hay GoF. Kể từ đó, càng ngày càng có nhiều mẫu thiết kế được ghi chép và phân loại. Tuy nhiên, 23 mẫu GoF đầu tiên được biết đến và được ứng dụng nhiều nhất.

* 1. **Phân loại**

Các mẫu thiết kế được chia thành 3 loại chính: Mẫu kiến tạo, mẫu kiến trúc và mẫu hành vi.

* + 1. **Mẫu kiến tạo**

Mẫu kiến tạo (Creational Pattern) trừu tượng hóa quá trình khởi tạo. Chúng giúp chúng ta tạo nên một hệ thống độc lập với cách mà các đối tượng của hệ thống được tạo ra, kết hợp và thể hiện. Các mẫu kiến tạo có thể được chia thành 2 nhóm nhỏ hơn là các mẫu Kiến tạo Lớp (class-creation) và các mẫu Kiến tạo Đối tượng (object-creation). Trong khi mẫu Kiến tạo Lớp sử dụng kế thừa để thay đổi lớp cần được khởi tạo, mẫu Kiến tạo Đối tượng sẽ ủy nhiệm quá trình khởi tạo đến một đối tượng khác.

Mẫu kiến tạo có vai trò quan trọng bởi khi hệ thống phát triển phụ thuộc theo thời gian. Khi đó hệ thống cần được phát triển dựa vào sự kết hợp các đối tượng nhiều hơn sự kế thừa các lớp.

Khi điều đó xảy ra, việc hard-coding các tập cố định các hành vi sẽ chuyển thành việc định nghĩa một tập nhỏ hơn các hành vi cơ bản để chúng có thể được kết hợp thành bất kỳ các thành phần phức tạo nào khác nhau. Khi đó, việc tạo ra một đối tượng với các hành vi cụ thể yêu cầu nhiều công sức hơn so với việ chỉ đơn giản là khởi tạo một lớp.

Có hai chủ đề quan trọng trong các mẫu khởi tạo. Đầu tiên, chúng đóng gói tất cả các thông tin về các lớp con cụ thể mà hệ thống sẽ sử dụng. Thứ hai, chúng ẩn đi cách mà các thể hiện của lớp con được hệ thống sử dụng sẽ được tạo ra và kết hợp với nhau. Tất cả những gì mà các hệ thống sử dụng các đối tượng đó biết được là giao diện của chúng đã được định nghĩa bởi các lớp ảo.

Mẫu kiến tạo cung cấp cho chúng ta sự linh hoạt vô cùng lớn, trong việc đối tượng nào sẽ được tạo ra, cái gì sẽ tạo ra chúng, làm thế nào để chúng tạo ra, và khi nào. Mẫu kiến tạo cho phép chúng ta cấu hình một hệ thống với các đối tượng “sản phẩm” thuộc đủ các thể loại cấu trúc và hành vi khác nhau. Việc cấu hình có thể tĩnh (được quy định tại thời điểm biên dịch – compile-time) hay động (trong khi chương trình đang trong thời gian chạy – run-time).

Các mẫu kiến tạo của GoF bao gồm 5 mẫu, được giới thiệu sơ lược sau đây:

|  |  |
| --- | --- |
| Mẫu | Mô tả ngắn gọn |
| Abstract Factory | Cung cấp một interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần qui định lớp hay xác định lớp cụ thể (concrete) khi tạo mỗi đối tượng. |
| 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 |
| Singleton | Đảm bảo 1 class chỉ có 1 instance và cung cấp 1 điểm truy xuất toàn cục đến nó. |
| 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. |
| 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 |

* + 1. **Mẫu cấu trúc**

Các mẫu cấu trúc có liên quan đến cách thức cấu tạo nên các lớp và các đối tượng để chúng có thể tạo thành các cấu trúc lớn và phức tạp hơn bằng cách sử dụng tính kế thừa để tạo ra các giao diện hoặc các triển khai.

Thay vì tạo ra các giao diện và các triển khai, mẫu cấu trúc mô tả các cách để chúng ta có thể phối hợp các đối tượng để nhận ra các chức năng mới. Sự linh hoạt được thêm vào của việc kết hợp các đối tượng đến từ khả năng thay đổi sự kết hợp đó tại thời điểm chương trình đang chạy – run-time, một điều bất khả thi khi chúng ta kết hợp tĩnh các lớp.

Các mẫu cấu trúc của GoF bao gồm 7 mẫu, được giới thiệu sơ lược sau đây:

|  |  |
| --- | --- |
| Mẫu | Mô tả Ngắn gọn |
| Adapter | Do vấn đề tương thích, thay đổi interface của một lớp thành một interface khác phù hợp với yêu cầu từ phía sử dụng lớp. |
| Bridge | Tách rời sự trừu tượng của một vấn đề khỏi việc cài đặt; mục đích để cả hai bộ phận (trừu tượng và cài đặt) có thể thay đổi độc lập với nhau. |
| 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 duy nhất như nhau. Tạo quan hệ thứ bậc bao gộp giữa các đối tượng. |
| Decorator | Thêm các trách nhiệm vào một đối tượng một cách tự động. |
| Facade | Một lớp duy nhất đại diện toàn bộ cho một hệ thống con. |
| Flyweight | Một thể hiện nhỏ vừa đủ để sử dụng trong việc chia sẻ. |
| Proxy | Cung cấp một đại diện cho một đối tượng khác để hỗ trợ hoặc kiểm soát quá trình truy xuất đối tượng đó. |

* + 1. **Mẫu hành vi**

Các mẫu hình vi có liên quan đến các giải thuật và việc phân công trách nhiệm giữa các đối tượng. Các mẫu hành vi mô tả không chỉ mô hình của các đối tượng hay các lớp mà còn mô hình sự tương tác giữa chúng. Chúng thể hiện các luồng điều khiển phức tạp mà chúng ta khó theo dõi trong quá trình chạy thành trở nên đơn giản hơn. Khi đó, chúng chuyển sự chú ý của chúng ta ra khỏi các dòng điều khiển rườm rà phức tạp, và ta chỉ cần tập trung vào cách mà các đối tượng liên kết với nhau.

Có ba cách mà các mẫu hành vi quản lý các đối tượng của mình và cách các đối tượng đó tương tác với nhau.

Cách thứ nhất, chúng sử dụng các tính kế thừa để phân phối các hành vi giữa các lớp. Có hai mẫu sử dụng cách này là Template Method và Interpreter.

Cách thứ hai, chúng sử dụng sự kết hợp các đối tượng thay vì kế thừa. Một số mẫu mô tả làm cách nào một nhóm các đối tượng hợp tác với nhau để thực hiện một thao tác mà không đối tượng nào có thể tự mình thực hiện được, ví dụ như Mediator quản lý cách thức các đối tượng giao tiếp với nhau, Chain of Responsibility làm các đối tượng trong tập hợp liên kết lỏng lẻo với nhau, Observer định nghĩa và quản lý sự phụ thuộc giữa các đối tượng .

Cách thứ ba, các mẫu hành vi đóng gói các hành vi trong một đối tượng và ủy nhiệm các yêu cầu đến đối tượng đó. Mẫu Strategy là một ví dụ cụ thể, nó đóng gói một giải thuật vào một đối tượng, khi đó chúng ta dễ dàng lựa chọn và thay đổi giải thuật của một đối tượng khác sử dụng mẫu này.

Các mẫu hành vi của GoF bao gồm 11 mẫu, được giới thiệu sơ lược sau đây:

|  |  |
| --- | --- |
| Mẫu | Mô tả Ngắn gọn |
| Chain of resp | 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 . |
| 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. |
| Interpreter | Hỗ trợ việc định nghĩa biểu diễn văn phạm và bộ thông dịch cho một ngôn ngữ. |
| 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. |
| Mediator | Định nghĩa một đối tượng để bao bọc việc giao tiếp giữa một số đối tượng với nhau. |
| Memento | Lưu giữ và phục hồi trạng thái bên trong của một đối tượng mà vẫn không vi phạm việc tính đóng gói của đối tượng đó. |
| 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. |
| state | Thay đổi hành vi của một đối tượng khi trạng thái bên trong của nó thay đổi. |
| 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 |
| 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. |
| 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. |

* 1. **Lợi ích khi sử dụng mẫu**

1. Mẫu thiết kế tạo điều kiện cho việc tái sử dụng

Một trong những vấn đề gây nhiều tranh cãi nhất trong phát triển phần mềm là làm thế nào để kết hợp và tái sử dụng các thành phần nhiều hơn. Nhiều người trong số các bên liên quan tham gia phát triển phần mềm nhận ra sự cần thiết và cơ hội để tái sử dụng mã, nhưng thử thách sẽ ngày càng lớn hơn để có thể tái sử dụng trên quy mô lớn. Các mẫu thiết kế cung cấp cho chúng ta một cách để tái sử dụng các giải pháp thiết kế qua nhiều ứng dụng khác nhau

1. Mẫu thiết kế làm cho việc thiết kế đơn giản hơn

The previous section illustrates how the application of design patterns can make the design phase easier. When a problem from the analysis phase is one for which a design pattern exists the design pattern offers an instant solution of high quality. Design patterns make design easier but their application still requires intellect. You must be able to recognize when the pattern applies and how to specialize the pattern for the particular context. Applying design patterns is much easier than solving design problems from first principles but it's still not algorithmic

* 1. **Cấu trúc trình bày mẫu**

Các mẫu thiết kế trong báo cáo sẽ được trình bày theo cấu trúc dưới đây:

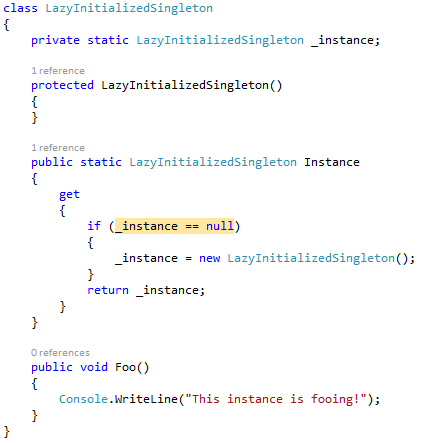
|  |  |
| --- | --- |
| Mục trình bày | Nội dung trình bày |
| Tên, phân loại, bí danh | Mô tả ngắn gọn về mẫu |
| Mục đích, ý định | Mô tả mẫu này làm được những gì |
| Động lực sử dụng | Nêu một ví dụ về một vấn đề cần giải quyết có thể sử dụng mẫu này. |
| Khả năng ứng dụng | Liệt kê một số tình huống cụ thể trong thiết kế phần mềm có thể áp dụng mẫu này |
| Cấu trúc mẫu | Mô tả mẫu bằng một sơ đồ UML bao gồm các lớp và đối tượng |
| Các thành viên | Trình bày ý nghĩa của các lớp/đối tượng tha gia vào mẫu thiết kế và trcahs nhiệm của chúng |
| Sự cộng tác | Mô tả các thức các thành viên của mẫu tương tác với nhau như thế nào để thực hiện trách nhiệm của chúng |
| Các hệ quả mang lại | Trình bày về ưu điểm khi sử dụng mẫu, các nhược điểm sẽ dẫn tới khi sử dụng mẫu. |
| Các chú ý khi cài đặt | Các chú ý đặc biệt khi cài đặt mẫu |
| Ví dụ thực tế | Nêu ra những ví dụ thực tế về các hệ thống (đã pt triển và đang chạy) có sử dụng mẫu này |
| Các mẫu liên quan | Những mẫu nào có liên hệ đến mẫu này, những điểm quan trọng cần phân biệt, mẫu này có thể phối hợp với những mẫu nào. |
| Mã nguồn minh họa | Trình bày một ví dụ demo cụ thể về mẫu |

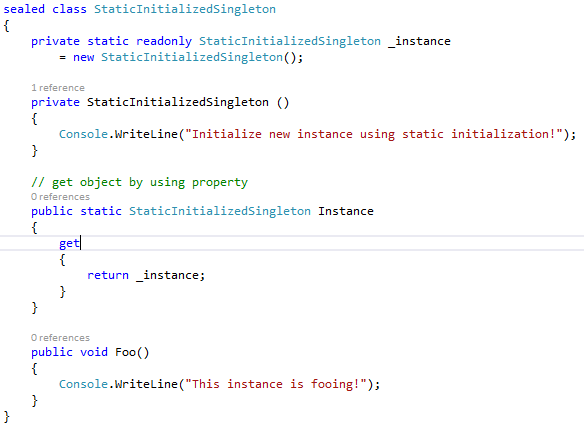
* 1. **Một số nguyên tắc**
* Bên sử dụng (client) không bao giờ gọi trực tiếp các lớp hiện thực, thay vào đó, chúng gọi các lớp ảo hoặc các interface (sự trừu tượng hóa của các lớp đó).
* Các thay đổi trong tương lai không ảnh hưởng đến hệ thống hiện tại

1. **Các mẫu Thiết kế GoF** 
   1. **Mẫu Singleton**
      1. **Tên, phân loại, bí danh**

* Tên: Singleton Pattern.
* Bí danh: Không có
* Phân loại: Mẫu kiến tạo
  + 1. **Mục đích, ý định**
* Đảm bảo rằng mỗi lớp chỉ có một thể hiện duy nhất và cung cấp khả năng truy xuất toàn cục đến thể hiện này.
  + 1. **Động lực sử dụng**
* Khi ta cần giới hạn việc tạo ra các thể hiện từ một lớp. Đặc biệt là các đối tượng tốn nhiều tài nguyên hệ thống khi được tạo ra.
  + 1. **Khả năng ứng dụng**
* Các hệ thống cần quản lý các đối tượng tài nguyên được chia sẻ chung, chẳng hạn như printer spooler.
  + 1. **Cấu trúc**



* + 1. **Các thành viên**
* Chỉ có một lớp Singleton tham gia vào mẫu. Lớp này phải định nghĩa hàm khởi tạo là private để ngăn việc khởi tạo thể hiện từ lớp này từ bên ngoài, đồng thời bắt buộc phải có hàm getInstace() để trả một tham chiếu đến đối tượng sẽ được trả về có kiểu của lớp đó.
  + 1. **Sự công tác**
* Lớp nào muốn thể hiện mẫu Singleton cần định nghĩa một biến thành viên có kiểu là chính nó, đông thời biến này cần được đặt modifier là static private.
* Hàm khởi tạo của lớp Singleton được định nghĩa thành protected hoặc private để người dùng không thể tạo thể hiện của lớp trực tiếp từ bên ngoài.
* Phương thức getInstance() dùng để khởi tạo đối tượng duy nhất, có modifier là public static. Client chỉ dùng phương thức getInstance() này để nhận được một đối tượng lớp Singleton.
  + 1. **Các hệ quả mang lại**
* Giới hạn số lượng thể hiện của lớp ở một con số nào đó, con số này phải có ý nghĩa nhất định trong chương trình của chúng ta.
* Nếu một lớp là Singleton, chúng ta có thể gặp khó khăn khi tạo lớp con của lớp đó.
  + 1. **Các chú ý liên quan đến cài đặt**
* Có nhiều cách để khởi tạo đối tượng sẽ được trả về trong C#:
* Sử dụng Lazy initialization và property method:
* Là cách thường dùng nhất.
* Cách tiếp cận này sẽ làm trì hoãn quá trình khởi tạo đối tượng cho đến khi nào client thực sự cần đến nó.
* Code minh họa:
* 
* Sử dụng Static Initialization – Khởi tạo tĩnh:
* Cách tiếp cận ở trên không phù hợp khi đối tượng được tạo ra cần được sử dụng chung, khi đó if (\_instance == null) sẽ gây ra lỗi khi lập trình đa luồng.
* Trong cách tiếp cận này, thể hiện \_instance sẽ được tạo ra bất cứ khi nào một thành viên của lớp StaticInitializedSingleton được tham chiếu đến. Khi đó chúng ta để cho CLR khởi tạo đối tượng. Lớp tương ứng sẽ được đánh dấu là sealed để ngăn việc kế thừa, đồng thời biến \_instance sẽ được đánh dấu là readonly, chỉ được gán giá trị lúc khởi tạo.
* Code minh họa:



* Trong nhiều trường hợp, cách tiếp cận này phù hợp hơn cách đầu tiên.
  + 1. **Các ví dụ về hệ thống thực tế**
    2. **Các mẫu liên quan**
    3. **Mã nguồn minh họa**
  1. **Mẫu Façade** 
     1. **Tên, phân loại, bí danh**
     2. **Mục đích, ý định**
     3. **Động lực sử dụng**
     4. **Khả năng ứng dụng**
     5. **Cấu trúc**
     6. **Các thành viên**
     7. **Sự công tác**
     8. **Các hệ quả mang lại**
     9. **Các chú ý liên quan đến cài đặt**
     10. **Các ví dụ về hệ thống thực tế**
     11. **Các mẫu liên quan**
     12. **Mã nguồn minh họa**
  2. **Mẫu Proxy**
     1. **Tên, phân loại, bí danh**
     2. **Mục đích, ý định**
     3. **Động lực sử dụng**
     4. **Khả năng ứng dụng**
     5. **Cấu trúc**
     6. **Các thành viên**
     7. **Sự công tác**
     8. **Các hệ quả mang lại**
     9. **Các chú ý liên quan đến cài đặt**
     10. **Các ví dụ về hệ thống thực tế**
     11. **Các mẫu liên quan**
     12. **Mã nguồn minh họa**
  3. **Mẫu Template Method**
     1. **Tên, phân loại, bí danh**
     2. **Mục đích, ý định**
     3. **Động lực sử dụng**
     4. **Khả năng ứng dụng**
     5. **Cấu trúc**
     6. **Các thành viên**
     7. **Sự công tác**
     8. **Các hệ quả mang lại**
     9. **Các chú ý liên quan đến cài đặt**
     10. **Các ví dụ về hệ thống thực tế**
     11. **Các mẫu liên quan**
     12. **Mã nguồn minh họa**
  4. **Mẫu Observer** 
     1. **Tên, phân loại, bí danh**
     2. **Mục đích, ý định**
     3. **Động lực sử dụng**
     4. **Khả năng ứng dụng**
     5. **Cấu trúc**
     6. **Các thành viên**
     7. **Sự công tác**
     8. **Các hệ quả mang lại**
     9. **Các chú ý liên quan đến cài đặt**
     10. **Các ví dụ về hệ thống thực tế**
     11. **Các mẫu liên quan**
     12. **Mã nguồn minh họa**

1. **Tổng kết**

1. **Phụ lục**
   1. **Phân loại**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | Mục đích | | |
| Khởi tạo | Cấu trúc | Hành vi |
| Phạm vi | Lớp | Factory Method, | Adapter(class), | Interpreter,  Template Method |
| Đối tượng | Abstract Factory,  Builder,  Prototype,  Singleton | Adapter(object),  Bridge,  Composite,  Decorator,  Façade, Proxy Flyweight, | Chain Of Respository ,  Command, Iterator,  Mediator, Memento,  Observer, State,  Strategy, Visitor |
| Tổng cộng | | 5 | 7 | 11 |

* 1. **Tần suất sử dụng các mẫu**

|  |  |  |
| --- | --- | --- |
| Tần suất sử dụng | Mẫu GoF | Số lượng |
| 5 - Cao | Singleton, Composite, Façade, Proxy, Iterator, Observer, Template Method | 7 |
| 4 – Tương đối cao | Abstract Factory, Factory Method, Adapter, Decorator, Command, State, Strategy | 7 |
| 3 – Trung bình | Builder, Bridge, Chain Of Responsitory | 3 |
| 2 – Tương đối thấp | Prototype, Flyweight, Mediator, Visitor | 4 |
| 1 – Thấp | Interpreter, Memento | 2 |

* 1. **Độ khó**

|  |  |  |
| --- | --- | --- |
| Cấp độ | Mẫu GoF | Tổng |
| A – Dễ | Façade, Singleton, Mediator, Iterator, Strategy, Command, Builder, State, Template Method, Factory Method, Memento, Prototype | 12 |
| B – Trung bình | Proxy, Decorator, Adapter, Bridge, Observer | 5 |
| C – Khó | Composite, Interpreter, Chain Of Respository, Abstract Factory, Flyweight, Visitor | 6 |

1. **Tài liệu tham khảo**

* Sách :
  + Thinking in java 4th Edition – tác giả - năm xuất bản – nhà xuất bản – chương sử dụng.
  + Design Elements Of Reusable Object Oriented Software Patterns – tác giả GoF – November 1994 – nhà xuất bản Addison Wesley.
* Tài liệu internet:
  + <http://sce2.umkc.edu/BIT/burrise/pl/design-patterns/>