

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**



**MẪU THIẾT KẾ**

**BÁO CÁO**

**SE401.L11**

**Nhóm 12**

|  |  |
| --- | --- |
| **NGUYỄN DU LỊCH** | **17520096** |
| **NGUYỄN VĂN ĐÔNG** | **17520350** |
| **NGUYỄN DUY MINH** | **17520754** |

***Sinh viên thực hiện:***

|  |  |
| --- | --- |
| **TRẦN ANH DŨNG** |  |

***Giảng viên hướng dẫn:***

**LỜI CẢM ƠN**

Nhóm 12 xin chân thành gửi lời cảm ơn đến thầy Trần Anh Dũng đã gắn bó cùng nhóm 12 trong suốt thời gian học tập môn học Mẫu thiết kế. Trong quá trình học tập, Thầy đã cung cấp cho nhóm những kiến thức chuyên môn cũng như hướng dẫn nhóm thực hiện các bài Seminar để đạt được kết quả như hôm nay.

Quá trình tìm hiểu các mẫu thiết kế đã gặp không ít khó khăn, với sự giúp đỡ và hỗ trợ từ Thầy đã giúp nhóm vượt qua và hạn chế những sai sót. Tuy vậy, dù có cố gắng nhiều song công việc của Nhóm 12 vẫn có nhiều thiếu sót, kính mong Thầy xem xét và góp ý để bài báo cáo và các bài Seminar phát triển và hoàn thiện hơn.

Xin chân thành cảm ơn.

Nhóm 12 – SE401.L11

**Nguyễn Du Lịch (17520096)**

**Nguyễn Văn Đông (17520350)**

**Nguyễn Duy Minh (17520754)**

**Mục lục**

[***I.*** ***Giới thiệu về Mẫu thiết kế (Design Pattern):*** 9](#_Toc60580343)

[***1.*** ***Mẫu thiết kế (Design Pattern) là gì?*** 9](#_Toc60580344)

[***2.*** ***Vai trò của Design pattern trong các dự án thực tế:*** 9](#_Toc60580345)

[***II.*** ***Giới thiệu sách Gang of Four 4.0 (GoF):*** 10](#_Toc60580346)

[***1.*** ***Tổng quan:*** 10](#_Toc60580347)

[***2.*** ***Phân loại Design Pattern theo GoF:*** 10](#_Toc60580348)

[***3.*** ***Danh sách các mẫu thiết kế GoF:*** 10](#_Toc60580349)

[***III.*** ***Các mẫu thiết kế GoF:*** 10](#_Toc60580350)

[***1.*** ***Abstract Factory:*** 10](#_Toc60580351)

[***1.1 Tổng quan:*** 10](#_Toc60580352)

[***1.2 Motivation:*** 10](#_Toc60580353)

[***1.3 Khả năng ứng dụng:*** 10](#_Toc60580354)

[***1.4 Cấu trúc:*** 10](#_Toc60580355)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 10](#_Toc60580356)

[***1.6 Mã nguồn minh hoạ:*** 11](#_Toc60580357)

[***1.7 Các hệ quả mang lại:*** 11](#_Toc60580358)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 11](#_Toc60580359)

[***1.9 Các mẫu liên quan:*** 11](#_Toc60580360)

[***2.*** ***Builder:*** 11](#_Toc60580361)

[***1.1 Tổng quan:*** 11](#_Toc60580362)

[***1.2 Motivation:*** 11](#_Toc60580363)

[***1.3 Khả năng ứng dụng:*** 11](#_Toc60580364)

[***1.4 Cấu trúc:*** 11](#_Toc60580365)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 11](#_Toc60580366)

[***1.6 Mã nguồn minh hoạ:*** 11](#_Toc60580367)

[***1.7 Các hệ quả mang lại:*** 12](#_Toc60580368)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 12](#_Toc60580369)

[***1.9 Các mẫu liên quan:*** 12](#_Toc60580370)

[***3.*** ***Factory Method:*** 12](#_Toc60580371)

[***1.1 Tổng quan:*** 12](#_Toc60580372)

[***1.2 Motivation:*** 12](#_Toc60580373)

[***1.3 Khả năng ứng dụng:*** 12](#_Toc60580374)

[***1.4 Cấu trúc:*** 12](#_Toc60580375)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 12](#_Toc60580376)

[***1.6 Mã nguồn minh hoạ:*** 12](#_Toc60580377)

[***1.7 Các hệ quả mang lại:*** 13](#_Toc60580378)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 13](#_Toc60580379)

[***1.9 Các mẫu liên quan:*** 13](#_Toc60580380)

[***4.*** ***Prototype:*** 13](#_Toc60580381)

[***1.1 Tổng quan:*** 13](#_Toc60580382)

[***1.2 Motivation:*** 13](#_Toc60580383)

[***1.3 Khả năng ứng dụng:*** 13](#_Toc60580384)

[***1.4 Cấu trúc:*** 13](#_Toc60580385)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 13](#_Toc60580386)

[***1.6 Mã nguồn minh hoạ:*** 13](#_Toc60580387)

[***1.7 Các hệ quả mang lại:*** 14](#_Toc60580388)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 14](#_Toc60580389)

[***1.9 Các mẫu liên quan:*** 14](#_Toc60580390)

[***5.*** ***Singleton:*** 14](#_Toc60580391)

[***1.1 Tổng quan:*** 14](#_Toc60580392)

[***1.2 Motivation:*** 14](#_Toc60580393)

[***1.3 Khả năng ứng dụng:*** 14](#_Toc60580394)

[***1.4 Cấu trúc:*** 14](#_Toc60580395)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 14](#_Toc60580396)

[***1.6 Mã nguồn minh hoạ:*** 14](#_Toc60580397)

[***1.7 Các hệ quả mang lại:*** 15](#_Toc60580398)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 15](#_Toc60580399)

[***1.9 Các mẫu liên quan:*** 15](#_Toc60580400)

[***6.*** ***Adapter:*** 15](#_Toc60580401)

[***1.1 Tổng quan:*** 15](#_Toc60580402)

[***1.2 Motivation:*** 15](#_Toc60580403)

[***1.3 Khả năng ứng dụng:*** 15](#_Toc60580404)

[***1.4 Cấu trúc:*** 15](#_Toc60580405)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 15](#_Toc60580406)

[***1.6 Mã nguồn minh hoạ:*** 15](#_Toc60580407)

[***1.7 Các hệ quả mang lại:*** 16](#_Toc60580408)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 16](#_Toc60580409)

[***1.9 Các mẫu liên quan:*** 16](#_Toc60580410)

[***7.*** ***Bridge:*** 16](#_Toc60580411)

[***1.1 Tổng quan:*** 16](#_Toc60580412)

[***1.2 Motivation:*** 16](#_Toc60580413)

[***1.3 Khả năng ứng dụng:*** 16](#_Toc60580414)

[***1.4 Cấu trúc:*** 16](#_Toc60580415)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 16](#_Toc60580416)

[***1.6 Mã nguồn minh hoạ:*** 16](#_Toc60580417)

[***1.7 Các hệ quả mang lại:*** 17](#_Toc60580418)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 17](#_Toc60580419)

[***1.9 Các mẫu liên quan:*** 17](#_Toc60580420)

[***8.*** ***Composite:*** 17](#_Toc60580421)

[***1.1 Tổng quan:*** 17](#_Toc60580422)

[***1.2 Motivation:*** 17](#_Toc60580423)

[***1.3 Khả năng ứng dụng:*** 17](#_Toc60580424)

[***1.4 Cấu trúc:*** 17](#_Toc60580425)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 17](#_Toc60580426)

[***1.6 Mã nguồn minh hoạ:*** 17](#_Toc60580427)

[***1.7 Các hệ quả mang lại:*** 18](#_Toc60580428)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 18](#_Toc60580429)

[***1.9 Các mẫu liên quan:*** 18](#_Toc60580430)

[***9.*** ***Decorator:*** 18](#_Toc60580431)

[***1.1 Tổng quan:*** 18](#_Toc60580432)

[***1.2 Motivation:*** 18](#_Toc60580433)

[***1.3 Khả năng ứng dụng:*** 18](#_Toc60580434)

[***1.4 Cấu trúc:*** 18](#_Toc60580435)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 18](#_Toc60580436)

[***1.6 Mã nguồn minh hoạ:*** 18](#_Toc60580437)

[***1.7 Các hệ quả mang lại:*** 19](#_Toc60580438)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 19](#_Toc60580439)

[***1.9 Các mẫu liên quan:*** 19](#_Toc60580440)

[***10.*** ***Facade:*** 19](#_Toc60580441)

[***1.1 Tổng quan:*** 19](#_Toc60580442)

[***1.2 Motivation:*** 19](#_Toc60580443)

[***1.3 Khả năng ứng dụng:*** 19](#_Toc60580444)

[***1.4 Cấu trúc:*** 19](#_Toc60580445)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 19](#_Toc60580446)

[***1.6 Mã nguồn minh hoạ:*** 19](#_Toc60580447)

[***1.7 Các hệ quả mang lại:*** 20](#_Toc60580448)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 20](#_Toc60580449)

[***1.9 Các mẫu liên quan:*** 20](#_Toc60580450)

[***11.*** ***Flyweight:*** 20](#_Toc60580451)

[***1.1 Tổng quan:*** 20](#_Toc60580452)

[***1.2 Motivation:*** 20](#_Toc60580453)

[***1.3 Khả năng ứng dụng:*** 20](#_Toc60580454)

[***1.4 Cấu trúc:*** 20](#_Toc60580455)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 20](#_Toc60580456)

[***1.6 Mã nguồn minh hoạ:*** 21](#_Toc60580457)

[***1.7 Các hệ quả mang lại:*** 21](#_Toc60580458)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 21](#_Toc60580459)

[***1.9 Các mẫu liên quan:*** 21](#_Toc60580460)

[***12.*** ***Proxy:*** 21](#_Toc60580461)

[***1.1 Tổng quan:*** 21](#_Toc60580462)

[***1.2 Motivation:*** 21](#_Toc60580463)

[***1.3 Khả năng ứng dụng:*** 21](#_Toc60580464)

[***1.4 Cấu trúc:*** 21](#_Toc60580465)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 21](#_Toc60580466)

[***1.6 Mã nguồn minh hoạ:*** 22](#_Toc60580467)

[***1.7 Các hệ quả mang lại:*** 22](#_Toc60580468)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 22](#_Toc60580469)

[***1.9 Các mẫu liên quan:*** 22](#_Toc60580470)

[***13.*** ***Chain of Responsibilty:*** 22](#_Toc60580471)

[***1.1 Tổng quan:*** 22](#_Toc60580472)

[***1.2 Motivation:*** 22](#_Toc60580473)

[***1.3 Khả năng ứng dụng:*** 22](#_Toc60580474)

[***1.4 Cấu trúc:*** 22](#_Toc60580475)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 22](#_Toc60580476)

[***1.6 Mã nguồn minh hoạ:*** 23](#_Toc60580477)

[***1.7 Các hệ quả mang lại:*** 23](#_Toc60580478)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 23](#_Toc60580479)

[***1.9 Các mẫu liên quan:*** 23](#_Toc60580480)

[***14.*** ***Command:*** 23](#_Toc60580481)

[***1.1 Tổng quan:*** 23](#_Toc60580482)

[***1.2 Motivation:*** 23](#_Toc60580483)

[***1.3 Khả năng ứng dụng:*** 23](#_Toc60580484)

[***1.4 Cấu trúc:*** 23](#_Toc60580485)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 23](#_Toc60580486)

[***1.6 Mã nguồn minh hoạ:*** 24](#_Toc60580487)

[***1.7 Các hệ quả mang lại:*** 24](#_Toc60580488)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 24](#_Toc60580489)

[***1.9 Các mẫu liên quan:*** 24](#_Toc60580490)

[***15.*** ***Interpreter:*** 24](#_Toc60580491)

[***1.1 Tổng quan:*** 24](#_Toc60580492)

[***1.2 Motivation:*** 24](#_Toc60580493)

[***1.3 Khả năng ứng dụng:*** 24](#_Toc60580494)

[***1.4 Cấu trúc:*** 24](#_Toc60580495)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 24](#_Toc60580496)

[***1.6 Mã nguồn minh hoạ:*** 25](#_Toc60580497)

[***1.7 Các hệ quả mang lại:*** 25](#_Toc60580498)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 25](#_Toc60580499)

[***1.9 Các mẫu liên quan:*** 25](#_Toc60580500)

[***16.*** ***Iterator:*** 25](#_Toc60580501)

[***1.1 Tổng quan:*** 25](#_Toc60580502)

[***1.2 Motivation:*** 25](#_Toc60580503)

[***1.3 Khả năng ứng dụng:*** 25](#_Toc60580504)

[***1.4 Cấu trúc:*** 25](#_Toc60580505)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 25](#_Toc60580506)

[***1.6 Mã nguồn minh hoạ:*** 26](#_Toc60580507)

[***1.7 Các hệ quả mang lại:*** 26](#_Toc60580508)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 26](#_Toc60580509)

[***1.9 Các mẫu liên quan:*** 26](#_Toc60580510)

[***17.*** ***Mediator:*** 26](#_Toc60580511)

[***1.1 Tổng quan:*** 26](#_Toc60580512)

[***1.2 Motivation:*** 26](#_Toc60580513)

[***1.3 Khả năng ứng dụng:*** 26](#_Toc60580514)

[***1.4 Cấu trúc:*** 26](#_Toc60580515)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 26](#_Toc60580516)

[***1.6 Mã nguồn minh hoạ:*** 27](#_Toc60580517)

[***1.7 Các hệ quả mang lại:*** 27](#_Toc60580518)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 27](#_Toc60580519)

[***1.9 Các mẫu liên quan:*** 27](#_Toc60580520)

[***18.*** ***Memento:*** 27](#_Toc60580521)

[***1.1 Tổng quan:*** 27](#_Toc60580522)

[***1.2 Motivation:*** 27](#_Toc60580523)

[***1.3 Khả năng ứng dụng:*** 27](#_Toc60580524)

[***1.4 Cấu trúc:*** 27](#_Toc60580525)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 27](#_Toc60580526)

[***1.6 Mã nguồn minh hoạ:*** 27](#_Toc60580527)

[***1.7 Các hệ quả mang lại:*** 28](#_Toc60580528)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 28](#_Toc60580529)

[***1.9 Các mẫu liên quan:*** 28](#_Toc60580530)

[***19.*** ***Observer:*** 28](#_Toc60580531)

[***1.1 Tổng quan:*** 28](#_Toc60580532)

[***1.2 Motivation:*** 28](#_Toc60580533)

[***1.3 Khả năng ứng dụng:*** 28](#_Toc60580534)

[***1.4 Cấu trúc:*** 28](#_Toc60580535)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 28](#_Toc60580536)

[***1.6 Mã nguồn minh hoạ:*** 28](#_Toc60580537)

[***1.7 Các hệ quả mang lại:*** 29](#_Toc60580538)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 29](#_Toc60580539)

[***1.9 Các mẫu liên quan:*** 29](#_Toc60580540)

[***20.*** ***State:*** 29](#_Toc60580541)

[***1.1 Tổng quan:*** 29](#_Toc60580542)

[***1.2 Motivation:*** 29](#_Toc60580543)

[***1.3 Khả năng ứng dụng:*** 29](#_Toc60580544)

[***1.4 Cấu trúc:*** 29](#_Toc60580545)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 29](#_Toc60580546)

[***1.6 Mã nguồn minh hoạ:*** 29](#_Toc60580547)

[***1.7 Các hệ quả mang lại:*** 29](#_Toc60580548)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 30](#_Toc60580549)

[***1.9 Các mẫu liên quan:*** 30](#_Toc60580550)

[***21.*** ***Strategy:*** 30](#_Toc60580551)

[***1.1 Tổng quan:*** 30](#_Toc60580552)

[***1.2 Motivation:*** 30](#_Toc60580553)

[***1.3 Khả năng ứng dụng:*** 30](#_Toc60580554)

[***1.4 Cấu trúc:*** 30](#_Toc60580555)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 30](#_Toc60580556)

[***1.6 Mã nguồn minh hoạ:*** 30](#_Toc60580557)

[***1.7 Các hệ quả mang lại:*** 30](#_Toc60580558)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 31](#_Toc60580559)

[***1.9 Các mẫu liên quan:*** 31](#_Toc60580560)

[***22.*** ***Template Method:*** 31](#_Toc60580561)

[***1.1 Tổng quan:*** 31](#_Toc60580562)

[***1.2 Motivation:*** 31](#_Toc60580563)

[***1.3 Khả năng ứng dụng:*** 31](#_Toc60580564)

[***1.4 Cấu trúc:*** 31](#_Toc60580565)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 31](#_Toc60580566)

[***1.6 Mã nguồn minh hoạ:*** 31](#_Toc60580567)

[***1.7 Các hệ quả mang lại:*** 31](#_Toc60580568)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 31](#_Toc60580569)

[***1.9 Các mẫu liên quan:*** 32](#_Toc60580570)

[***23.*** ***Visitor:*** 32](#_Toc60580571)

[***1.1 Tổng quan:*** 32](#_Toc60580572)

[***1.2 Motivation:*** 32](#_Toc60580573)

[***1.3 Khả năng ứng dụng:*** 32](#_Toc60580574)

[***1.4 Cấu trúc:*** 32](#_Toc60580575)

[***1.5 Vai trò và mối quan hệ các thành phần:*** 32](#_Toc60580576)

[***1.6 Mã nguồn minh hoạ:*** 32](#_Toc60580577)

[***1.7 Các hệ quả mang lại:*** 32](#_Toc60580578)

[***1.8 Ví dụ trong các hệ thống thực tế:*** 32](#_Toc60580579)

[***1.9 Các mẫu liên quan:*** 33](#_Toc60580580)

[***IV.*** ***Tài liệu tham khảo:*** 33](#_Toc60580581)

**NỘI DUNG CHÍNH**

1. ***Giới thiệu về Mẫu thiết kế (Design Pattern):***
2. ***Mẫu thiết kế (Design Pattern) là gì?***

Font này.

1. ***Vai trò của Design pattern trong các dự án thực tế:***

1. ***Giới thiệu sách Gang of Four 4.0 (GoF):***
2. ***Tổng quan:***

Ddddddddddddddddddddaaâsssssssssssssssssssssssssssssssssssssssssssssssssssssss

1. ***Phân loại Design Pattern theo GoF:***

Ddddddddddddddddddddaaâsssssssssssssssssssssssssssssssssssssssssssssssssssssss

1. ***Danh sách các mẫu thiết kế GoF:***

lựa chọn gồm có phần back end là ASP.NET Core và front end là ReactJS (dùng TypeScript)

1. ***Các mẫu thiết kế GoF:***
2. ***Abstract Factory:***

***1.1 Tổng quan:***

- Tên: Abstract Factory.

- Phân loại: Abstract Factory thuộc nhóm Creational Pattern.

- Bí danh: Fabrique Abstraite.

- Mục đích: là 1 super factory, cung cấp cho các cách thức để khởi tạo các factory con, từ đó có thể khởi tạo các đối tượng liên quan đến factory con đó.

***1.2 Khả năng ứng dụng:***

- Ứng dụng trong 1 hệ thống có nhiều module và có ràng buộc về mặt cấu hình giữa các module này.

- Inversion of Control (IOC)

- Khi cần khởi tạo các đối tượng có hình thức khởi tạo phức tạp và không ổn định.

***1.3 Cấu trúc:***

Cấu tạo của Abstract Factory gồm 3 thành phần chính:

* **AbstractFactory**: lớp này có trách nhiệm khởi tạo các factory con trong hệ thống.
* **Factory con**: khởi tạo 1 loại đối tượng nhất định trong hệ thống. Ví dụ:

+ FactoryX khởi tạo đối tượng của ProductA

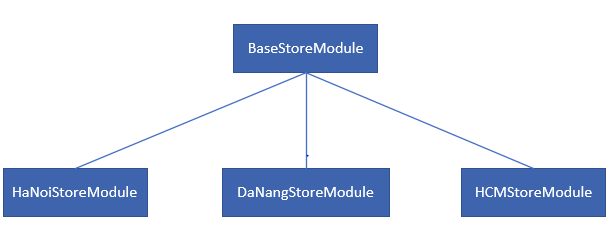
+ FactoryY khởi tạo đối tượng của ProductB

* **FactoryManager**: quản lí cấu hình của hệ thống và khởi tạo **AbstractFactory** dựa trên cấu hình đó.

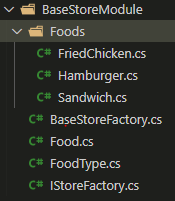
***1.4 Mã nguồn minh hoạ:***

**Vấn đề**: Bạn cần phát triển hệ thống sản xuất chuỗi cửa hàng ăn nhanh tên là FastFood. Các món bán trong cửa hàng bao gồm: **Hamburger**, **Sandwich**, **Fried chicken**. Tuy nhiên do sự khác nhau về vị trí địa lý và tính phổ biến mà nguyên liệu và giá cả của cùng 1 món sẽ khác nhau ở nhiều cửa hàng. Có tổng cộng 3 cửa hàng: **Đà Nẵng, Hà Nội, Hồ Chí Minh**

Source code sẽ có 3 module chính như trong hình.

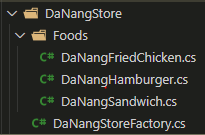


**BaseStoreModule**

**Foods**: các abstract class của các món ăn mà chuỗi cửa hàng sẽ phục vụ.

**BaseStoreFactory**: abstract factory quy định các đặc điểm của cửa hàng (như tên, danh sách các món ăn, …)

**DaNangStoreModule**

**Foods**: các lớp kế thừa từ các class món ăn tương ứng từ BaseStoreModule.

**DaNangStoreFactory**: factory kế thừa từ BaseStoreFactory

Tương tự với **HaNoiStoreModule** và **HCMStoreModule**

BaseStoreFactory có 1 hàm duy nhất là **ServeFood**. Hàm này trả về đối tượng ứng với món ăn mà cửa hàng cung cấp.



***StoreManager:*** *quản lí các cửa hàng của hệ thống*



***Class Program****: entrypoint của chương trình.*

***1.5 Các hệ quả mang lại:***

- Tuân thủ nguyên tắc Đóng gói (Encapsulation): Hệ thống giấu đi việc khởi tạo các class con. Ngược lại, việc khởi tạo là công việc của các Factory.

- Tuân thủ nguyên tắc Dependency Inversion: Thay vì người dùng trực tiếp khởi tạo các Factory thì lúc này việc khởi tạo được dành cho AbstractFactory.

- Đảm bảo sự đồng nhất giữa các class có cùng hệ.

- Khả năng mở rộng cao.

***1.6 Ví dụ trong các hệ thống thực tế:***

**Inversion of Control (IoC)**: được ứng dụng để khởi tạo các đối tượng phức tạp trong 1 hệ thống

**Color theme**

Việc cài đặt các theme khác nhau cho hệ thống được cài đặt dựa trên Abstract factory. Mỗi các nhóm màu được quy định trong 1 factory cụ thể. Lớp abstract factory sẽ trả về factory con tương ứng và lấy ra được nhóm màu của theme.

***1.7 Các mẫu liên quan:***

**Factory Method**

Mẫu Abstract Factory được thiết kế dựa trên Factory Method. Hay cụ thể hơn, Abstract Factory là 1 super factory khởi tạo các factory con.

**Singleton**

Thường được sử dụng với Singleton để khởi tạo đối tượng duy nhất của các factory con, từ đó giúp giảm tài nguyên của hệ thống.

1. ***Builder:***

***1.1 Tổng quan:***

- Tên: Builder.

- Phân loại: Abstract Factory thuộc nhóm Creational pattern.

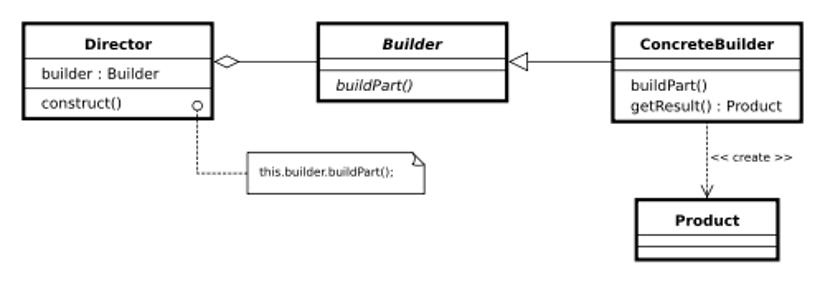
- Bí danh: Fabrique Abstraite.

- Mục đích, ý định: Cung cấp một đối tượng để xây dựng một đối tượng phức tạm một cách đơn giản bằng cách sử dụng tiếp cận từng bước. Việc xây dựng này là độc lập.

***1.3 Khả năng ứng dụng:***

- Builder Pattern được xây dựng để khắc phục một số nhược điểm của Factory Pattern và Abstract Factory Pattern khi mà Object có quá nhiều thuộc tính. Và khi Object có quá nhiều thuộc tính sẽ nảy sinh nhiều vấn đề như constructor, get set, tham số tùy chọn, …

***1.4 Cấu trúc:***

-Một builder gồm các thành phần cơ bản sau:

Product : đại diện cho đối tượng cần tạo, đối tượng này phức tạp, có nhiều thuộc tính.

Builder : là abstract class hoặc interface khai báo phương thức tạo đối tượng.

ConcreteBuilder : kế thừa Builder và cài đặt chi tiết cách tạo ra đối tượng. Nó sẽ xác định và nắm giữ các thể hiện mà nó tạo ra, đồng thời nó cũng cung cấp phương thức để trả các các thể hiện mà nó đã tạo ra trước đó.

Director/ Client: là nơi sẽ gọi tới Builder để tạo ra đối tượng.

Trường hợp đơn giản, chúng ta có thể gộp Builder và ConcreteBuilder thành class nội của Product.

***1.4 Mã nguồn minh hoạ:***

**Vấn đề**: ở công an ở nước X muốn một database chứa thông tin người dân của mình gồm nhiều trường sao cho một/một số trường có thể rỗng và phải có bắt lỗi các trường (format/true data)​

<https://github.com/LichND/Design-Pattern/tree/main/Builder>

***1.5 Các hệ quả mang lại:***

Dễ dàng thay đổi sửa chữa nhưng vẫn đảm bảo sự ổn định của interface gốc

Các class có thể thay thế dễ dàng nhưng vẫn bị ràng buộc đầy đủ chức năng

Các interface của builder có thể dễ dàng kế thừa, mở rộng

Thành phần của builder do interface/abstract class quy định

***1.6 Ví dụ trong các hệ thống thực tế:***

- Tạo object message phức tạp của discord

- Đơn giản hóa tạo các control trong thiết kế UI

***1.9 Các mẫu liên quan:***

Factory Pattern: tương tự nhưng cung cấp nhiều quyền kiểm soát hơn đối với quá trình tạo đối tượng. Builder tạo đối tượng step by step trong khi Factory Pattern không rõ cách tạo

1. ***Factory Method:***

***1.1 Tổng quan:***

- Tên: Abstract Factory.

- Phân loại: Abstract Factory thuộc nhóm Creational Pattern.

- Bí danh: Fabrique Abstraite.

- Mức độ phổ biến:

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.2 Motivation:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.3 Khả năng ứng dụng:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.4 Cấu trúc:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.5 Vai trò và mối quan hệ các thành phần:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.6 Mã nguồn minh hoạ:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.7 Các hệ quả mang lại:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.8 Ví dụ trong các hệ thống thực tế:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.9 Các mẫu liên quan:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

1. ***Prototype:***

***1.1 Tổng quan:***

- Tên: Abstract Factory.

- Phân loại: Abstract Factory thuộc nhóm Creational Pattern.

- Bí danh: Fabrique Abstraite.

- Mức độ phổ biến:

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.2 Motivation:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.3 Khả năng ứng dụng:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.4 Cấu trúc:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.5 Vai trò và mối quan hệ các thành phần:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.6 Mã nguồn minh hoạ:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.7 Các hệ quả mang lại:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.8 Ví dụ trong các hệ thống thực tế:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.9 Các mẫu liên quan:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

1. ***Singleton:***

***1.1 Tổng quan:***

- Tên: Abstract Factory.

- Phân loại: Abstract Factory thuộc nhóm Creational Pattern.

- Bí danh: Fabrique Abstraite.

- Mức độ phổ biến:

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.2 Motivation:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.3 Khả năng ứng dụng:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.4 Cấu trúc:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.5 Vai trò và mối quan hệ các thành phần:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.6 Mã nguồn minh hoạ:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.7 Các hệ quả mang lại:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.8 Ví dụ trong các hệ thống thực tế:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.9 Các mẫu liên quan:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

1. ***Adapter:***

***1.1 Tổng quan:***

- Tên: Adapter.

- Phân loại: Structual pattern.

- Mức độ phổ biến: Cao

- Mục đích, ý định: Adapter hoạt động như 1 cầu nối giữa 2 interface không tương thích với nhau.

Có 2 loại mẫu adapter: **Object adapter** và **Class adapter**

***1.2. Khả năng ứng dụng:***

* Khi hệ thống cần sử dụng các class ở bên thứ 3 nhưng interface lại không tương thích.
* Khi cần mở rộng chức năng của một số class có sẵn trong hệ thống nhưng lại không thể đưa các chức năng đó vào class cha.

***1.4 Cấu trúc:***

****Object Adapter**

Adapter gồm 3 thành phần chính:

**Client:** bao gồm các thành phần của client dùng để giao tiếp trực tiếp với hệ thống. Cụ thể trong thiết kế gồm 1 interface **IClientInterface** và 1 class **ConcreteClient** kế thừa từ interface trên

**Service:** các thành phần của service cần sử dụng trong client. Cụ thể ở đây có 1 interface **I3rdService** và 2 class Service kế thừa từ interface trên.

**Adapter**: Kế thừa từ **IClientInterface** và có tham chiếu đến **I3rdService**. Bản chất adapter lúc này như 1 client nhưng được tích hợp thêm service vào trong class.

**Class Adapter**

**Class Adapter** cũng bao gồm 3 thành phần giống như Object Adapter. Tuy nhiên lúc này Adapter kế thừa từ class **3rdService**.

*Hạn chế:*

- Ngôn ngữ phải hỗ trợ đa kế thừa (multiple classes inheritance)

- **3rdService** phải là 1 class (tăng mức độ phụ thuộc của adapter)

***1.5 Mã nguồn minh hoạ:***

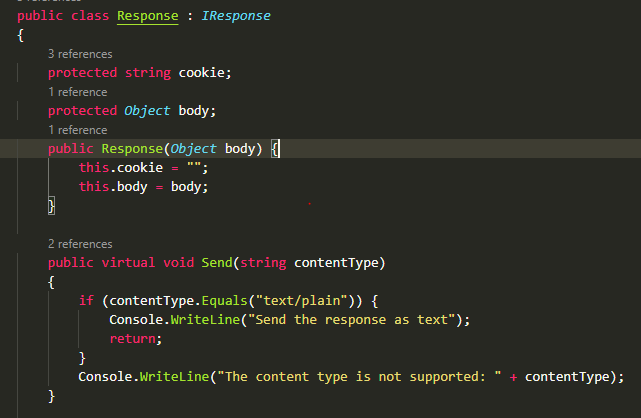
Server sử dụng **IResponse** để gửi response về cho client. Mặc định sendBody chỉ ở dạng text.

**ResponseSender** là 1 service thứ 3 cho phép gửi response ở format khác



*(các thành phần của chương trình)*

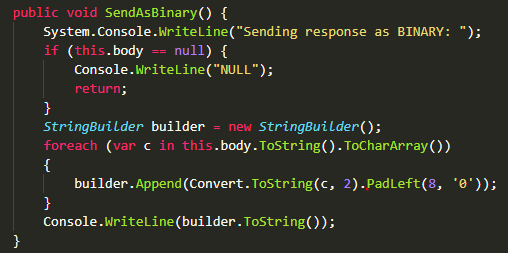
Sử dụng **ResponseFormatAdapter** để bổ sung them định dạng cho response



Lớp **ResponseSender** hỗ trợ 2 hàm:

**SendAsJSON**: hỗ trợ gửi body dưới dạng JSON

**SendAsBinary**: hỗ trợ gửi body dưới dạng Binary



Lớp **ResponseFormatAdapter**:

* Có tham chiếu đến lớp **ResponseSender** và kế thừa từ interface **IResponse**.
* Implement hàm **Send()** và bổ sung thêm format của body.



Entrypoint của demo:

***1.6 Các hệ quả mang lại:***

* Không cần mở rộng interface cho cùng 1 chức năng hoặc cho các chức năng mà hệ thống sẽ không dùng (**Interface Segragation**)
* Đảm bảo tính ổn định của các interface gốc của hệ thống (**Open/Close principle**).
* Đảm bảo các thành phần mở rộng không phụ thuộc vào các lớp hệ thống

***1.7 Ví dụ trong các hệ thống thực tế:***

**Socket adapter**

Socket là 1 mô hình giao tiếp realtime giữa client và server trên các channel với kết nối ổn định. Tuy nhiên, khá với bản chất của RESTful API, Socket lại không phải là stateless. Mặc định các client kết nối socket với nhau trên 1 process mới có thể giao tiếp với nhau.

Do đó, socket adapter là 1 mô hình thiết kế giúp mở rộng việc giao tiếp giữa các socket trên nhiều process khác nhau. Socket adapter lúc này giữ tham chiếu đến thể hiện của Server và các thông điệp gửi và nhận của client sẽ được gửi qua Server.

1. ***Bridge:***

***1.1 Tổng quan:***

- Tên: Abstract Factory.

- Phân loại: Abstract Factory thuộc nhóm Creational Pattern.

- Bí danh: Fabrique Abstraite.

- Mức độ phổ biến:

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.2 Motivation:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.3 Khả năng ứng dụng:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.4 Cấu trúc:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.5 Vai trò và mối quan hệ các thành phần:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.6 Mã nguồn minh hoạ:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.7 Các hệ quả mang lại:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.8 Ví dụ trong các hệ thống thực tế:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.9 Các mẫu liên quan:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

1. ***Composite:***

***1.1 Tổng quan:***

- Tên: Abstract Factory.

- Phân loại: Abstract Factory thuộc nhóm Creational Pattern.

- Bí danh: Fabrique Abstraite.

- Mức độ phổ biến:

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.2 Motivation:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.3 Khả năng ứng dụng:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.4 Cấu trúc:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.5 Vai trò và mối quan hệ các thành phần:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.6 Mã nguồn minh hoạ:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.7 Các hệ quả mang lại:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.8 Ví dụ trong các hệ thống thực tế:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.9 Các mẫu liên quan:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

1. ***Decorator:***

***1.1 Tổng quan:***

- Tên: Decorator.

- Phân loại: Decorator thuộc nhóm Structural Pattern.

- Mục đích, ý định: giúp mở rộng cài đặt của 1 lớp mà không thay đổi cấu trúc của nó. Lớp mở rộng được xem như 1 wrapper của lớp đã có sẵn.

***1.2 Khả năng ứng dụng:***

* Sử dụng khi cần bổ sung thêm các hành vi cho đối tượng tại thời điểm thực thi mà không cần phải thay đổi code.
* Sử dụng khi cảm thấy việc kế thừa là bất khả thi hoặc sai lệch bản chất của đối tượng.

***1.3 Cấu trúc:***



Gồm 2 thành phần:

**Decoratee**: các thành phần của hệ thống cần được mở rộng chức năng. Cụ thể bao gồm 1 interface **IDecoratee** và các class con tương ứng.

**Decorator**: các thành phần mở rộng chức năng cho **Decoratee**. Trong đó:

**- BaseDecorator**: là 1 lớp trừu tượng implement **IDecoratee** và có 1 tham chiếu đến **IDecoratee.**

- Các class con kế thừa từ BaseDecorator và cài đặt các chức năng mở rộng cho Decoratee.

***1.4 Mã nguồn minh hoạ:***

Bạn bước vào tiệm bánh canh, khi gọi món bạn có thể có nhiều yêu cầu khác nhau về gia vị và thành phần. Trong đó bánh canh là món chính, gia vị và các món ăn thêm như chả cá, hành ngò, da heo, tiêu, … Tùy vào khẩu vị mà bạn có thể yêu cầu khác nhau về món ăn như bánh canh không hành, hoặc không cho tiêu vào, … Từ đó có thể thấy chỉ một món ăn chính và vài thành phần phụ đã tạo ra rất nhiều biến thể của món ăn. Tuy nhiên, với việc áp dụng mẫu Decorator, chúng ta có thể quản lí chúng một cách dễ dàng.

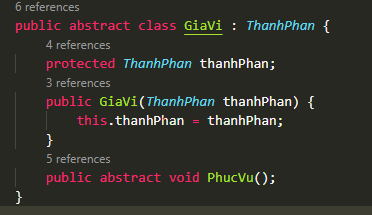
*(hình minh họa)*



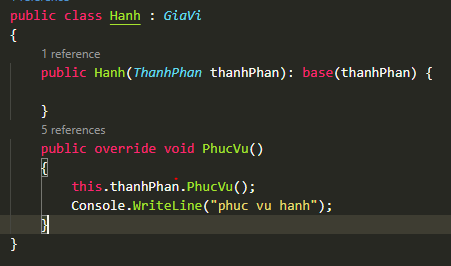
Thiết kế gồm 2 phần:

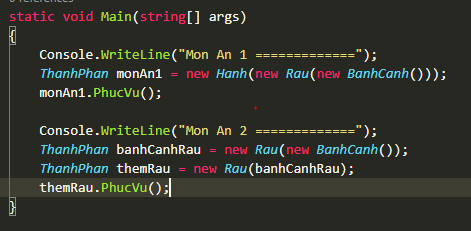
**MonAn**: class này tương ứng với Decoratee trong thiết kế.

**GiaVi:** bao gồm các class thuộc Decorator. Các lớp này có nhiệm vụ mở rộng lớp **MonAn** (cũng giống như gia vị bổ sung thêm hương vị cho món ăn).



Constructor của lớp **GiaVi** nhận tham số là interface **ThanhPhan**.

Các lớp kế thừa **GiaVi** sẽ override hàm **PhucVu**. Hàm này được mở rộng bằng cách gọi hàm PhucVu của thành phần trước, sau đó tiếp tục thực thi chức năng chính của nó.

**Chương trình chính:**

Các Decorator được khởi tạo bằng cách khởi tạo lồng nhau thông qua tham số của constructor của Decorator.

***1.5 Các hệ quả mang lại:***

- Đơn giản hóa hệ thống: Các lớp này đều có cùng 1 hành vi. Do đó việc quản lí dựa trên Decorator sẽ đơn giản hóa quá trình quản lí cũng như việc gọi thực thi chương trình.

- Khả năng mở rộng cao, linh hoạt.

*Nhược điểm:*

- Các thành phần bắt buộc phải khởi tạo từ đầu và không thể thay đổi trong quá trình hoạt động.

- Không thể áp dụng với các thành phần có nhiều chức năng hoặc quá phức tạp hoặc phụ thuộc lẫn nhau.

***1.6 Ví dụ trong các hệ thống thực tế:***

**Được sử dụng nhiều trong việc mở rộng chức năng của lớp**

Trong rất nhiều trường hợp, chúng ta cần thay đổi chức năng của 1 hàm nhưng lại không nắm rõ những yêu cầu có thể được đưa ra trong tương lai. Nếu chúng ta mở rộng bằng việc thêm hàm mới thì class sẽ phình to 1 cách không cần thiết (vì các hàm đều có chung 1 biểu hiện). Ngược lại, việc sử dụng Decorator cho phép chúng ta mở rộng hàm mà không cần can thiệp vào lớp cơ sở. Khi có sự thay đổi về yêu cầu thì chúng ta chỉ cần đơn giản thay lớp Decorator bằng lớp Decorator khác.

***1.7 Các mẫu liên quan:***

**Prototype**

Decorator có cấu trúc nhiều lớp phức tạp nên có thể kết hợp với Prototype để đơn giản hóa việc sao chép.

**Adapter**

Tuy cả adapter và decorator có cấu trúc giống nhau lại có mục đích sử dụng hoàn toàn khác nhau. Adapter giúp chúng ta tích hợp các lớp bên thứ 3 vào trong hệ thống, trong khi đó, Decorator giúp mở rộng phạm vi hoạt động của 1 hàm bên trong class cơ sở.

1. ***Facade:***

***1.1 Tổng quan:***

- Tên: Abstract Factory.

- Phân loại: Abstract Factory thuộc nhóm Creational Pattern.

- Bí danh: Fabrique Abstraite.

- Mức độ phổ biến:

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.2 Motivation:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.3 Khả năng ứng dụng:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.4 Cấu trúc:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.5 Vai trò và mối quan hệ các thành phần:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.6 Mã nguồn minh hoạ:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.7 Các hệ quả mang lại:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.8 Ví dụ trong các hệ thống thực tế:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.9 Các mẫu liên quan:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

1. ***Flyweight:***

***1.1 Tổng quan:***

- Tên: Abstract Factory.

- Phân loại: Abstract Factory thuộc nhóm Creational Pattern.

- Bí danh: Fabrique Abstraite.

- Mức độ phổ biến:

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.2 Motivation:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.3 Khả năng ứng dụng:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.4 Cấu trúc:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.5 Vai trò và mối quan hệ các thành phần:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.6 Mã nguồn minh hoạ:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.7 Các hệ quả mang lại:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.8 Ví dụ trong các hệ thống thực tế:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.9 Các mẫu liên quan:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

1. ***Proxy:***

***1.1 Tổng quan:***

- Tên: Abstract Factory.

- Phân loại: Abstract Factory thuộc nhóm Creational Pattern.

- Bí danh: Fabrique Abstraite.

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.2 Motivation:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.3 Khả năng ứng dụng:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.4 Cấu trúc:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.5 Vai trò và mối quan hệ các thành phần:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.6 Mã nguồn minh hoạ:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.7 Các hệ quả mang lại:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.8 Ví dụ trong các hệ thống thực tế:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.9 Các mẫu liên quan:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

1. ***Chain of Responsibilty:***

***1.1 Tổng quan:***

- Tên: Abstract Factory.

- Phân loại: Abstract Factory thuộc nhóm Creational Pattern.

- Bí danh: Fabrique Abstraite.

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.2 Motivation:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.3 Khả năng ứng dụng:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.4 Cấu trúc:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.5 Vai trò và mối quan hệ các thành phần:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.6 Mã nguồn minh hoạ:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.7 Các hệ quả mang lại:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.8 Ví dụ trong các hệ thống thực tế:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.9 Các mẫu liên quan:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

1. ***Command:***

***1.1 Tổng quan:***

* Tên: Command.
* Phân loại: Command thuộc nhóm Behavioral Pattern.
* Mục đích, ý định:
* Đưa việc xử lí logic và các thông tin cần thiết vào trong thành phần gọi là **Command**.
* Command sau đó sẽ được gọi để thực thi công việc. Thành phần gọi tới Command để xử lý được gọi là **Invoker**.

***1.2 Khả năng ứng dụng:***

* Cần 1 giải pháp quản lí các xử lí logic như các object riêng biệt (thêm, loại bỏ, thực thi, …).
* Khi cần linh động về thời gian thực thi (xử lí request theo queue, schedule, …).
* Khi cần implement các thao tác reverse, undo.

***1.4 Cấu trúc:***

Bao gồm 3 thành phần chính:

**Command**: là 1 wrapper đối với phần lệnh cần thực thi. Cụ thể ở đây là interface **Command** và 2 class **CommandA** và **CommandB.**

Có 2 loại command:

* Normal command: các command được thực thi độc lập mà không cần tham số nào từ bên ngoài
* Param command: các command phụ thuộc vào tham số để thực thi.

Mỗi command đều có hàm **execute**. Hàm này chịu trách nhiệm thực thi công việc được yêu cầu.

**Receiver:** đối tượng có chức năng cần được thực thi bên trong command. Cụ thể ở trong thiết kế là hàm **doSomeThing**. **Command** có giữ tham chiếu đến **Receiver** để có thể gọi hàm này.

**Invoker:** là 1 class quản lí và thực thi command. Invoker có tham chiếu đến command.

* 1. ***Các hệ quả mang lại:***

- Tuân thủ **Single Responsibility.** Bởi vì chỉ có trách nhiệm duy nhất là gọi hàm cần thực thi, do đó, việc bổ sung các command mới không ảnh hưởng nhiều đến các command trước đó.

- Tuân thủ **Open/Closed Principle.**

- Phân tầng giữa các thành phần riêng biệt trong hệ thống **( Separation of Concerns).** Lúc này Client sẽ gọi thực thi các hàm của hệ thống thông qua các command.

***1.6 Ví dụ trong các hệ thống thực tế:***

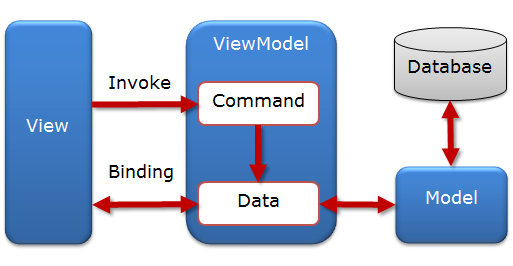
**Reverse Operations**

Gần như các hệ thống làm việc với dữ liệu đều có chức năng Reverse (hoặc Undo) để khôi phục lại trạng thái của hệ thống trước khi thực hiện 1 thao tác nào đó. Tuy nhiên để làm được điều này cần phải có 1 phương thức hiệu quả để quản lí các thao tác cũng như trạng thái của hệ thống.

Trong sơ đồ trên, các command được quản lí thông qua class **CommandHistory**. Class này có 2 hàm push và pop dùng để lưu lại thao tác mới và lấy ra thao tác mới nhất khi cần reverse.

Trạng thái của chương trình được lưu giữ bên trong **UndoCommand**. Do đó, khi lệnh undo được gọi, chương trình có thể trả về trạng thái trước đó.

**Mô hình MVC, MVVM:**

Các mô hình thiết kế hệ thống MVC và MVVM đều có sự phân lớp giữa các thành phần giao diện, logic và dữ liệu. Các thành phần giao diện không tương tác trực tiếp với phần xử lý business mà thông qua các command để thực thi.

***1.7 Các mẫu liên quan:***

**Observer:**

Mẫu observer dùng hiện thực hóa mô hình event-based system mà tại đó, các chức năng được gọi khi có sự kiện diễn ra.

Khi kết hợp với **Command**, các subcriber có thể giao tiếp với hệ thống thông qua các Command thay vì gọi trực tiếp phần xử lí bên trong của hệ thống.

**Prototype**

Có thể kết hợp với prototype để đơn giản hóa việc tạo các bản copy của command.

**Visitor**

Visitor và Command có chung mục đích sử dụng.

Visitor linh động hơn trong việc xử lý các object của các class khác nhau và có tham số khác nhau.

1. ***Interpreter:***

***1.1 Tổng quan:***

* Tên: Interpreter.
* Phân loại: Intepreter thuộc nhóm Behavioral Pattern.
* Mục đích, ý định: Dùng để định nghĩa các cú pháp (syntax) cho 1 ngôn ngữ và cách diễn giải các syntax đó. Cấu trúc này được gọi là **Abtract Syntax Tree (AST)**

***1.2 Khả năng ứng dụng:***

* Khi cần định nghĩa (hoặc biên dịch) một loại ngôn ngữ với cú pháp đơn giản (SQL Parsing, XML, các biểu thức toán, …).

***1.4 Cấu trúc:***

Mô hình gồm 2 thành phần:

**Expression**: định nghĩa các syntax và cách diển giải các syntax đó. Có 2 loại expression

- Terminal expression: biểu thức đơn có khả năng tự định nghĩa.

- Nonterminal expression: biểu thức có cấu trúc phức (tạo nên từ các biểu thức đơn hoặc phức khác).

**Context**: đầu vào của expression, chứa thông tin mà expression cần diễn giải,

***1.5 Mã nguồn minh hoạ:***

Thiết kế chương trình để chuyển từ chữ số La Mã sang chữ số thường.

**TerminalNumeral**: thể hiện các giá trị số đơn (I, V, X , L, C, D , M)

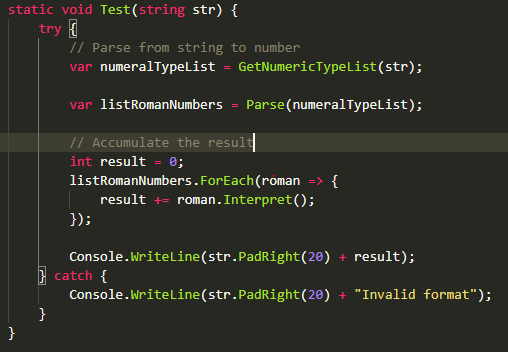
**CombinedNumeral**: thể hiện các giá trị số ghép từ 2 ký tự (IV, IX, XL, XC, CD)

**Terminal numeral**

**Combined numeral**

*(2 class quy định các diễn giải số I và V trong La Mã)*

*(class quy định cách diễn giải các số thuộc nhóm I)*

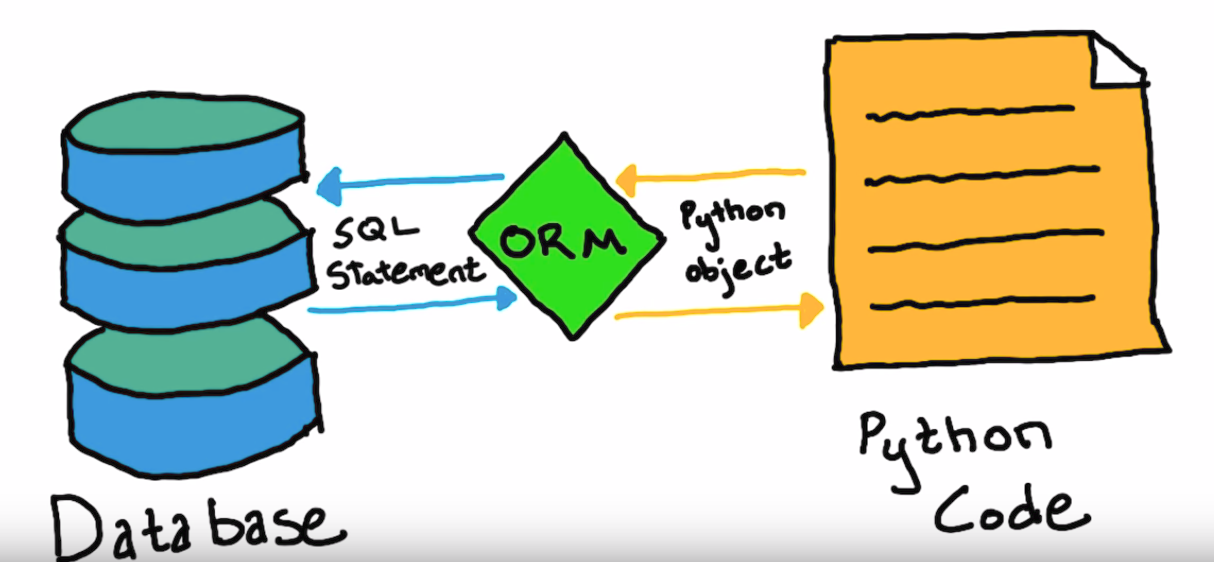
Chương trình chính:

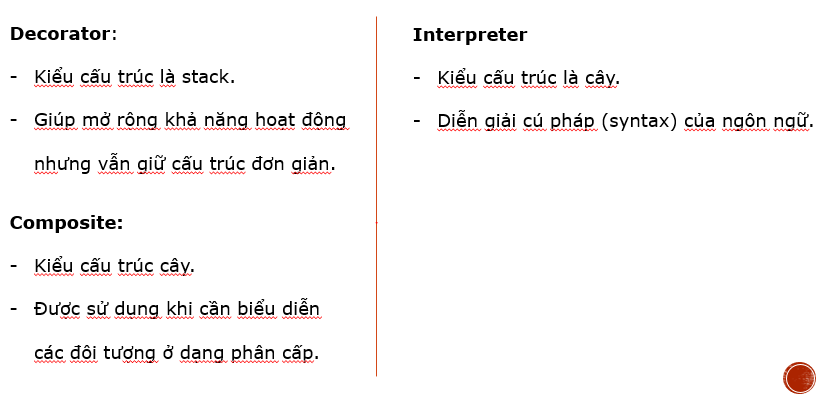
***1.6 Các hệ quả mang lại:***

- Tuân thủ nguyên tắc **Open/Closed principle.**

***1.7 Ví dụ trong các hệ thống thực tế:***

**ORM (object relational mapping)**

ORM là mô hình nhằm diễn giải các đối tượng, câu lệnh ở 1 hệ thống khác (như SQL, MySQL, Cloud, …) thành các hàm có thể thực thi được bên trong chương trình, qua đó giúp đơn giản hóa việc giao tiếp và đồng bộ giữa các hệ thống với nhau.

***1.8 Các mẫu liên quan:***

1. ***Iterator:***

***1.1 Tổng quan:***

- Tên: Abstract Factory.

* Phân loại: Abstract Factory thuộc nhóm Creational Pattern.

- Bí danh: Fabrique Abstraite.

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.2 Motivation:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.3 Khả năng ứng dụng:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.4 Cấu trúc:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.5 Vai trò và mối quan hệ các thành phần:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.6 Mã nguồn minh hoạ:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.7 Các hệ quả mang lại:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.8 Ví dụ trong các hệ thống thực tế:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.9 Các mẫu liên quan:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

1. ***Mediator:***

***1.1 Tổng quan:***

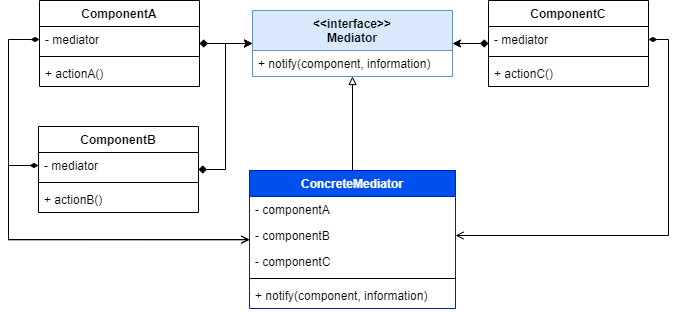
- Tên: Mediator.

- Phân loại: Mediator thuộc nhóm Behavioral Pattern.

* - Mục đích, ý định: giúp giảm thiểu sự phụ thuộc phức tạp giữa các object với nhau bằng cách giao tiếp thông qua 1 object trung gian (**mediator**).

***1.3 Khả năng ứng dụng:***

* Khi mức độ phụ thuộc của các component phức tạp nhưng vẫn muốn đảm bảo khả năng tái sử dụng của component.
* Khi các component chỉ phụ thuộc nhau trong 1 ngữ cảnh nhất định.

***1.4 Cấu trúc:***

Thiết kế chỉ có duy nhất 1 thành phần là **Mediator**, trong đó:

**Mediator**: là 1 Interface và hàm **notify**. Hàm này có tác dụng gửi thông điệp từ 1 component đến 1 component khác.

**ConcreteMediator**. Implement Mediator và giữ tham chiếu đến các component trong hệ thống.

Các component giữ tham chiếu đến interface **Mediator**. Khi muốn gửi đi thông điệp hoặc thông tin sự kiện, component sẽ gọi hàm **notify** của mediator.

***1.6 Mã nguồn minh hoạ:***

Xây dựng phòng chat cho phép người dùng gửi và nhận tin nhắn của nhau. Phòng chat sẽ có các sự kiện thông báo như: người dùng mới tham gia phòng, người dùng gửi tin nhắn, người dùng rời phòng.

Graphical user interface, application, Teams

Description automatically generated

Gồm 2 thành phần chính:

**Phòng chat**: bao gồm

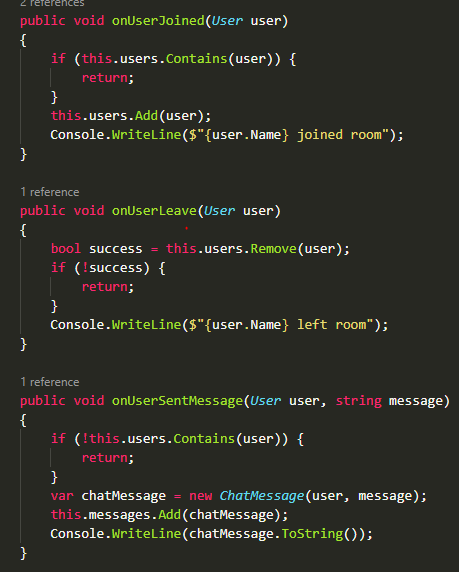
- IChatRoom**:** Interfacequy định các sự kiện của phòng chat.

- ConcreteChatRoom**:** implement IChatRoom. Lớp này giữ tham chiếu đến các users đang có mặt trong phòng chat.

**User**: Lớp mang thông tin của người dùng. Lớp này có tham chiếu đến IChatRoom để quy định phòng chat hiện tại của người dùng.



Các sự kiện của phòng chat được gọi bên trong các hàm của lớp **User**.



Các sự kiện của **ConcreteChatRoom** khi được gọi sẽ thực hiện các chức năng của phòng chat như thêm user, xóa user, in ra tin nhắn user đã gửi.

Chương trình chính:

***1.7 Các hệ quả mang lại:***

- Đảm bảo nguyên tắc Single Responsibility: Lúc này các component không cần phải thêm các chức năng giúp giao tiếp với các component khác. Từ đó tăng khả năng tái sử dụng component.

- Đảm bảo nguyên tắc Open/Closed.

- Đảm bảo nguyên tắc Dependency Inversion: Lúc này thay vì các component phụ thuộc lẫn nhau thì luồng phụ thuộc được đẩy sang 1 lớp có mức cao hơn là **Mediator**.

*Nhược điểm:*

- Khi Mediator trở nên quá phức tạp thì sẽ rất khó quản lí.

- Chỉ phát huy tối đa công dụng khi các sự kiện được thông báo ở mức global.

***1.8 Ví dụ trong các hệ thống thực tế:***

**Phòng chat**

**Form**

Khi xây dựng các form, các component trong form có thể phụ thuộc lẫn nhau (như khi ta chọn check box hiện mật khẩu thì textbox mật khẩu sẽ chuyển sang chế độ text hay vì mask). Lúc này, ta sử dụng Form như 1 Mediator, các component khác sẽ giao tiếp với nhau thông qua Form.

***1.9 Các mẫu liên quan:***

**Observer**

Cả 2 mẫu đều nhằm mục đích truyền đạt thông tin đến các thành phần của hệ thống.

**Observer**

- Các subscriber nhận thông điệp từ 1 nguồn duy nhất (ở đây là observer).

- Event-based.

**Mediator**

- Các component có thể nhận thông điệp từ nhiều nguồn khác nhau (các component khác).

- Object-based.

**Façade**

Tổ chức lại các thành phần để giảm bớt các sự phụ thuộc không cần thiết. Tuy nhiên, nếu như Façade cung cấp một interface đại diện cho thành phần hệ thống để giao tiếp với các hệ thống khác thì Mediator chuyển đổi sự phụ thuộc của các component về 1 lớp trung tâm.

1. ***Memento:***

***1.1 Tổng quan:***

- Tên: Memento.

- Phân loại: Memento thuộc nhóm Behavioral Pattern.

* - Mục đích, ý định: giúp lưu trữ và khôi phục trạng thái của 1 object mà không tiết lộ quá nhiều về chi tiết của object đó.

***1.3 Khả năng ứng dụng:***

- Khi cần quản lí (lưu trữ, truy xuất) toàn bộ thông tin state của 1 object.

- Có thể sử dụng trong transaction để rollback khi có lỗi.

***1.4 Cấu trúc:***

Mục tiêu thiết kế:

- Không để public state ở cả component lẫn Memento

- Chỉ có component mới tác động trực tiếp đến state của Memento

Graphical user interface, application

Description automatically generated**Giải pháp 1**: Sử dụng nested class

Gồm 3 thành phần:

**Component**: thành phần cần quản lí state. Có 2 hàm:

- **save**: lưu trữ lại state của component. Hàm này trả về Memento.

- **restore**: Khôi phục lại state của component. Tham số nhận vào là 1 memento.

**Memento**: Lưu trữ state của component. Đây là lớp nằm bên trong lớp **Component**. Do đó chỉ có Component có thể access trực tiếp các thành phần bên trong Memento.

**ComponentWrapper**: có tham chiếu đến component và danh sách các Memento của component cần quản lí. Có thể hiểu ComponentWrapper như 1 component nhưng được cài đặt thêm chức năng quản lí state.

*Nhược điểm:* Nested class chỉ có ở một số ngôn ngữ nhất định.

**Giải pháp 2:** Sử dụng interface trung gianGraphical user interface

Description automatically generated

Thiết kế này cũng gồm 3 thành phần như trong giải pháp 1. Tuy nhiên lúc này Memento bao gồm 1 interface rỗng và class **ConcreteMemento** quản lí state của component. ComponentWrapper quản lí trạng thái của component thông qua interface **Memento**.

Đây là giải pháp thay thế cho giải pháp 1 đối với những ngôn ngữ không hỗ trợ nested class.

Graphical user interface, application

Description automatically generated **Giải pháp 3:** Sử dụng Memento như một wrapper với component.

Khác với 2 giải pháp đầu, giải pháp 3 sử dụng Memento như 1 wrapper của component. **ConcreteMemento** lúc này sẽ giữ tham chiếu đến component.

Interface **Memento** có hàm **restore**. Khi hàm này được gọi, component của Memento sẽ được gán lại state của Memento hiện tại.

**StateManager**: quản lí các Memento của component. StateManager có thể quản lí nhiều memento của nhiều component khác nhau.

* 1. ***Các hệ quả mang lại:***

- Đảm bảo nguyên tắc **Single Responsibility**: lúc này việc quản lí history của state do 1 thành phần khác đảm nhiệm.

- Đảm bảo nguyên tắc **Open/Closed**: cung cấp 1 access modifier phù hợp, không để lộ các thông tin quan trọng của component

*Nhược điểm:*

- Tốn bộ nhớ nếu lưu trữ quá nhiều thông tin trong memento hoặc quản lí quá nhiều memento.

- StateManager cần phải theo dõi trong trường hợp component không còn tồn tại. Trong trường hợp vòng đời của component đã kết thúc thì StateManager phải có giải pháp để loại bỏ memento tương ứng của component đó.

***1.5 Các mẫu liên quan:***

**Command**

A picture containing shape

Description automatically generatedKết hợp với Command để hiện thực hóa chức năng undo.

1. ***Observer:***

***1.1 Tổng quan:***

- Tên: Abstract Factory.

- Phân loại: Abstract Factory thuộc nhóm Creational Pattern.

- Bí danh: Fabrique Abstraite.

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.2 Motivation:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.3 Khả năng ứng dụng:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.4 Cấu trúc:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.5 Vai trò và mối quan hệ các thành phần:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.6 Mã nguồn minh hoạ:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.7 Các hệ quả mang lại:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.8 Ví dụ trong các hệ thống thực tế:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.9 Các mẫu liên quan:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

1. ***State:***

***1.1 Tổng quan:***

- Tên: Abstract Factory.

- Phân loại: Abstract Factory thuộc nhóm Creational Pattern.

- Bí danh: Fabrique Abstraite.

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.2 Motivation:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.3 Khả năng ứng dụng:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.4 Cấu trúc:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.5 Vai trò và mối quan hệ các thành phần:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.6 Mã nguồn minh hoạ:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.7 Các hệ quả mang lại:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.8 Ví dụ trong các hệ thống thực tế:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.9 Các mẫu liên quan:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

1. ***Strategy:***

***1.1 Tổng quan:***

- Tên: Strategy,

- Phân loại: Abstract Factory thuộc nhóm Behavioral Pattern.

- Mục đích, ý định: Tách các “**chiến lược**” cụ thể cho một hành vi nào đó thành các class khác nhau. Class đảm nhiệm vai trò chọn “chiến lược” và thực thi được gọi là **context.**

***1.3 Khả năng ứng dụng:***

* Khi có nhiều biến thể về cách giải quyết cho cùng 1 vấn đề và các biến thể này có thể thay thế lẫn nhau.
* Sử dụng khi cần tách việc cài đặt ra khỏi ngữ cảnh của class hiện tại (nhất là khi chi tiết việc cài đặt không quá quan trọng).
* Giải pháp hiệu quả để thay thế các toán tử điều kiện phức tạp.

Nhược điểm:

* Các chiến lược chịu ràng buộc bởi interface.
* Đôi khi sử dụng strategy sẽ làm phức tạp hóa vấn đề.

***1.4 Cấu trúc:***



Bao gồm 2 thành phần chính:

**Strategy:**  Bao gồm các thành phần quy định chiến lược thực thi. Cụ thể ở đây là 1 interface **Strategy** và 2 class **ConcreteStrategyA** và **ConcreteStrategyB**.

**Context:** Ngữ cảnh sử dụng của strategy. Context có giữ tham chiếu đến interface **Streategy**. Context có vai trò lựa chọn Strategy để thực thi ở client bằng hàm **setStrategy**.

***1.6 Mã nguồn minh hoạ:***

* Thiết kế 1 module để xác thực thông tin người dùng (authentication).
* Có 3 phương thức xác thực:
* Xác thực bằng tài khoản và mật khẩu.
* Xác thực bằng token
* Xác thực bằng bên thứ 3.

**Thiết kế**

**Route**: context của strategy. Route nắm giữ các thông tin của 1 request (body, headers, cookies, …) mà qua đó, có thể implement các Strategy để tận dụng những thông tin này để làm các chức năng riêng biệt.

**AuthStrategy**: là 1 interface quy định strategy xác thực người dùng.

**AuthResult**: 1 lớp quy định kết quả của quá trình xác thực.

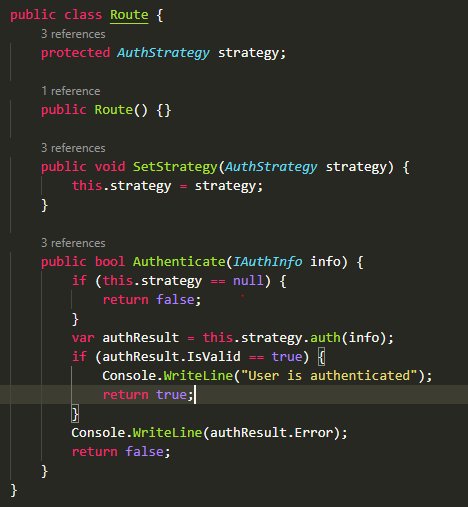
*(các strategy được implement trong demo)*

Bao gồm các phương thức xác thực:

**BasicStrategy**: Xác thực dựa trên username và password

**TokenStrategy**: Xác thực dựa trên token

**GoogleStrategy**: Xác thực thông qua bên thứ 3 là google

**Class Route:**

**Class BasicAuth:**



**Class TokenStrategy**



**Class GoogleStrategy:**



**Chương trình chính:**

***1.7 Các hệ quả mang lại:***

* Thể hiện rõ ràng nhất ý tưởng “Kết hợp hơn là kế thừa” (composition over inheritance). Vì kế thừa không phải là cách duy nhất để tái sử dụng code. Ngoài ra kế thừa còn có hạn chế đó là ràng class con phụ thuộc vào class cha, trong khi composition thì không.
* Abstraction over implementation.
* Tuân thủ nguyên tắc mở/đóng (**Open-Closed principle**).
* Tuân thủ nguyên tắc **Liskov substitution.**

***1.8 Ví dụ trong các hệ thống thực tế:***

**Passport**

Passport là 1 package được sử dụng khá phổ biến trong NodeJS. Cụ thể, passport hỗ trợ nhiều loại xác thực khác nhau bằng cách tách các loại xác thực thành nhiều Strategy khác nhau, qua đó giúp tối ưu hóa việc tái sử dụng.

***1.9 Các mẫu liên quan:***

* Cả **Strategy**, **Bridge** và **Adapter** đều sử dụng composition (thể hiện việc chuyển giao vấn đề cài đặt 1 lớp khác)
* Kết hợp với **Template Method**:
* Template Method quyết định thứ tự các công việc được thực hiện.
* Strategy: chịu trách nhiệm thực thi công việc.

1. ***Template Method:***

***1.1 Tổng quan:***

- Tên: Abstract Factory.

- Phân loại: Abstract Factory thuộc nhóm Creational Pattern.

- Bí danh: Fabrique Abstraite.

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.2 Motivation:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.3 Khả năng ứng dụng:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.4 Cấu trúc:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.5 Vai trò và mối quan hệ các thành phần:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.6 Mã nguồn minh hoạ:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.7 Các hệ quả mang lại:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.8 Ví dụ trong các hệ thống thực tế:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

***1.9 Các mẫu liên quan:***

- Mục đích, ý định: Cung cấp 1 interface cho việc tạo lập các đối tượng (có liên hệ với nhau) mà không cần quy định lớp khi tạo mỗi đối tượng.

1. ***Visitor:***

***1.1 Tổng quan:***

* Tên: Visitor.
* Phân loại: Visitor thuộc nhóm Behavioral Pattern.
* Mục đích, ý định: gồm 2 thành phần:
* Visitor: là thành phần cài đặt các hàm mà **Element** cần gọi.
* Element: Thành phần của hệ thống có vai trò lựa chọn hàm cần gọi ở **Visitor.**

***1.2 Khả năng ứng dụng:***

* Khi có quá nhiều loại đối tượng cần mở rộng thêm cùng 1 chức năng mới.
* Tách các logic không liên quan tới thành phần chính và cài đặt chúng trong một thành phần khác.
* Nếu chức năng mới cần cài đặt chỉ tương thích với một số thành phần nhất định.

***1.4 Cấu trúc:***



Gồm 2 thành phần:

**Element**: Bao gồm các interface và class của hệ thống đã có từ trước. Mỗi class Element có hàm **accept**. Hàm này nhân vào tham số là 1 interface **Visitor**.

**Visitor:** Thành phần chứa các hàm để các đối tượng trong Element có thể gọi được. Mỗi hàm **visit** như trong hình sẽ được cài đặt tương ứng cho mỗi class bên trong Element.

* 1. ***Mã nguồn minh hoạ:***

**Đặt vấn đề:**

- Một bệnh viện cần quản lí thông tin của nhân viên. Có 3 loại nhân viên: Bác sĩ, y tá, bảo vệ.

- Thông tin quản lí bao gồm các thông tin cơ bản như: họ tên, ngày sinh, giới tính.

+ Bác sĩ: trình độ học vấn.

+ Bảo vệ: công ty đã thuê.

(thiết kế ban đầu của chương trình)

**Mở rông yêu cầu:**

Hệ thống cần tính và tổng hợp lương của các nhân viên. Trong đó, mức lương được tính như sau:

- Bác sĩ: lương = 1.5 x lương cơ bản

- Y tá: lương = số ngày làm việc x 300,000.

- Bảo vệ: công ty đã thuê sẽ chịu trách nhiệm chi trả.

*Giải pháp 1:*

Lúc này ta có thể thêm hàm **getSalary** vào trong các class của chương trình. Tuy nhiên, giải pháp này không hiệu quả vì việc tính lương không phải là nhiệm vụ của các class này. Do đó việc cài đặt sẽ gây ra sự phụ thuộc không cần thiết.

*Giải pháp 2:* Sử dụng **Visitor**



***1.6 Các hệ quả mang lại:***

- Tuân thủ nguyên tắc đóng / mở (Open/Closed Principle).

- Tuân thủ nguyên tắc Single Responsibility. Đảm bảo Element chỉ tập trung vào chức năng chính của nó. Việc mở rộng các hành vi được thực thi bên trong Visitor.

- Có thể hoạt động hiệu quả trên các cấu trúc phức tạp của Element (cây, đồ thị, …).

*Nhược điểm:*

- Khi có 1 element thay đổi thì tất cả Visitor có cài đặt của element đó cũng sẽ có khả năng bị thay đổi.

- Visitor không có quyền truy cập hoàn toàn tới Element.

* Cần public một số thông tin từ Element.
* Vi phạm quy tắc đóng gói.

***1.7 Các mẫu liên quan:***

**- Composite**: có thể sử dụng Visitor để mở rộng và thực thi hành vi cho các thành phần của Composite.

1. ***Tài liệu tham khảo:***

\*Các tài liệu tham khảo:

- Gang of Four 4.0:

\*Các trang web tham khảo:

- stackoverflow.com

- refactoring.guru

- medium.com

- …

**NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

\_\_\_\_\_\_*\_\_\_\_\_\_ , ngày \_\_\_ tháng \_\_\_ năm 2021*

***Người nhận xét***