TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

KHOA CÔNG NGHỆ THÔNG TIN

🙥🕮🙧

A blue and yellow logo

Description automatically generated

BÁO CÁO BÀI TẬP LỚN

ĐẶC TẢ PHẦN MỀM

Cài đặt các mẫu thiết kế

|  |  |
| --- | --- |
| **Giảng viên** | : Trần Văn Dũng |
| **Sinh viên thực hiện** | : Vũ Bảo Lâm |
| **Mã sinh viên** | : 211241205 |
| **Lớp** | : Công nghệ thông tin 1 |
| **Khóa** | : 62 |

**Hà Nội, Năm 2024**

---------------o0o---------------

**MỤC LỤC**

[1. Mẫu Bridge 2](#_Toc182425288)

[2. Mẫu thiết kế Flyweight 3](#_Toc182425289)

[3. Mẫu thiết kế Decorator 5](#_Toc182425290)

[4. Mẫu thiết kế State 7](#_Toc182425291)

[5. Mẫu thiết kế Mediator 8](#_Toc182425292)

[6. Mẫu thiết kế Observer 11](#_Toc182425293)

[7. Mẫu thiết kế Strategy 12](#_Toc182425294)

[8. Mẫu thiết kế Prototype 14](#_Toc182425295)

[9. Mẫu thiết kế Factory 15](#_Toc182425296)

[10. Mẫu thiết kế Builder 17](#_Toc182425297)

1. Mẫu Bridge
   1. Phát biểu bài toán

Giả sử chúng ta có một ứng dụng vẽ hình học, và chúng ta cần vẽ các hình (chẳng hạn tam giác và ngũ giác) với nhiều màu vẽ khác nhau (như đỏ và xanh). Thay vì tạo các lớp cho từng kết hợp giữa hình và màu vẽ (ví dụ: RedTriangle, GreenPentagon, v.v.), chúng ta sẽ sử dụng **Bridge Pattern** để tách biệt hình học và màu vẽ.

Mục đích sử dụng Bridge Pattern:

* Tách rời phần trừu tượng và phần triển khai: Có thể thay đổi hoặc mở rộng phần hình học (như thêm hình Square, Hexagon) mà không cần thay đổi phần màu sắc. Ngược lại, có thể thêm màu sắc mới (như Yellow, Black) mà không cần thay đổi các hình hiện có.
* Dễ dàng mở rộng: Dễ dàng mở rộng hệ thống bằng cách thêm các lớp con cho Shape hoặc Color mà không ảnh hưởng đến các lớp khác.
  1. Giải thích mô hình các lớp và các phương thức cài đặt

Abstraction (Shape): Lớp Shape tham chiếu đến Color gọi phương thức applyColor ()

A screenshot of a computer code

Description automatically generated

Các lớp hình cụ thể gồm Pentagon và Triangle

A screenshot of a computer code

Description automatically generated

Implementor (Color): Giao diện Color định nghĩa phương thức applyColor ()

Lớp kiểm tra (BridgePatternTest): Đây là lớp chính, nơi tạo ra các đối tượng hình với màu sắc tương ứng. Khi gọi draw (), hình sẽ được vẽ với màu đã được truyền vào khi khởi tạo.

A screenshot of a computer code

Description automatically generated

* 1. Sơ đồ lớp và kết quả chạy chương trình

A diagram of a diagram

Description automatically generated

Kết quả chạy chương trình:



1. Mẫu thiết kế Flyweight
   1. Phát biểu bài toán

Giả sử chúng ta đang phát triển một trò chơi trong đó có rất nhiều loại cây được hiển thị trên màn hình (ví dụ: rừng với hàng ngàn cây). Mỗi cây có một vị trí (x, y), nhưng tất cả các cây cùng loại có chung một hình ảnh và các thuộc tính khác. Thay vì tạo ra hàng ngàn đối tượng cây với cùng thông tin hình ảnh, chúng ta có thể sử dụng mẫu **Flyweight** để chia sẻ các thuộc tính chung giữa các cây cùng loại.

* 1. Giải thích mô hình các lớp và các phương thức cài đặt

Flyweight (TreeType): Lớp TreeType lưu trữ các thông tin chung của cây (như tên, màu sắc, và texture). Những thông tin này có thể chia sẻ giữa nhiều cây giống nhau.

A screenshot of a computer code

Description automatically generated

ConcreteFlyweight: Trong ví dụ này, TreeType chính là ConcreteFlyweight, nó quản lý việc hiển thị thông tin chung của cây.

FlyweightFactory (TreeFactory): Lớp này chịu trách nhiệm tạo và quản lý các đối tượng TreeType. Khi một loại cây mới được yêu cầu, TreeFactory sẽ kiểm tra xem đối tượng TreeType đó đã tồn tại chưa, nếu đã tồn tại thì trả về đối tượng đó, nếu chưa có thì tạo mới.

A screenshot of a computer code

Description automatically generated

Context (Tree): Lớp Tree chứa các thuộc tính không thể chia sẻ (như vị trí x, y) và một tham chiếu đến đối tượng TreeType. Các thuộc tính không thể chia sẻ là duy nhất cho từng cây, trong khi TreeType có thể được chia sẻ giữa nhiều cây.

A screenshot of a computer code

Description automatically generated

Client (Forest): Đây là lớp Forest, quản lý nhiều cây và sử dụng mẫu Flyweight để giảm bớt việc tạo ra các đối tượng TreeType trùng lặp.

A screenshot of a computer code

Description automatically generated

Lợi ích của mẫu Flyweight:

* Giảm số lượng đối tượng được tạo ra bằng cách chia sẻ đối tượng. Vì vậy, tiết kiệm bộ nhớ và các thiết bị lưu trữ cần thiết.
* Hiệu suất cao hơn: Việc quản lý và tái sử dụng các đối tượng giúp giảm thời gian và tài nguyên cần thiết cho việc khởi tạo các đối tượng tương tự.
  1. Sơ đồ lớp và kết quả chạy chương trình

**A diagram of a tree and forest

Description automatically generated**

Kết quả chạy chương trình:

A screenshot of a computer

Description automatically generated

1. Mẫu thiết kế Decorator
   1. Phát biểu bài toán

Một cửa hàng burger muốn cung cấp cho khách hàng tùy chọn để tự do xây dựng burger của riêng họ với các thành phần như phô mai, thịt xông khói, xà lách, và nhiều hơn nữa. Cửa hàng có một số loại burger cơ bản như Burger cơ bản (Basic Burger), Burger chay (Vegetarian Burger), và Burger đặc biệt (Special Burger). Mỗi loại burger có giá ban đầu khác nhau, và mỗi thành phần bổ sung đều có giá riêng.

Khách hàng có thể chọn loại burger cơ bản, sau đó thêm các thành phần bổ sung. Ví dụ:

* Một khách hàng có thể chọn Vegetarian Burger và thêm phô mai và xà lách.
* Một khách khác có thể chọn Special Burger và thêm thịt xông khói, phô mai, và xà lách.

Cửa hàng muốn một hệ thống linh hoạt cho phép thêm các thành phần tùy ý vào các loại burger cơ bản mà không cần phải tạo ra quá nhiều lớp mới cho từng loại kết hợp.

* 1. Giải thích mô hình các lớp và phương thức cài đặt

Giao diện Burger - Định nghĩa các phương thức cơ bản mà tất cả các loại burger và lớp trang trí cần triển khai

A close-up of a computer code

Description automatically generated

Lớp cụ thể (Concrete Component): BasicBurger, VegetarianBurger, và SpecialBurger là các lớp cụ thể triển khai Burger. Mỗi lớp đại diện cho một loại burger với mô tả và giá cơ bản riêng.

A screenshot of a computer code

Description automatically generated

Lớp Decorator: BurgerDecorator là lớp trừu tượng thực thi giao diện Burger và lưu trữ một tham chiếu tới đối tượng Burger (đối tượng gốc hoặc đối tượng đã trang trí).

A screenshot of a computer program

Description automatically generated

Lớp trang trí cụ thể (Concrete Decorators): Các lớp cụ thể như CheeseDecorator, LettuceDecorator, và BaconDecorator kế thừa BurgerDecorator. Mỗi lớp trang trí cụ thể này thêm một thành phần vào burger, đồng thời cập nhật mô tả và giá.

* 1. Sơ đồ lớp và kết quả chạy chương trình

A diagram of a network

Description automatically generated

Kết quả chạy chương trình:

A screenshot of a computer

Description automatically generated

1. Mẫu thiết kế State
   1. Phát biểu bài toán

Trong một hệ thống máy rửa bát tự động, máy có thể ở một trong nhiều trạng thái khác nhau trong suốt quá trình vận hành, bao gồm: chờ (Idle), đã nạp chén đĩa (Loaded), đang rửa (Washing), và hoàn tất (Completed). Mỗi trạng thái có các hành vi và quy tắc xử lý riêng biệt:

* Khi máy ở trạng thái chờ, nó có thể nhận thêm chén đĩa để bắt đầu chu trình rửa.
* Sau khi nạp chén đĩa, máy chuyển sang trạng thái đã nạp chén đĩa và có thể bắt đầu chu trình rửa.
* Trong trạng thái đang rửa, máy sẽ hoàn tất quy trình sau một khoảng thời gian và chuyển sang trạng thái hoàn tất.
* Sau khi quá trình rửa hoàn tất, người dùng có thể lấy chén đĩa ra khỏi máy.

Vấn đề nảy sinh ở chỗ, mỗi trạng thái chỉ cho phép một số hành động nhất định. Ví dụ, không thể bắt đầu rửa nếu chưa nạp chén đĩa, hoặc không thể lấy chén đĩa ra khi máy chưa hoàn thành chu trình. Nếu không có hệ thống quản lý trạng thái rõ ràng, người dùng có thể gặp lỗi hoặc sự cố khi thực hiện các thao tác không hợp lệ.

Mẫu thiết kế State là lựa chọn phù hợp cho bài toán quản lý trạng thái máy rửa bát nhờ khả năng tách biệt hành vi từng trạng thái, dễ mở rộng, bảo trì, và quản lý trạng thái động một cách hiệu quả. Nó giúp mã nguồn gọn gàng hơn, giảm thiểu lỗi và cải thiện trải nghiệm của người dùng.

* 1. Giải thích mô hình các lớp và phương thức cài đặt

Giao diện State: Định nghĩa các hành vi chung mà mọi trạng thái của máy rửa bát cần triển khai.

A screenshot of a computer program

Description automatically generated

Lớp quản lý Dishwasher ()

A screenshot of a computer

Description automatically generated

* 1. Sơ đồ lớp và kết quả chạy chương trình

A diagram of a diagram

Description automatically generated

Kết quả chạy chương trình:

A white background with black text

Description automatically generated

1. Mẫu thiết kế Mediator
   1. Phát biểu bài toán

Trong một sân bay, có nhiều máy bay cần quản lý quá trình hạ cánh. Nếu các máy bay giao tiếp trực tiếp với nhau để xác định thứ tự hạ cánh, hệ thống sẽ trở nên phức tạp và dễ xảy ra xung đột. Khi có máy bay hạ cánh, các máy bay khác cần được thông báo về tình trạng đường băng (đã có sẵn hoặc đang bận). Một cách tiếp cận tốt là sử dụng một cơ chế trung gian (mediator) để quản lý các tương tác này, nhằm giảm thiểu sự phụ thuộc giữa các máy bay.

* 1. Giải thích mô hình các lớp và phương thức cài đặt

AirportMediator(): là đối tượng trung gian đóng vai trò như "trạm kiểm soát không lưu", quản lý việc hạ cánh của các máy bay.

A screenshot of a computer program

Description automatically generated

Các máy bay (lớp Aircraft) sẽ liên kết với AirportMediator khi chúng được khởi tạo. Mỗi máy bay sẽ được "đăng ký" với Mediator thông qua phương thức registerAircraft().

A screenshot of a computer program

Description automatically generated

Khi một máy bay muốn hạ cánh, nó sẽ gọi phương thức requestToLand() để gửi yêu cầu đến AirportMediator.

* Phương thức requestToLand() sẽ yêu cầu Mediator kiểm tra xem đường băng có trống không thông qua phương thức notifyAircraftToLand().
* Mediator sẽ kiểm tra trạng thái đường băng bằng phương thức canLand(). Nếu đường băng trống, máy bay được phép hạ cánh, và Mediator sẽ thông báo điều này bằng cách hiển thị thông điệp "is landing".
* Nếu đường băng không trống, máy bay sẽ phải chờ.

Sau khi máy bay được phép hạ cánh, nó sẽ hoàn tất việc hạ cánh bằng cách gọi phương thức land().

A screenshot of a computer program

Description automatically generated

* Phương thức land() sẽ hiển thị thông báo rằng máy bay đã hạ cánh và gọi mediator.aircraftLanded() để thông báo cho Mediator rằng quá trình hạ cánh đã kết thúc.
* Sau khi máy bay hạ cánh, Mediator sẽ giải phóng đường băng bằng cách đặt lại trạng thái đường băng (runwayFree = true) và thông báo rằng đường băng đã trống.

Khi đường băng đã được giải phóng, máy bay khác (ví dụ: aircraft2) có thể yêu cầu hạ cánh. Mediator sẽ kiểm tra trạng thái đường băng và cấp quyền hạ cánh nếu nó trống.

* 1. Sơ đồ lớp và kết quả chạy chương trình

A diagram of a computer

Description automatically generated

Kết quả chạy chương trình:

A screenshot of a computer

Description automatically generated

1. Mẫu thiết kế Observer
   1. Phát biểu bài toán

Một hệ thống cần thông báo cho nhiều đối tượng về một thay đổi trạng thái. Ví dụ, trong một ứng dụng thương mại điện tử, khi có một sản phẩm mới được nhập về, hệ thống cần thông báo cho tất cả các khách hàng đã đăng ký theo dõi sản phẩm đó. Nếu số lượng khách hàng thay đổi, hệ thống vẫn phải hoạt động mà không cần thay đổi logic thông báo. Yêu cầu cho hệ thống:

* Hệ thống phải dễ dàng thêm hoặc bớt đối tượng cần nhận thông báo.
* Khi trạng thái thay đổi, tất cả đối tượng liên quan cần được thông báo tự động.
* Không phụ thuộc cứng giữa các đối tượng để tăng tính mở rộng.

**Giải pháp:** Sử dụng **Observer Pattern**, nơi một đối tượng (Subject) quản lý danh sách các đối tượng khác (Observers) và thông báo cho họ khi có thay đổi.

* 1. Giải thích mô hình các lớp và các phương thức cài đặt

Interface Observer: phương thức update(String message) mà tất cả các Observer phải triển khai.

A close up of a sign

Description automatically generated

Interface Subject: Quản lý danh sách các Observer và định nghĩa các phương thức: **addObserver()**: Đăng ký Observer; **removeObserver()**: Hủy đăng ký Observer; **notifyObservers()**: Gửi thông báo cho tất cả Observer.

A close-up of a computer code

Description automatically generated

Lớp Product: Khi trạng thái thay đổi (ví dụ: sản phẩm mới đến), nó sẽ gọi notifyObservers() để thông báo cho tất cả các khách hàng đã đăng ký.

A screenshot of a computer

Description automatically generated

Lớp Customer: Khi nhận được thông báo, hiển thị thông báo đến người dùng.

A screenshot of a computer

Description automatically generated

* 1. Sơ đồ lớp và kết quả chạy chương trình

A diagram of a product

Description automatically generated

Kết quả chạy chương trình:

A screenshot of a computer

Description automatically generated

1. Mẫu thiết kế Strategy
   1. Phát biểu bài toán

Một hệ thống có nhiều cách thực hiện cùng một nhiệm vụ, và các cách này có thể thay đổi tùy thuộc vào ngữ cảnh. Ví dụ, trong một ứng dụng giao hàng, chi phí giao hàng có thể được tính dựa trên nhiều chiến lược khác nhau: theo khoảng cách, theo trọng lượng, hoặc theo tốc độ giao hàng. Sử dụng **Strategy Pattern** để tách rời thuật toán tính toán chi phí giao hàng ra khỏi logic chính. Mỗi chiến lược sẽ được triển khai như một lớp riêng biệt, có thể thay đổi linh hoạt.

* 1. Giải thích mô hình các lớp và các phương thức cài dặt

**Interface**ShippingCostStrategy: phương thức calculateCost(), mà mọi chiến lược tính toán chi phí phải triển khai.

A close up of a text

Description automatically generated

* DistanceBasedStrategy: Tính chi phí dựa trên khoảng cách.
* WeightBasedStrategy: Tính chi phí dựa trên trọng lượng.
* SpeedBasedStrategy: Tính chi phí dựa trên tốc độ giao hàng.

Lớp ShippingService: Sử dụng chiến lược đã được chọn để tính toán chi phí. Cho phép thay đổi chiến lược bất kỳ lúc nào bằng cách sử dụng phương thức setStrategy().

A screenshot of a computer program

Description automatically generated

Lớp Main:

A screenshot of a computer program

Description automatically generated

* 1. Sơ đồ lớp và kết quả chạy chương trình

A diagram of a business

Description automatically generated

Kết quả chạy chương trình

A close-up of a sign

Description automatically generated

1. Mẫu thiết kế Prototype
   1. Phát biểu bài toán

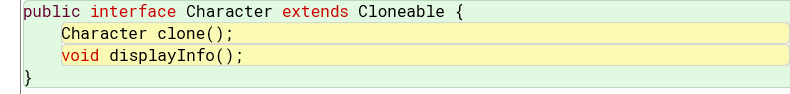
Trong một trò chơi trực tuyến, có rất nhiều loại nhân vật (character) như Warrior, Mage, Archer, mỗi nhân vật có các thuộc tính như tên, kỹ năng, cấp độ, sức mạnh, và trang bị.

Khi người chơi tạo một nhân vật mới, việc khởi tạo toàn bộ thuộc tính và cấu hình lại từ đầu (ví dụ, kỹ năng, vũ khí, cấp độ ban đầu) có thể tốn thời gian và phức tạp. Thay vào đó, ta cần một cách để nhanh chóng sao chép các nhân vật mẫu (prototype) với các thuộc tính cơ bản đã được định sẵn và tùy chỉnh thêm nếu cần.

**Lợi ích sử dụng mẫu prototye cho bài toán:**

* Cung cấp cách thức sao chép nhanh các đối tượng nhân vật dựa trên mẫu (prototype).
* Giảm thiểu chi phí và thời gian tạo đối tượng mới từ đầu.
* Dễ dàng tùy chỉnh các nhân vật được sao chép từ prototype.
  1. Giải thích mô hình các lớp và các phương thức cài đặt

Interface Character: là giao diện cung cấp phương thức để sao chép đối tượng. với lớp clone () để tạo bản sao của đối tượng.



Lớp trừu tượng Character: Định nghĩa các thuộc tính và hành vi chung cho các nhân vật trong game như: name, level, skilss.

A screenshot of a computer program

Description automatically generated

Các lớp nhân vật Warrior, Mage, Archer, mỗi lớp này kế thừa từ Character và tuỳ chỉnh thêm thuộc tính.

Lớp Game: Quản lý các prototype mẫu và cung cấp khả năng sao chép, tạo bản sao mới từ Prototype dựa trên loại nhân vật

A screenshot of a computer code

Description automatically generated

* 1. Sơ dồ lớp và kết quả chạy chương trình

A diagram of a computer generated diagram

Description automatically generated with medium confidence

Kết quả chạy chương trình

A screenshot of a computer

Description automatically generated

1. Mẫu thiết kế Factory
   1. Phát biểu bài toán

Một hệ thống phần mềm quản lý động vật trong vườn thú cần tạo ra các đối tượng động vật (như Sư Tử, Hổ, Sói). Mỗi loại động vật có hành vi và đặc điểm riêng biệt. Tuy nhiên, nếu trực tiếp tạo từng đối tượng bằng cách sử dụng new trong mã nguồn, khi thêm một loại động vật mới, bạn phải sửa đổi tất cả các nơi liên quan. Điều này làm tăng nguy cơ lỗi và giảm tính linh hoạt.

Giải pháp: Factory Design Pattern

* Tạo ra một cách thống nhất để khởi tạo các đối tượng động vật mà không cần thay đổi mã nguồn chính.
* Ẩn logic khởi tạo cụ thể của từng loại động vật, đảm bảo mã nguồn chính không bị phụ thuộc trực tiếp vào các lớp động vật cụ thể.
  1. Giải thích mô hình các lớp và các phương thức cài đặt

**Interface Animal**: Định nghĩa hành động chung cho các động vật.

A close up of a screen

Description automatically generated

Các lớp động vật cụ thể: Lion, Tiger, Wolf.

**Class Factory AnimalFactory**: Chịu trách nhiệm tạo các đối tượng động vật dựa trên yêu cầu.

A screenshot of a computer code

Description automatically generated

**Lớp Zoo**: Sử dụng Factory để tạo động vật mà không cần biết chi tiết bên trong từng loại động vật.

A screenshot of a computer program

Description automatically generated

* 1. Sơ dồ lớp và kết quả chạy chương trình

A diagram of a company

Description automatically generated

Kết quả chạy chương trình

A white background with black text

Description automatically generated

1. Mẫu thiết kế Builder
   1. Phát biểu bài toán

Một ứng dụng thương mại điện tử cần tạo các đơn hàng cho khách hàng. Mỗi đơn hàng có thể bao gồm: thông tin cơ bản (ID đơn hàng, tên khách hàng), danh sách sản phẩm (tên, số lượng, giá), tùy chọn thanh toán (trả trước, trả sau), tùy chọn giao hàng (giao nhanh, giao tiêu chuẩn). Tuy nhiên khi có nhiều tuỳ chọn thì việc khởi tạo đối tượng đơn hàng sẽ trở nên phức tạp. Nếu sử dụng constructor với nhiều tham số, mã nguồn sẽ trở nên khó đọc và dễ bị lỗi.

Giải pháp cho bài toán là sử dụng mẫu Builder Pattern. Lợi ích của việc sử dụng mẫu là làm giảm sự phức tạp khi khởi tạo các đối tượng với nhiều tùy chọn mà không cần constructor quá tải. Tính linh hoạt cao dễ tuỳ chỉnh các thuộc tính tuỳ chọn mà không thay đổi thuộc tính mặc định. Khi cần thêm thuộc tính mới, chỉ cần mở rộng Builder mà không cần thay đổi mã nguồn chính.

* 1. Giải thích mô hình các lớp và các phương thức cài đặt

Lớp Customer: Quản lý thông tin chi tiết của khách hàng.

A screenshot of a computer program

Description automatically generated

Lớp Order: Đại diện cho một đơn hàng, liên kết với một khách hàng.

A screen shot of a computer code

Description automatically generated

Interface OrderBuilderInterface: Đảm bảo mọi Builder thực hiện đầy đủ các phương thức cần thiết.

A screenshot of a computer

Description automatically generated

Lớp OrderBuilder: Cung cấp các phương thức tạo đối tượng Order và các getter và setter

A screenshot of a computer

Description automatically generated

Hàm Main:

A screenshot of a computer program

Description automatically generated

* 1. Sơ đồ lớp và kết quả chạy chương trình

A diagram of a computer

Description automatically generated

Kết quả chạy chương trình:

A screen shot of a computer

Description automatically generated