Design Pattern

Creational Pattern

1. Factory Method
2. Abstract Factory
3. Prototype
4. Singleton

Structural Pattern

1. Adapter
2. Bridge
3. Composite
4. Decorator
5. Facade
6. Proxy

Behavioral Pattern

1. Template Method
2. Chain of Responsibility
3. Command
4. Iterator
5. Mediator
6. Observer
7. State
8. Strategy

Creational Pattern

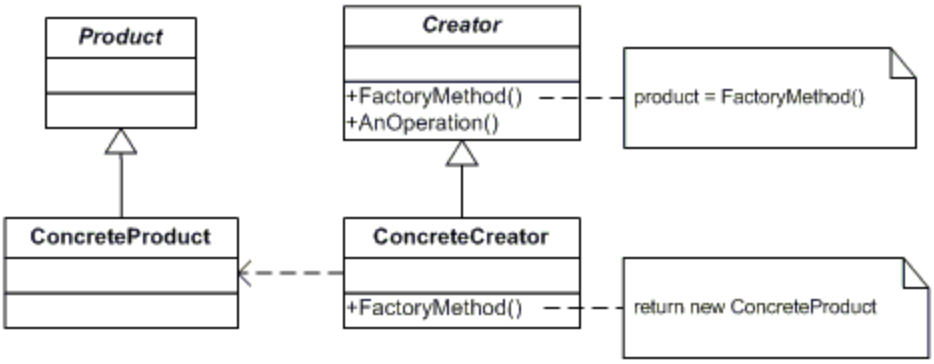
Factory Method

Loại: Creational Pattern

Factory Method định nghĩa một interface cho việc tạo ra một object, nhưng các lớp con sẽ quyết định lớp nào được khởi tạo. Mẫu này cho phép một lớp trì hoãn việc khởi tạo thành các lớp con.

Tần suất sử dụng: cao

UML

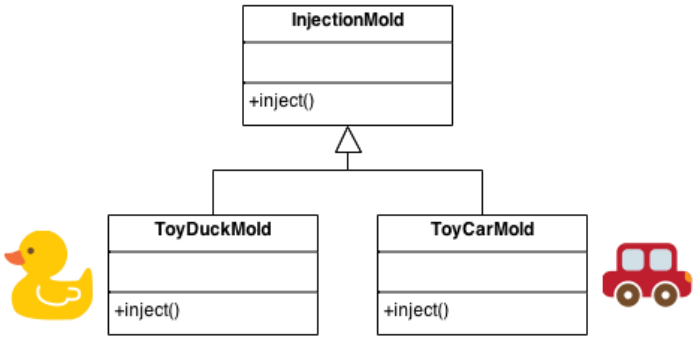


Thành phần

- *Product*: định nghĩa interface của những đối tượng mà factory method tạo ra
- *ConcreteProduct*: thực thi interface Product
- *Creator*
    - Khai báo factory method, trả về một object thuộc loại Product. Creator cũng có thể xác định một sự thực thi mặc định của factory method trả về một object ConcreteProduct mặc định
    - Có thể gọi factory method để tạo ra một object Product
- *ConcreteCreator*: ghi đè factory method để trả về một instance của ConcreteProduct

Ví dụ

Những nhà máy sản xuất đồ chơi nhựa đúc ra các khuôn, và bơm nhựa vào khuôn mẫu có hình dạng mong muốn. Loại đồ chơi (xe, nhân vật hoạt hình, …) sẽ được xác định bởi khuôn.



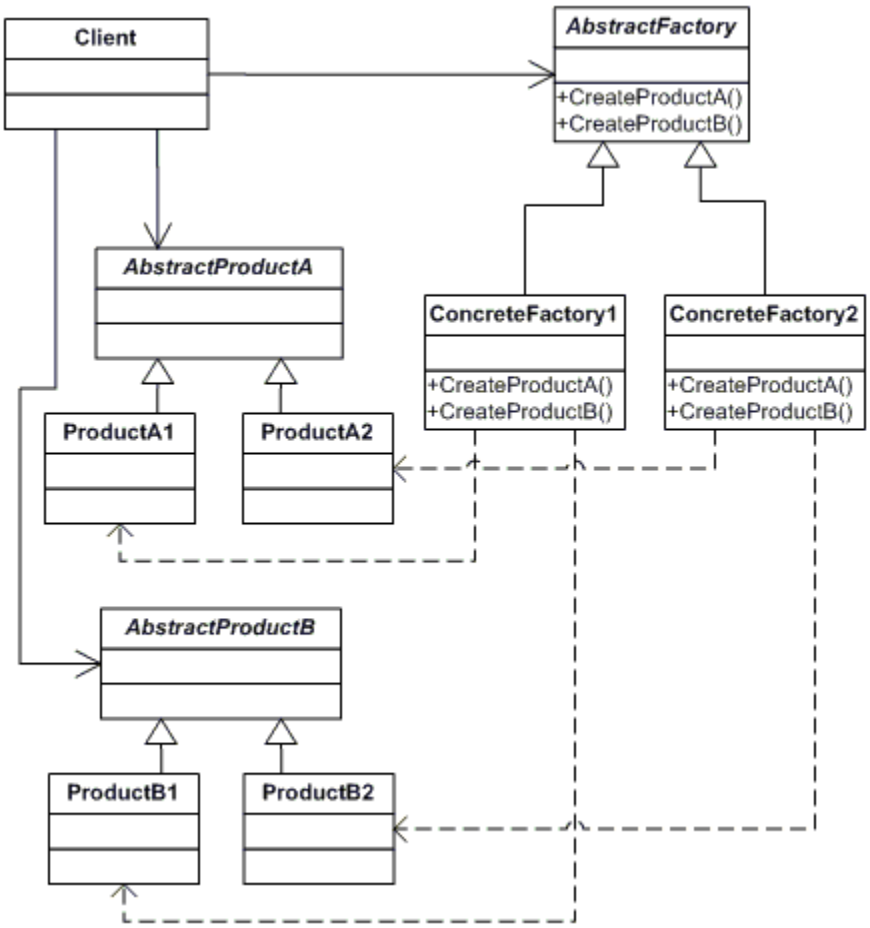
Abstract Factory

Loại: Creational Pattern

Mẫu Abstract Factory cung cấp một interface cho việc tạo ra họ những đối tượng liên quan hoặc phụ thuộc mà không chỉ định các lớp cụ thể của chúng.

Tần suất sử dụng: cao

UML



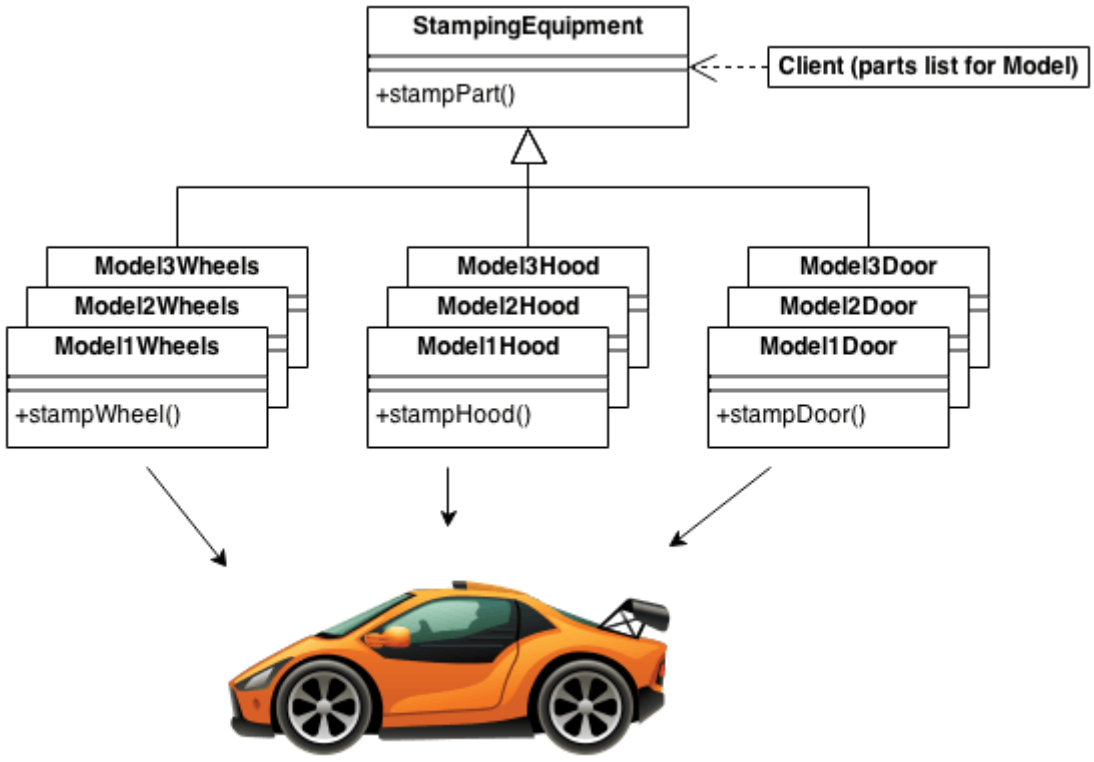
Thành phần

- *AbstractFactory*: khai báo một interface cho việc tạo ra những AbstractProduct

- *ConcreteFactory*: thực thi các hoạt động để tạo ra các object Product cụ thể
- *AbstractProduct*: khai báo một interface cho một loại object Product
- *Product*
    - Định nghĩa một object Product được tạo ra bởi Factory cụ thể tương ứng
    - Thực thi interface AbstractProduct
- *Client*: sử dụng những interface được khai báo bởi các lớp AbstractFactory và AbstractProduct

Ví dụ

Mục đích của AbstractFactory là cung cấp một interface để tạo các họ đối tượng liên quan mà không chỉ định các lớp cụ thể. Mẫu này được tìm thấy trong thiết bị dập kim loại tấm được sử dụng trong sản xuất ô tô của Nhật Bản. Thiết bị dập là một AbstractFactory tạo ra các bộ phận thân xe. Cùng một loại máy dùng để dán cửa phải, cửa trái, chắn bùn trước phải, chắn bùn trước trái, mui xe, … cho các dòng xe ô tô khác nhau. Thông qua việc sử dụng các con lăn để thay đổi khuôn dập, các lớp cụ thể do máy tạo ra có thể được thay đổi trong vòng ba phút.



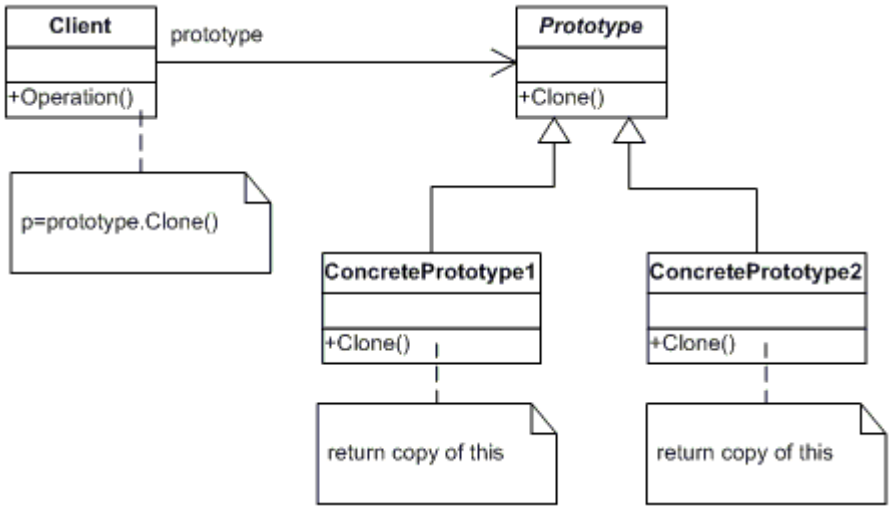
Prototype

Loại: Creational Pattern

Mẫu Prototype chỉ định loại object cần tạo bằng cách sử dụng prototypical instance, và tạo các object mới bằng cách copy các nguyên mẫu này.

Tần suất sử dụng: bình thường

UML

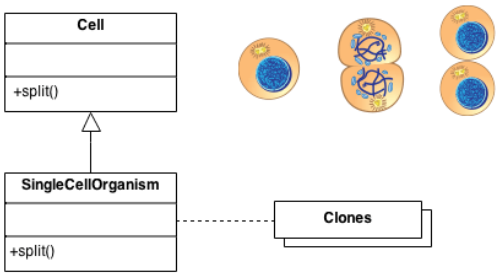


Thành phần

- *Prototype*: khai báo một interface để nhân bản chính nó
- *ConcretePrototype*: thực thi hoạt động để nhân bản chính nó
- *Client*: tạo một object mới bằng cách yêu cầu một nguyên mẫu sao chép chính nó

Ví dụ

Mẫu Prototype chỉ định loại object sẽ tạo bằng cách sử dụng một prototypical instance. Nguyên mẫu của sản phẩm mới thường được xây dựng trước khi sản xuất hoàn chỉnh, nhưng trong ví dụ này, nguyên mẫu là thụ động và không tham gia vào quá trình sao chép chính nó. Sự phân chia nguyên phân của một tế bào - kết quả là tạo ra hai tế bào giống hệt nhau - là một ví dụ về nguyên mẫu đóng vai trò tích cực trong việc sao chép chính nó và do đó, thể hiện mẫu Nguyên mẫu. Khi một tế bào phân chia sẽ tạo ra hai tế bào có kiểu gen giống hệt nhau. Nói cách khác, tế bào tự nhân bản chính nó.



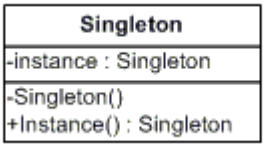
Singleton

Loại: Creational Pattern

Mẫu Singleton đảm bảo một lớp chỉ có một instance và cung cấp một điểm truy cập toàn cục cho nó.

Tần suất sử dụng: trung bình - cao

UML

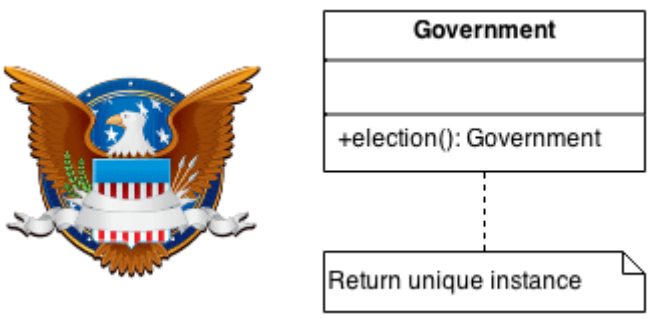


Thành phần

- *Singleton*
  - Định nghĩa một thao tác Instance cho phép client truy cập instance duy nhất của nó. Instance là một class operation
  - Chịu trách nhiệm tạo và duy trì instance duy nhất của riêng mình

Ví dụ

Mẫu Singleton đảm bảo rằng một lớp chỉ có một thể hiện và cung cấp một điểm truy cập toàn cục cho thể hiện đó. Nó được đặt tên theo tập hợp singleton, được định nghĩa là tập hợp chứa một phần tử. Văn phòng của Tổng thống Hoa Kỳ là một Singleton. Hiến pháp Hoa Kỳ quy định cách thức bầu tổng thống, giới hạn nhiệm kỳ và xác định thứ tự kế vị. Kết quả là, có thể có nhiều nhất một tổng thống đang hoạt động tại bất kỳ thời điểm nào. Bất kể danh tính cá nhân của tổng thống đang hoạt động là gì, chức danh "Tổng thống Hoa Kỳ" là một điểm truy cập toàn cầu xác định người trong văn phòng.



Structural Pattern

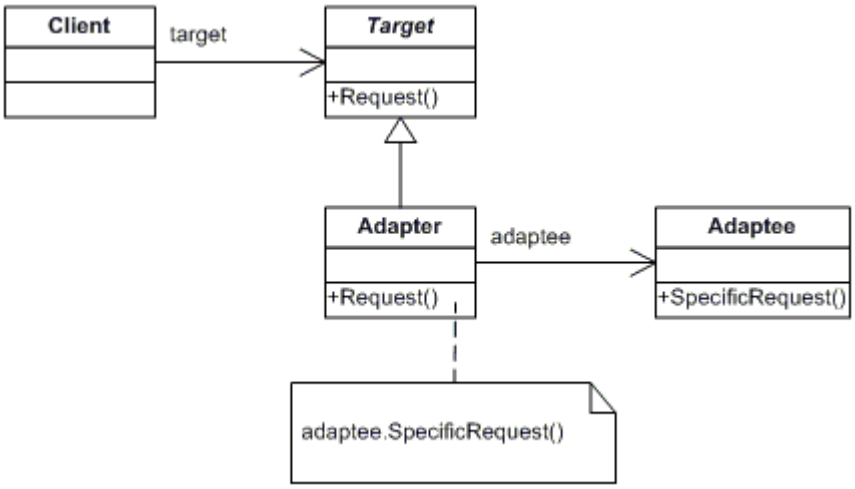
Adapter

Loại: Structural Pattern

Mẫu Adapter chuyển đổi interface của một class thành một interface khác mà client mong đợi. Mẫu thiết kế này cho phép các class làm việc cùng nhau mà không thể làm việc khác do interface không tương thích.

Tần suất sử dụng: trung bình - cao

UML

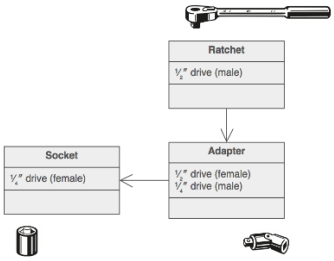


Thành phần

- *Target*: xác định interface dành riêng cho domain-specific mà *Client* sử dụng
- *Adapter*: điều chỉnh interface *Adaptee* với interface *Target*
- *Adaptee*: xác định một giao diện hiện có cần điều chỉnh
- *Client*: cộng tác với các đối tượng phù hợp với interface *Target*

Ví dụ

Mẫu Adapter cho phép các lớp không tương thích làm việc cùng nhau bằng cách chuyển đổi interfacee của một lớp thành interface mà *Client* mong đợi. Cờ lê ổ cắm cung cấp một ví dụ về Adapter. Một ổ cắm gắn vào một bánh cóc, với điều kiện là kích thước của ổ đĩa là như nhau. Kích thước ổ điển hình ở Hoa Kỳ là 1/2" và 1/4". Rõ ràng, bánh cóc truyền động 1/2" sẽ không vừa với ổ cắm truyền động 1/4" trừ khi sử dụng adapter. Bộ chuyển đổi 1/2" đến 1/4" có đầu cái 1/2" để vừa với bánh cóc truyền động 1/2" và đầu nối đực 1/4" để vừa với ổ cắm 1/4".



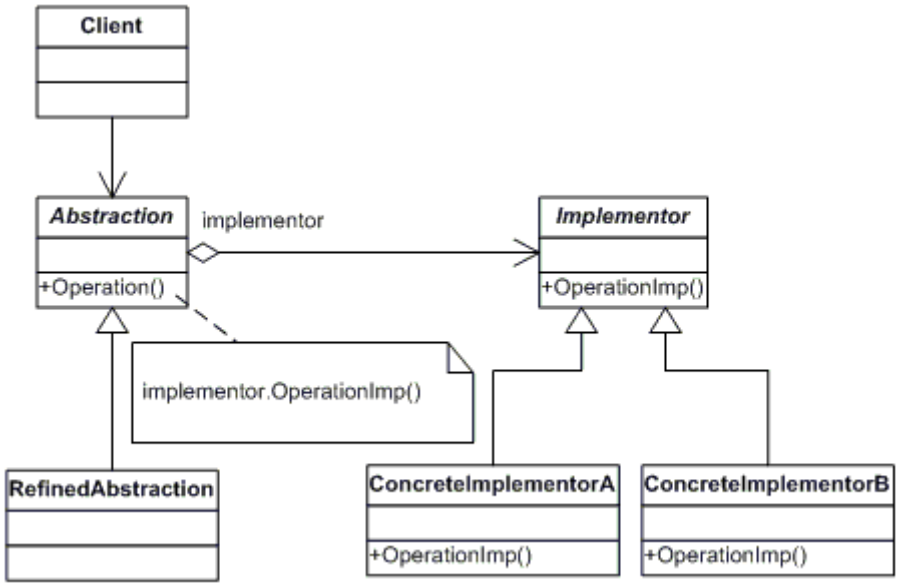
Bridge

Loại: Structural Pattern

Mẫu thiết kế Bridge tách rời một phần trừu tượng khỏi phần thực thi của nó để cả hai có thể thay đổi độc lập.

Tần suất sử dụng: trung bình – thấp

UML



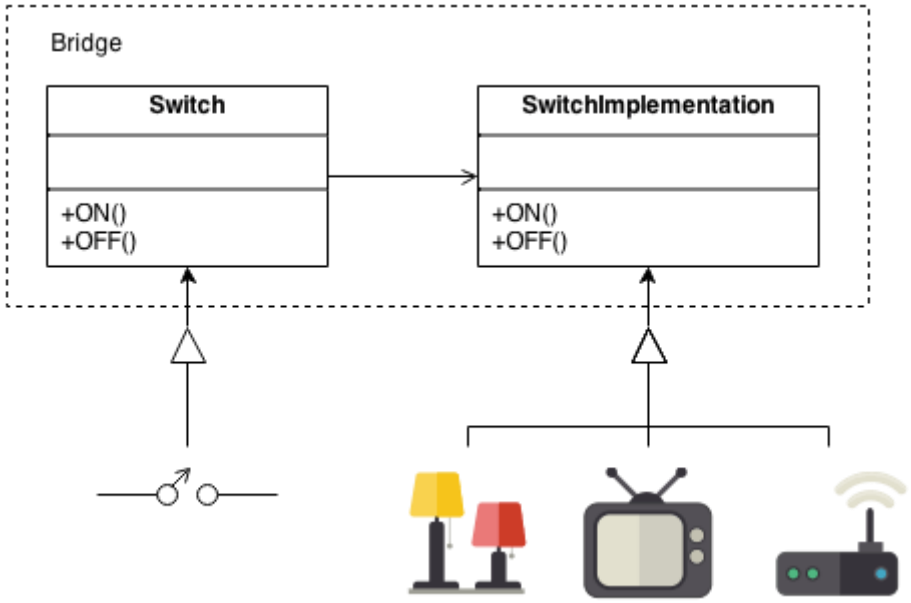
Thành phần

- *Abstraction*
  - Định nghĩa abstract interface
  - Duy trì một tham chiếu đến một đối tượng kiểu Implementor
- *RefinedAbstractio*
  - Mở rộng interface được xác định bởi Abstraction
- *Implementor*
  - Định nghĩa interface cho các lớp thực thi. Interface này không nhất thiết phải tương ứng chính xác với interface của Trừu tượng; trên thực tế, hai interface có thể khá khác nhau. Thông thường, interface Implementation chỉ cung cấp các hoạt động nguyên thủy và Abstraction xác định các hoạt động cấp cao hơn dựa trên các hoạt động nguyên thủy này
- *ConcreteImplementor*

- Thực thi interface Implementor và xác định việc thực thi cụ thể của nó

Ví dụ

Mẫu Bridge tách rời một phần trừu tượng khỏi phần triển khai của nó, để cả hai có thể thay đổi độc lập. Một công tắc gia dụng điều khiển đèn, quạt trần, v.v. là một ví dụ về Bridge. Mục đích của công tắc là bật hoặc tắt thiết bị. Công tắc thực tế có thể được triển khai dưới dạng chuỗi kéo, công tắc hai vị trí đơn giản hoặc nhiều loại công tắc điều chỉnh độ sáng.



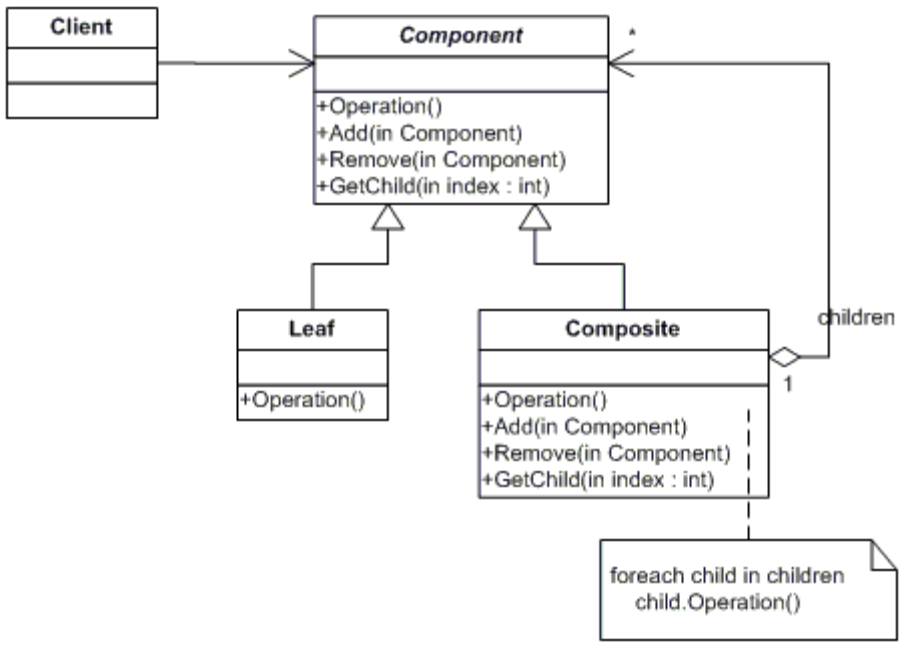
Composite

Loại: Structural Pattern

Mẫu thiết kế Composite kết hợp các đối tượng thành cấu trúc cây để biểu thị cấu trúc phân cấp từng phần-toàn bộ. Mẫu này cho phép client xử lý các đối tượng riêng lẻ và thành phần của các đối tượng một cách thống nhất.

Tần suất sử dụng: trung bình - cao

UML

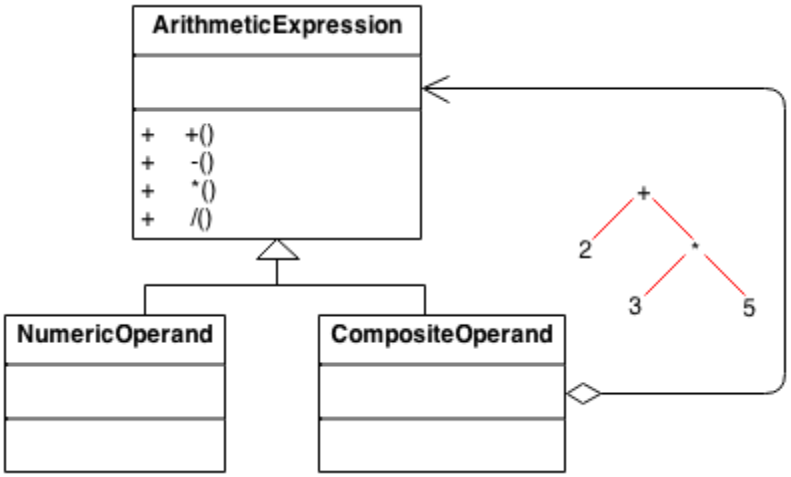


Thành phần

- *Component*
  - Khai báo interface cho các đối tượng trong thành phần
  - Thực thi hành vi mặc định cho interface chung cho tất cả các lớp, khi thích hợp
  - Khai báo một interface để truy cập và quản lý các thành phần con của nó
  - (tùy chọn) xác định interface để truy cập phần tử cha của thành phần trong cấu trúc đệ quy và thực thi interface đó nếu điều đó phù hợp
- *Leaf*
  - Đại diện cho các đối tượng lá trong thành phần. Chiếc lá không có con
  - Xác định hành vi cho các đối tượng nguyên thủy trong thành phần
- *Composite*
  - Xác định hành vi cho các thành phần có con
  - Lưu trữ các thành phần con
  - Thực thi các hoạt động liên quan đến con trong interface Component
- *Client*
  - Thao tác với các đối tượng trong bố cục thông qua interface Component

Ví dụ

Composite kết hợp các đối tượng thành cấu trúc cây và cho phép Client xử lý các đối tượng và bố cục riêng lẻ một cách thống nhất. Mặc dù ví dụ này là trừu tượng, nhưng các biểu thức số học là hợp số. Một biểu thức số học bao gồm một toán hạng, một toán tử (+ - * /) và một toán hạng khác. Toán hạng có thể là một số hoặc một biểu thức số học khác. Do đó, 2 + 3 và (2 + 3) + (4 * 6) đều là các biểu thức hợp lệ.



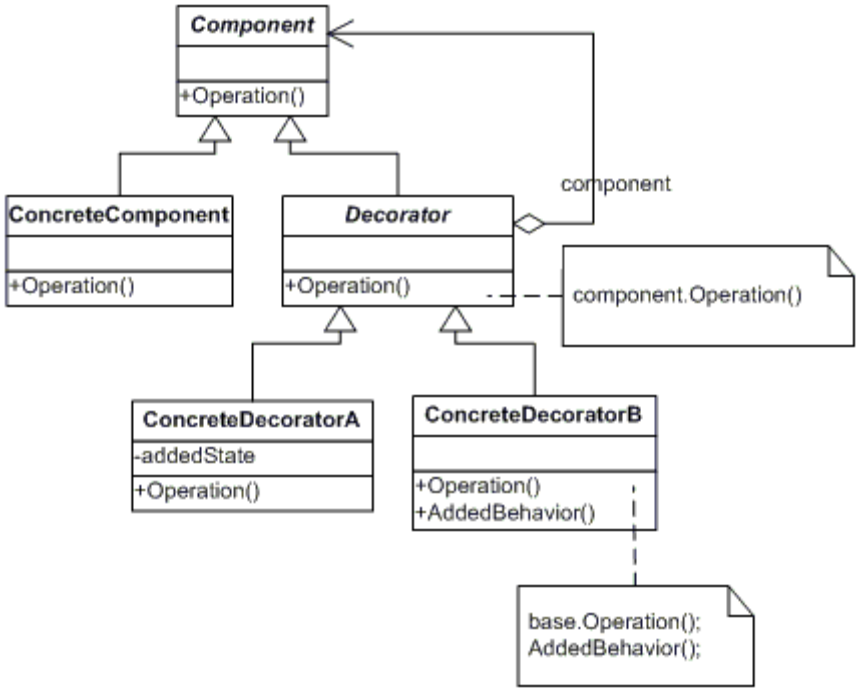
Decorator

Loại: Structural Pattern

Mẫu thiết kế Decorator gắn các trách nhiệm bổ sung cho một đối tượng một cách linh hoạt. Mẫu này cung cấp một giải pháp thay thế linh hoạt cho phân lớp để mở rộng chức năng.

Tần suất sử dụng: trung bình

UML



Thành phần

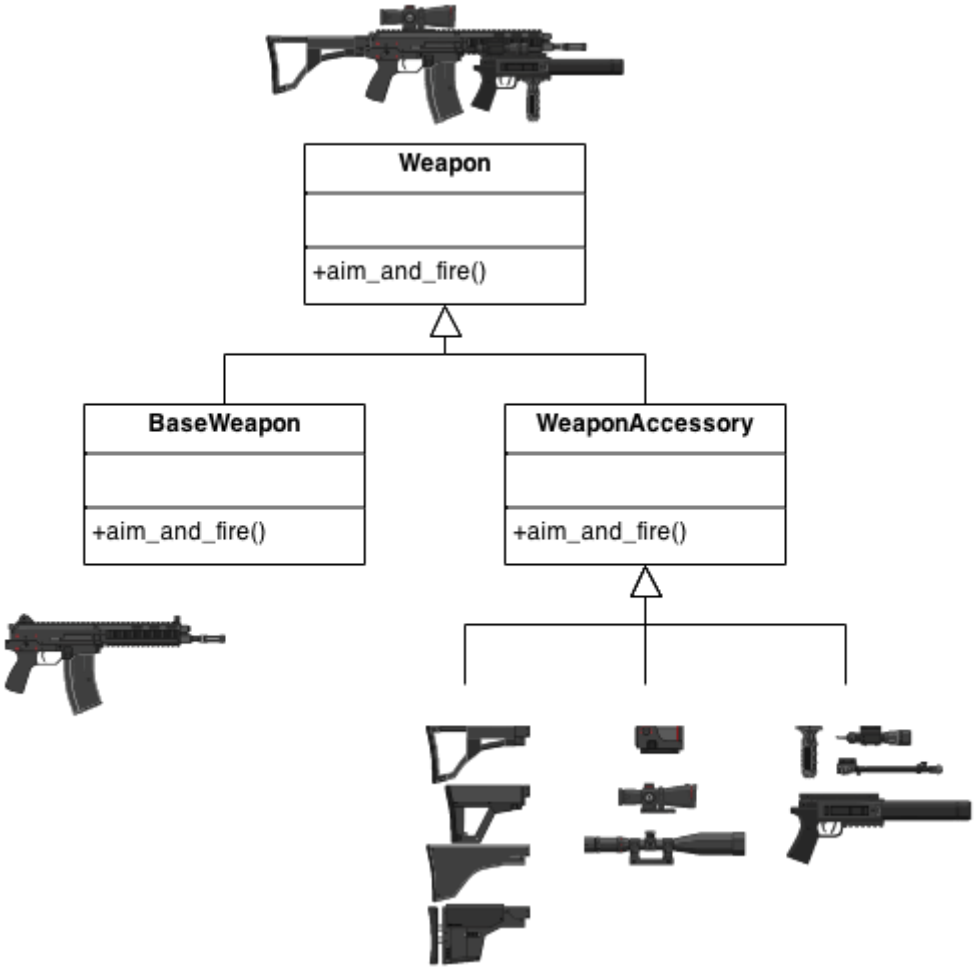
- *Component*
  - Xác định interface cho các đối tượng có thể có các trách nhiệm được thêm vào chúng một cách linh hoạt
- *Concretecomponent*
  - Định nghĩa một đối tượng mà các trách nhiệm bổ sung có thể được gắn vào
- *Decorator*
  - Duy trì tham chiếu đến đối tượng thành phần và xác định giao diện phù hợp với giao diện của thành phần
- *Concretedecorator*
  - Thêm trách nhiệm cho thành phần

Ví dụ

Decorator gắn các trách nhiệm bổ sung cho một đối tượng một cách linh hoạt. Các đồ trang trí được thêm vào cây thông hoặc cây linh sam là những ví dụ về Decorator. Đèn,

vòng hoa, kẹo que, đồ trang trí bằng thủy tinh, v.v., có thể được thêm vào cây để mang lại vẻ lễ hội cho cây. Các đồ trang trí không làm thay đổi bản thân cây, có thể nhận ra là cây thông Noel bất kể đồ trang trí cụ thể nào được sử dụng. Như một ví dụ về chức năng bổ sung, việc bổ sung đèn cho phép một người "thắp sáng" cây thông Noel.

Một ví dụ khác: bản thân súng tấn công là một vũ khí chết người. Nhưng bạn có thể áp dụng một số "trang trí" nhất định để làm cho nó chính xác hơn, im lặng và tàn phá hơn.



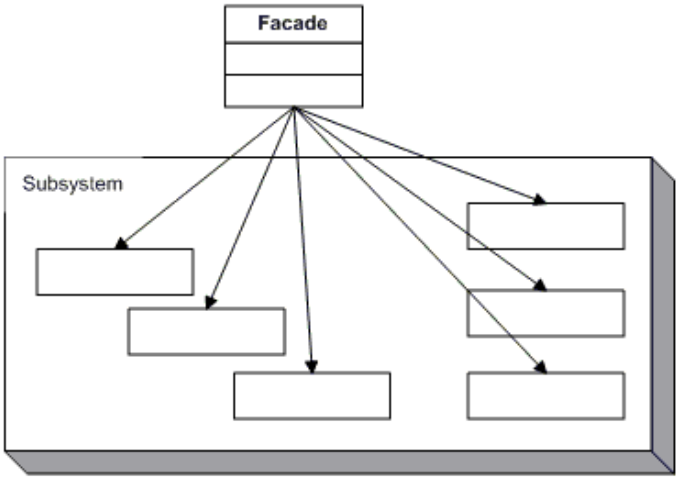
Facade

Loại: Structural Pattern

Mẫu thiết kế Facade cung cấp một interface hợp nhất cho một tập hợp các interface trong một hệ thống con. Mẫu này xác định interface cấp cao hơn giúp hệ thống con dễ sử dụng hơn.

Tần suất sử dụng: cao

UML

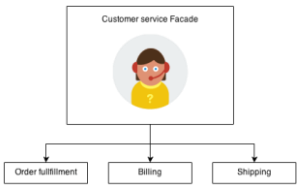


Thành phần

- *Façade*
  - Biết các lớp hệ thống con nào chịu trách nhiệm cho một yêu cầu
  - Ủy quyền các yêu cầu của Client cho các đối tượng hệ thống con thích hợp
- *Subsystem classes*
  - Thực thi chức năng hệ thống con
  - Xử lý công việc do đối tượng Facade giao
  - Không có kiến thức về Facade và không tham khảo nó

Ví dụ

Facade xác định một interface thống nhất, cấp cao hơn cho một hệ thống con giúp sử dụng dễ dàng hơn. Người tiêu dùng bắt gặp Facade khi đặt hàng từ một danh mục. Người tiêu dùng gọi một số và nói chuyện với đại diện dịch vụ khách hàng. Đại diện dịch vụ khách hàng hoạt động như một Facade, cung cấp interface cho bộ phận thực hiện đơn hàng, bộ phận thanh toán và bộ phận vận chuyển.



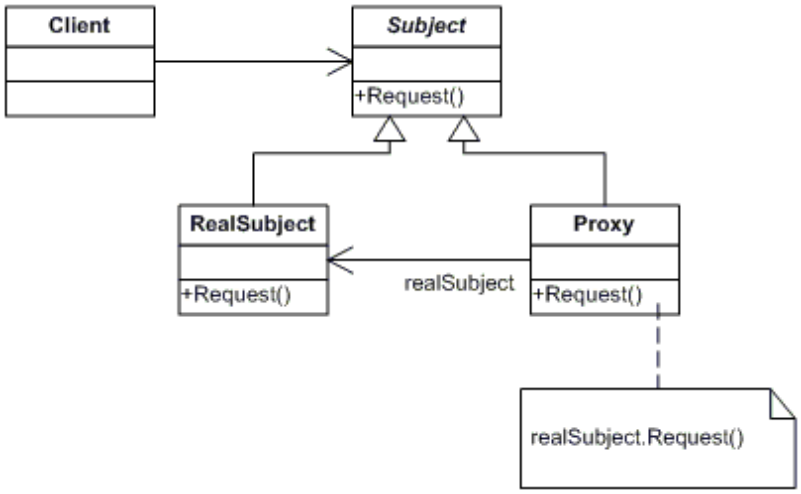
Proxy

Loại: Structural Pattern

Mẫu thiết kế Proxy cung cấp một trình thay thế hoặc trình giữ chỗ cho một đối tượng khác để kiểm soát quyền truy cập vào nó.

Tần suất sử dụng: trung bình - cao

UML

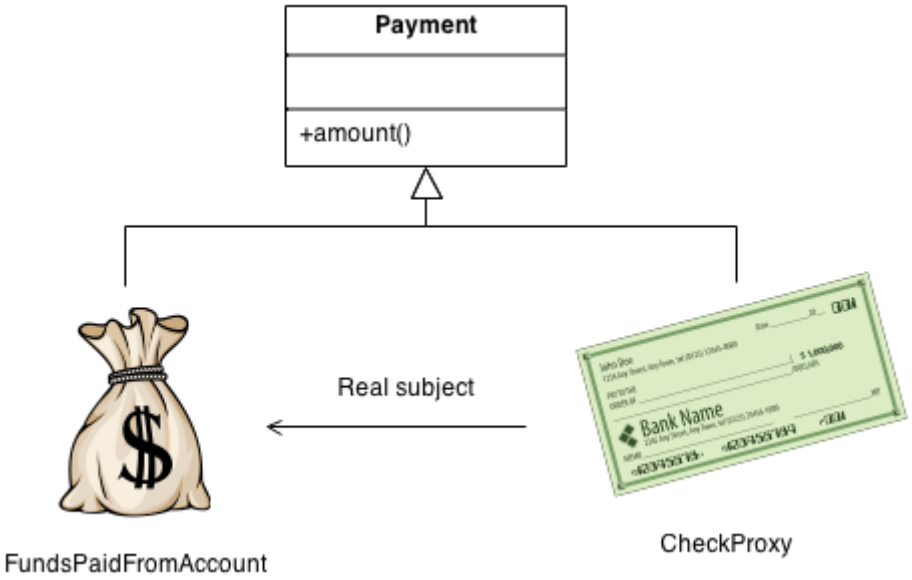


Thành phần

- *Proxy*
  - Duy trì một tham chiếu cho phép proxy truy cập chủ đề thực. Proxy có thể đề cập đến Subject nếu interface *RealSubject* và *Subject* giống nhau
  - Cung cấp một interface giống hệt với Subject để có thể thay thế proxy cho subject thực
  - Kiểm soát quyền truy cập vào subject thực và có thể chịu trách nhiệm tạo và xóa nó
  - Các trách nhiệm khác phụ thuộc vào loại proxy
    - Remote proxies chịu trách nhiệm mã hóa yêu cầu và các đối số của nó và gửi yêu cầu được mã hóa tới subject thực trong một không gian địa chỉ khác
    - Virtual proxies có thể lưu trữ thông tin bổ sung về chủ đề thực để họ có thể trì hoãn việc truy cập nó. Ví dụ: ImageProxy từ Motivation lưu trữ phạm vi của hình ảnh thực
    - Protection proxies bảo vệ kiểm tra xem người gọi có quyền truy cập cần thiết để thực hiện yêu cầu không
- *Subject*
  - Xác định interface chung cho RealSubject và Proxy để Proxy có thể được sử dụng ở bất kỳ đâu mà RealSubject được mong đợi
- *Realsubject*
  - Xác định đối tượng thực mà proxy đại diện

Ví dụ

Proxy cung cấp một người thay thế hoặc người giữ chỗ để cung cấp quyền truy cập vào một đối tượng. Séc hoặc hối phiếu ngân hàng là giấy ủy quyền cho các khoản tiền trong tài khoản. Séc có thể được sử dụng thay cho tiền mặt để mua hàng và cuối cùng kiểm soát quyền truy cập vào tiền mặt trong tài khoản của người phát hành.



Behavioral Pattern

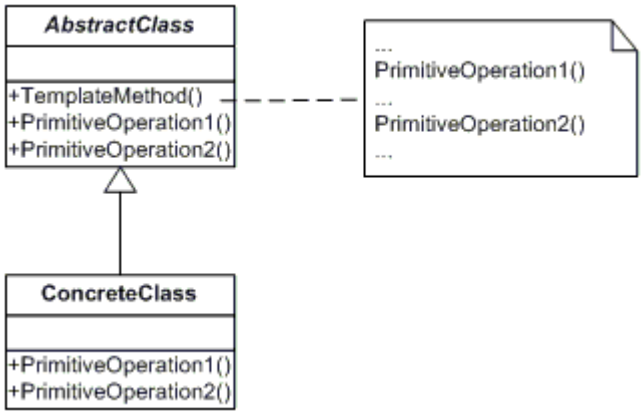
Template Method

Loại: Behavioral Pattern

Mẫu thiết kế Template Method xác định khung của thuật toán trong một thao tác, trì hoãn một số bước cho các lớp con. Mẫu này cho phép các lớp con xác định lại các bước nhất định của thuật toán mà không làm thay đổi cấu trúc của thuật toán.

Tần suất sử dụng: trung bình

UML



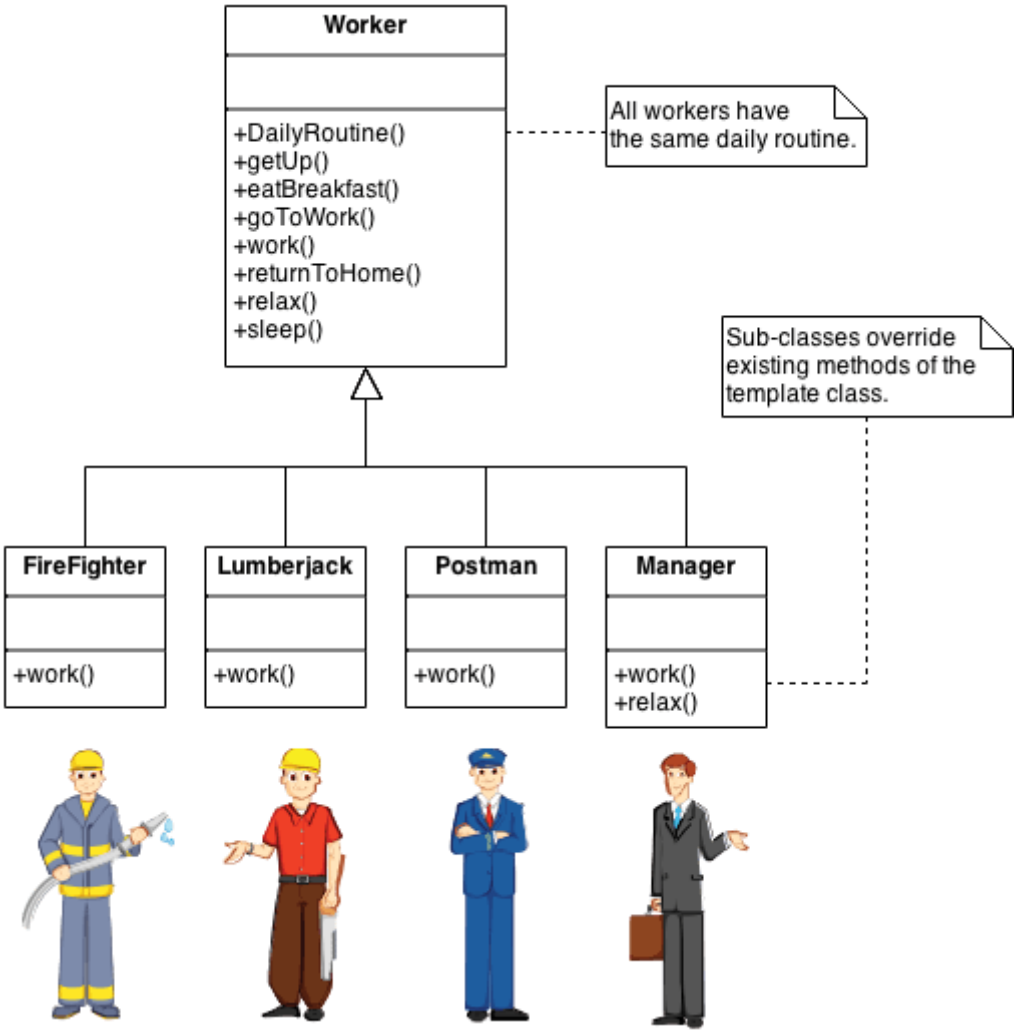
Thành phần

- *AbstractClass*
  - Xác định các *primitive operations* mà các lớp con cụ thể xác định để thực hiện các bước của thuật toán
  - Thực hiện một Template method xác định khung của thuật toán. Template method gọi các *primitive operations* cũng như các thao tác được định nghĩa trong *AbstractClass* hoặc của các đối tượng khác
- *Concreteclass*
  - Thực hiện các *primitive operations* để thực hiện các bước dành riêng cho lớp con của thuật toán

Ví dụ

Template method xác định bộ khung của thuật toán trong một thao tác và trì hoãn một số bước đối với các lớp con. Những người xây dựng nhà sử dụng Template method khi phát triển một phân khu mới. Một phân khu điển hình bao gồm một số sơ đồ tầng hạn chế với các biến thể khác nhau có sẵn cho mỗi loại. Trong một sơ đồ mặt bằng, nền móng, khung, hệ thống ống nước và hệ thống dây điện sẽ giống hệt nhau cho mỗi ngôi nhà. Biến thể được giới thiệu trong các giai đoạn xây dựng sau này để tạo ra nhiều kiểu mẫu hơn.

Một ví dụ khác: thói quen hàng ngày của một công nhân.



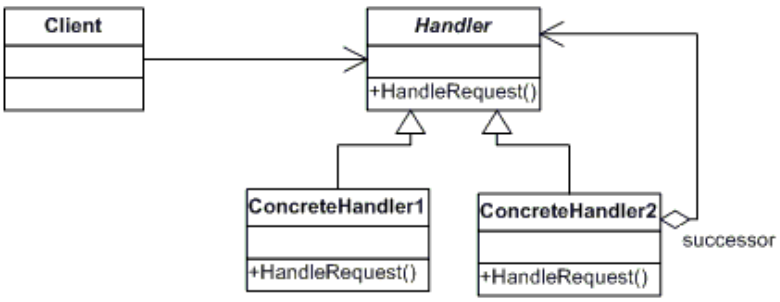
Chain of Responsibility

Loại: Behavioral Pattern

Mẫu thiết kế Chain of Responsibility tránh ghép người gửi yêu cầu với người nhận bằng cách cho nhiều đối tượng cơ hội xử lý yêu cầu. Mẫu này xâu chuỗi các đối tượng nhận và chuyển yêu cầu dọc theo chuỗi cho đến khi một đối tượng xử lý nó.

Tần suất sử dụng: trung bình – thấp

UML



Thành phần

- *Handler*
  - Định nghĩa một interface để xử lý các yêu cầu
  - (tùy chọn) thực hiện liên kết kế thừa
- *Concretehandler*
  - Xử lý các yêu cầu mà nó chịu trách nhiệm
  - Có thể truy cập người kế thừa của nó
  - Nếu *ConcreteHandler* có thể xử lý yêu cầu, nó sẽ làm như vậy; mặt khác, nó chuyển tiếp yêu cầu đến người kế thừa của nó
- *Client*
  - Bắt đầu yêu cầu đối tượng *ConcreteHandler* trên chuỗi

Ví dụ

Mẫu Chuỗi trách nhiệm tránh ghép người gửi yêu cầu với người nhận bằng cách cho nhiều đối tượng cơ hội xử lý yêu cầu. ATM sử dụng Chuỗi trách nhiệm trong cơ chế trao tiền.

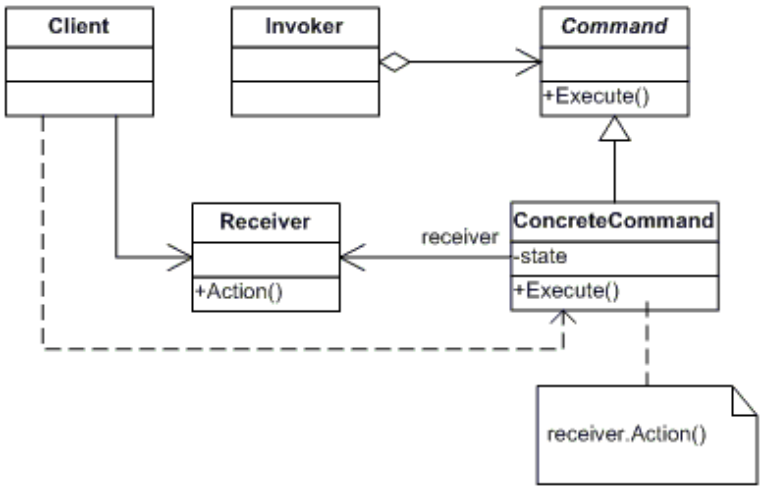
Command

Loại: Behavioral Pattern

Mẫu thiết kế Command đóng gói một yêu cầu dưới dạng một đối tượng, do đó cho phép bạn tham số hóa các ứng dụng khách với các yêu cầu, hàng đợi hoặc yêu cầu nhật ký khác nhau và hỗ trợ các thao tác không thể hoàn tác.

Tần suất sử dụng: trung bình - cao

UML



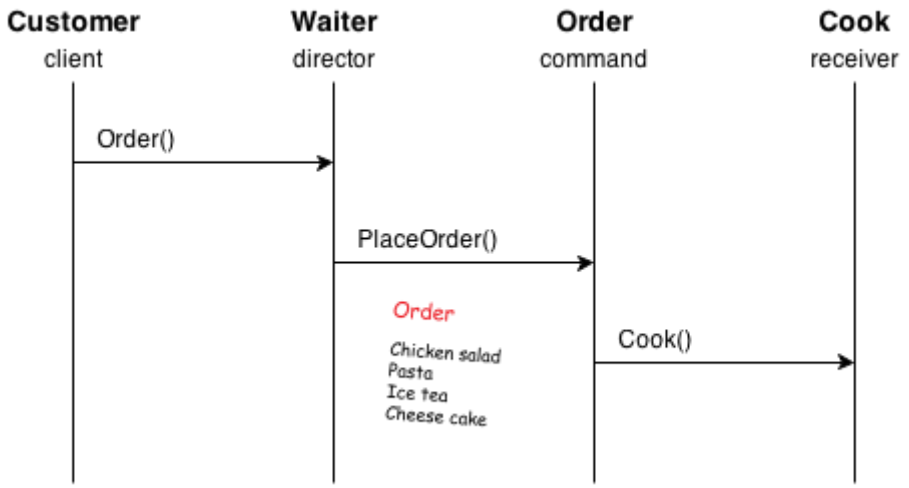
Thành phần

- *Command*
  - Khai báo một interface để thực thi một thao tác
- *ConcreteCommand*
  - Xác định một ràng buộc giữa một đối tượng *Receiver* và một hành động
  - Thực hiện thực thi bằng cách gọi (các) thao tác tương ứng trên *Receiver*
- *Client*
  - Tạo một đối tượng *ConcreteCommand* và đặt *Receiver* của nó
- *Invoker*
  - Yêu cầu C*ommand* thực hiện yêu cầu
- *Receiver*
  - Biết thực hiện các thao tác liên quan đến việc thực hiện yêu cầu

Ví dụ

Mẫu Command cho phép các yêu cầu được gói gọn dưới dạng các đối tượng, do đó cho phép các máy khách được tham số hóa với các yêu cầu khác nhau. Việc "kiểm tra" tại một quán ăn là một ví dụ về mẫu Command. Nhân viên phục vụ bàn nhận đơn đặt hàng hoặc yêu cầu từ khách hàng và đóng gói đơn hàng đó bằng cách viết vào séc. Đơn đặt hàng sau đó được xếp hàng cho một đầu bếp đặt hàng ngắn. Lưu ý rằng bảng "séc" được

sử dụng bởi mỗi người phục vụ không phụ thuộc vào menu và do đó họ có thể hỗ trợ các lệnh để nấu nhiều món khác nhau.



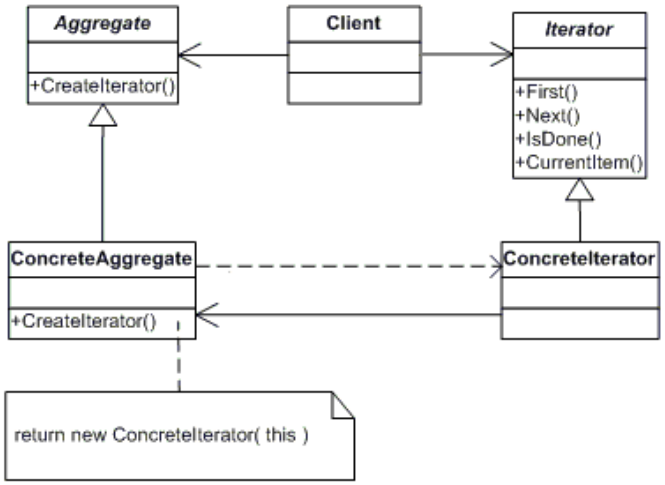
Iterator

Loại: Behavioral Pattern

Mẫu thiết kế Iterator cung cấp một cách để truy cập các phần tử của một đối tượng tổng hợp một cách tuần tự mà không để lộ biểu diễn cơ bản của nó.

Tần suất sử dụng: cao

UML



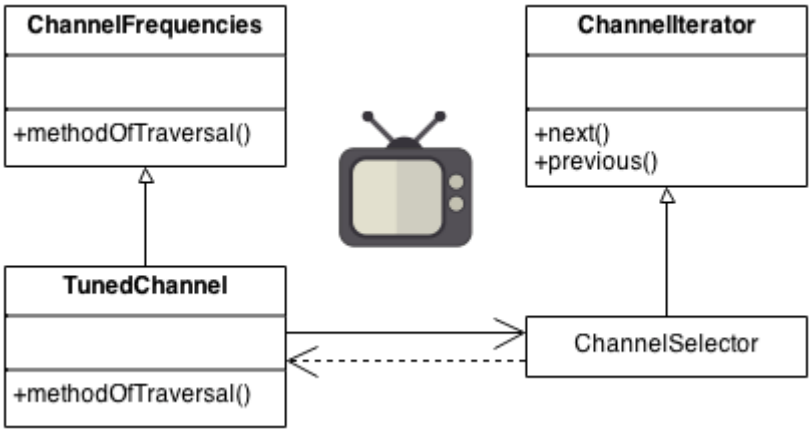
Thành phần

- *Iterator*
  - Định nghĩa một interface để truy cập và duyệt qua các phần tử
- *ConcreteIterator*
  - Thực hiện interface *Iterator*
  - Theo dõi vị trí hiện tại trong đường truyền của *Aggregate*
- *Aggregate*
  - Định nghĩa một interface để tạo một đối tượng *Iterator*
- *ConcreteAggregate*
  - Triển khai interface tạo *Iterator* để trả về một instance của *ConcreteIterator* thích hợp

Ví dụ

Iterator cung cấp các cách để truy cập các phần tử của một đối tượng tổng hợp một cách tuần tự mà không để lộ cấu trúc bên dưới của đối tượng. Tệp là đối tượng tổng hợp. Trong cài đặt văn phòng nơi quyền truy cập vào các tệp được thực hiện thông qua nhân viên hành chính hoặc thư ký, mẫu Iterator được thể hiện với thư ký đóng vai trò là Iterator. Một số tiểu phẩm hài trên truyền hình đã được phát triển xoay quanh tiền đề là một giám đốc điều hành đang cố gắng hiểu hệ thống nộp hồ sơ của thư ký. Đối với giám đốc điều hành, hệ thống hồ sơ khó hiểu và phi logic, nhưng thư ký có thể truy cập hồ sơ nhanh chóng và hiệu quả.

Trên những chiếc tivi đời đầu, một nút xoay được sử dụng để chuyển kênh. Khi lướt kênh, người xem phải di chuyển nút xoay qua từng vị trí của kênh, bất kể kênh đó có thu sóng hay không. Trên TV hiện đại, nút tiếp theo và trước đó được sử dụng. Khi người xem chọn nút "tiếp theo", kênh đã dò tiếp theo sẽ được hiển thị. Cân nhắc việc xem tivi trong phòng khách sạn ở một thành phố xa lạ. Khi lướt qua các kênh, số kênh không quan trọng, mà là chương trình. Nếu chương trình trên một kênh không được quan tâm, người xem có thể yêu cầu kênh tiếp theo mà không cần biết số của kênh đó.



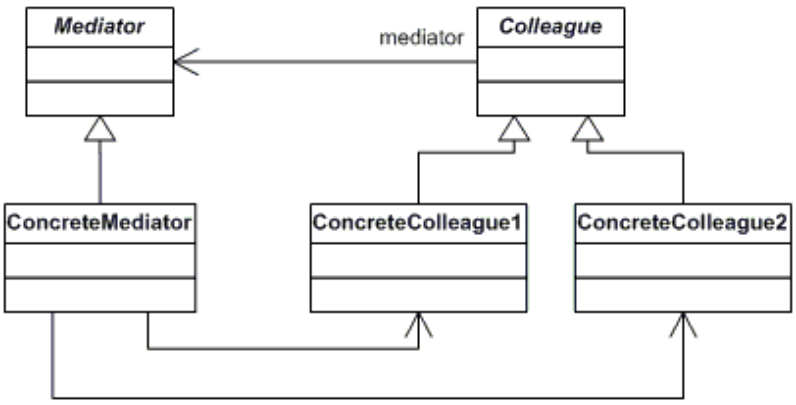
Mediator

Loại: Behavioral Pattern

Mẫu thiết kế Mediator xác định một đối tượng bao hàm cách một tập hợp các đối tượng tương tác. Mediator thúc đẩy khớp nối lỏng lẻo bằng cách giữ cho các đối tượng không đề cập đến nhau một cách rõ ràng và nó cho phép bạn thay đổi tương tác của chúng một cách độc lập.

Tần suất sử dụng: trung bình – thấp

UML

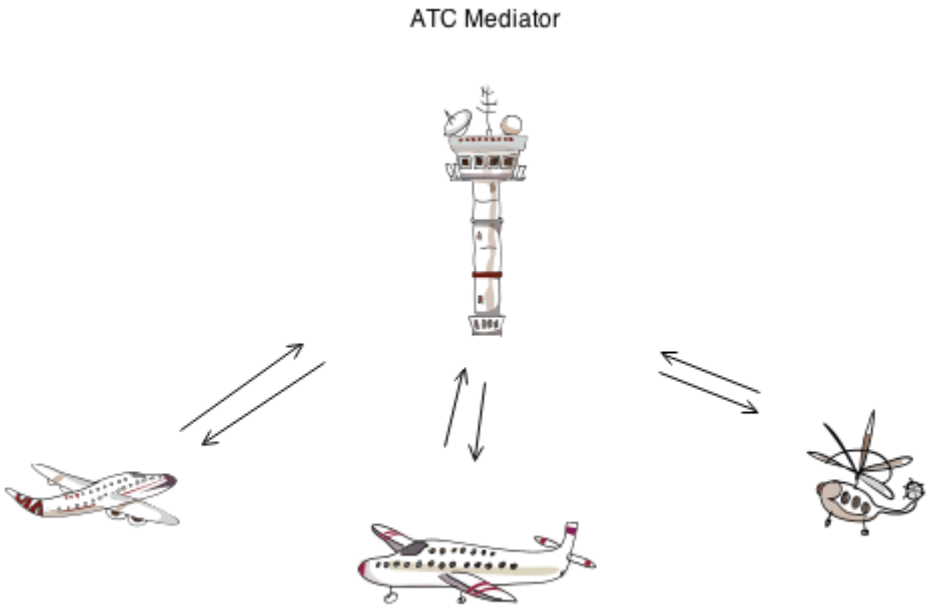


Thành phần

- *Mediator*
  - Định nghĩa một interface để giao tiếp với các đối tượng *Colleague*
- *ConcreteMediator*
  - Thực hiện hành vi hợp tác bằng cách phối hợp các đối tượng *Colleague*
  - Biết và duy trì các *Colleague* của mình
- *Colleague* classes
  - Mỗi lớp *Colleague* biết đối tượng *Mediator* của nó
  - Mỗi *Colleague* liên lạc với *Mediator* của nó bất cứ khi nào nó có thể liên lạc với một *Colleague* khác

Ví dụ

Mediator xác định một đối tượng kiểm soát cách một tập hợp các đối tượng tương tác. Kết nối lỏng lẻo giữa các đối tượng Colleague đạt được bằng cách để các Colleague giao tiếp với Mediator, thay vì với nhau. Tháp điều khiển tại một sân bay được kiểm soát thể hiện rất rõ mô hình này. Các phi công của máy bay tiếp cận hoặc rời khỏi khu vực nhà ga liên lạc với tòa tháp hơn là liên lạc rõ ràng với nhau. Các ràng buộc về người có thể cất cánh hoặc hạ cánh được thi hành bởi tòa tháp. Điều quan trọng cần lưu ý là tòa tháp không kiểm soát toàn bộ chuyến bay. Nó chỉ tồn tại để thực thi các ràng buộc trong khu vực đầu cuối.


ATC Mediator

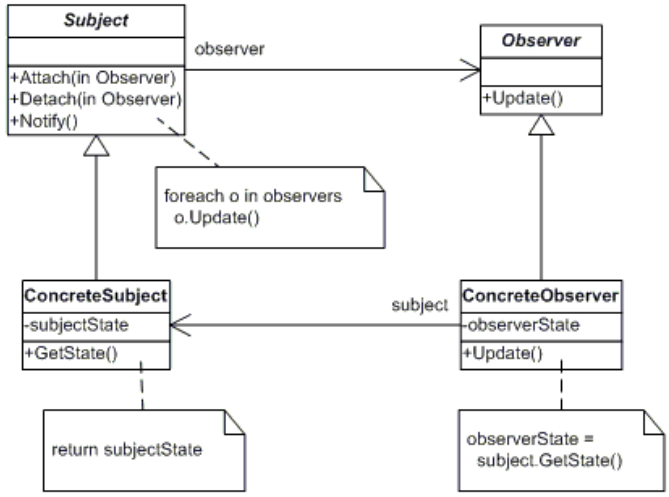
Observer

Loại: Behavioral Pattern

Mẫu thiết kế Observer xác định sự phụ thuộc một-nhiều giữa các đối tượng để khi một đối tượng thay đổi trạng thái, tất cả các đối tượng phụ thuộc của nó sẽ được thông báo và cập nhật tự động.

Tần suất sử dụng: cao

UML



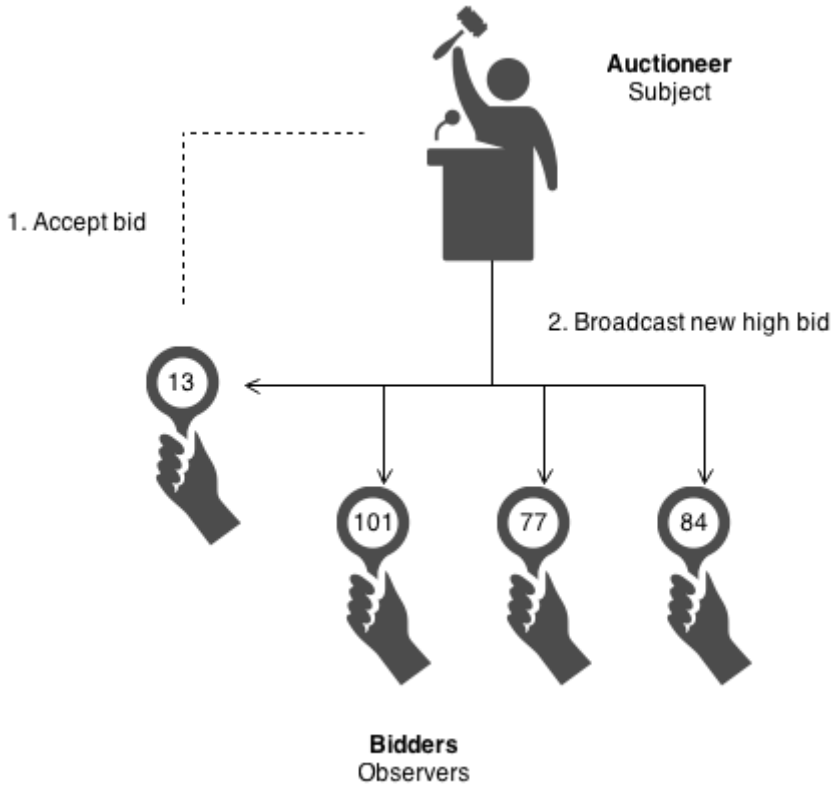
Thành phần

- *Subject*
  - Biết các *Observer* của nó. Bất kỳ số lượng đối tượng *Observer* nào cũng có thể quan sát một *Subject*
  - Cung cấp interface để gắn và tách các đối tượng *Observer*
- *ConcreteSubject*
  - Lưu trữ trạng thái quan tâm tới *ConcreteObserver*
  - Gửi thông báo cho *Observer* của nó khi trạng thái của nó thay đổi
- *Observer*
  - Xác định interface cập nhật cho các đối tượng sẽ được thông báo về những thay đổi trong *Subject*
- *ConcreteObserver*
  - Duy trì một tham chiếu đến một đối tượng *ConcreteSubject*
  - Lưu trữ trạng thái phải phù hợp với *Subject*
  - Thực thi interface cập nhật của *Observer* để giữ cho trạng thái của nó nhất quán với đối tượng

Ví dụ

Observer xác định mối quan hệ một-nhiều để khi một đối tượng thay đổi trạng thái, những đối tượng khác sẽ được thông báo và cập nhật tự động. Một số cuộc đấu giá

chứng minh mô hình này. Mỗi nhà thầu sở hữu một mái chèo được đánh số được sử dụng để biểu thị giá thầu. Người điều hành cuộc đấu giá bắt đầu cuộc đấu giá và "quan sát" khi một mái chèo được nâng lên để chấp nhận cuộc đấu giá. Việc chấp nhận giá thầu làm thay đổi giá dự thầu được phát cho tất cả các nhà thầu dưới hình thức giá thầu mới.



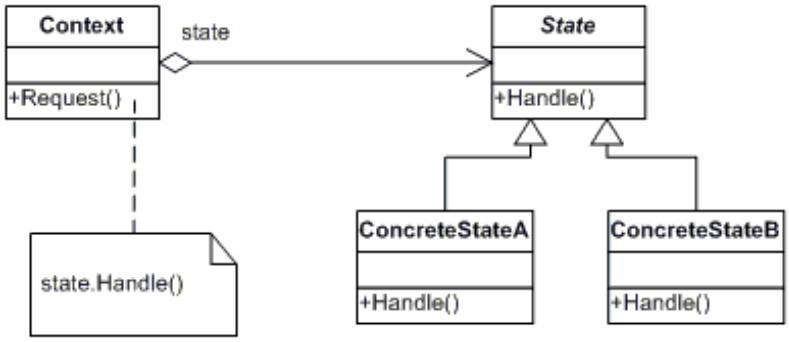
State

Loại: Behavioral Pattern

Mẫu thiết kế State cho phép một đối tượng thay đổi hành vi của nó khi trạng thái bên trong của nó thay đổi. Đối tượng sẽ xuất hiện để thay đổi lớp của nó.

Tần suất sử dụng: trung bình

UML

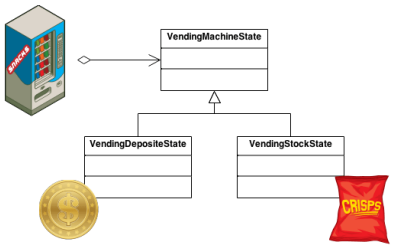


Thành phần

- *Context*
  - Xác định interface quan tâm đến Client
  - Duy trì một thể hiện của lớp con *Concrete State* xác định trạng thái hiện tại
- *State*
  - Định nghĩa một interface để đóng gói hành vi được liên kết với một trạng thái cụ thể của *Context*
- *Concrete State*
  - Mỗi lớp con thực hiện một hành vi được liên kết với trạng thái *Context*

Ví dụ

Mẫu State cho phép một đối tượng thay đổi hành vi của nó khi trạng thái bên trong của nó thay đổi. Mô hình này có thể được quan sát trong một máy bán hàng tự động. Máy bán hàng tự động có các trạng thái dựa trên hàng tồn kho, số lượng tiền gửi, khả năng thay đổi, mặt hàng được chọn, v.v. Khi tiền được gửi và lựa chọn được thực hiện, máy bán hàng tự động sẽ giao sản phẩm và không thay đổi, giao một sản phẩm và thay đổi, không giao sản phẩm do không đủ tiền đặt cọc hoặc không giao sản phẩm do hết hàng tồn kho.



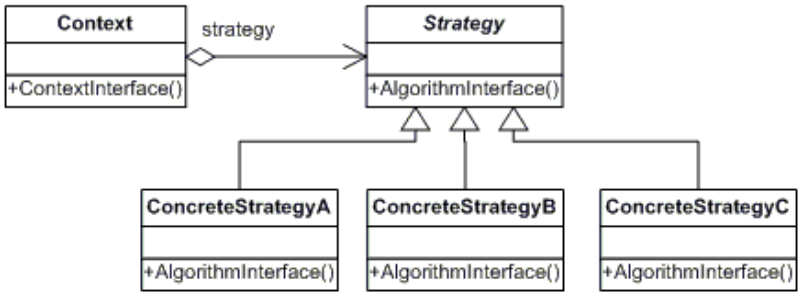
Strategy

Loại: Behavioral Pattern

Mẫu thiết kế Strategy xác định một nhóm các thuật toán, đóng gói từng thuật toán và làm cho chúng có thể hoán đổi cho nhau. Mẫu này cho phép thuật toán thay đổi độc lập với các Client sử dụng nó.

Tần suất sử dụng: cao

UML

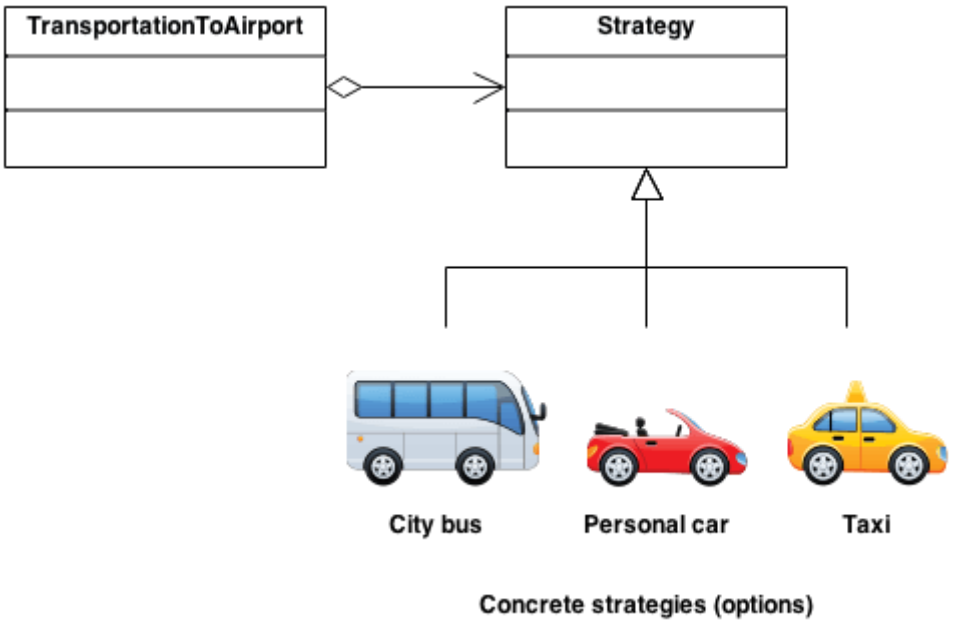


Thành phần

- *Strategy*
  - Khai báo một interface chung cho tất cả các thuật toán được hỗ trợ. Context sử dụng interface này để gọi thuật toán được xác định bởi *ConcreteStrategy*
- *ConcreteStrategy*
  - Thực thi thuật toán bằng interface *Strategy*
- *Context*
  - Được cấu hình với một đối tượng *ConcreteStrategy*
  - Duy trì một tham chiếu đến một đối tượng *Strategy*
  - Có thể xác định một interface cho phép *Strategy* truy cập dữ liệu của nó

Ví dụ

Chiến lược xác định một tập hợp các thuật toán có thể được sử dụng thay thế cho nhau. Phương thức vận chuyển đến sân bay là một ví dụ về Chiến lược. Có một số lựa chọn như lái ô tô của chính mình, đi taxi, xe đưa đón sân bay, xe buýt thành phố hoặc dịch vụ xe limousine. Đối với một số sân bay, tàu điện ngầm và máy bay trực thăng cũng có sẵn như một phương thức vận chuyển đến sân bay. Bất kỳ phương thức vận chuyển nào trong số này sẽ đưa khách du lịch đến sân bay và chúng có thể được sử dụng thay thế cho nhau. Khách du lịch phải chọn Chiến lược dựa trên sự đánh đổi giữa chi phí, sự thuận tiện và thời gian.



Concrete strategies (options)