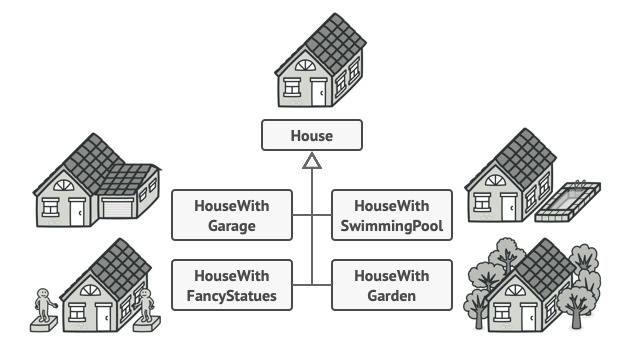
**Builder Pattern**

1. **Định nghĩa:**

Builder pattern thuộc nhóm Creational Pattern, cho phép xây dựng các đối tượng phức tạp theo từng bước.

1. **Đặt vấn đề:**

****

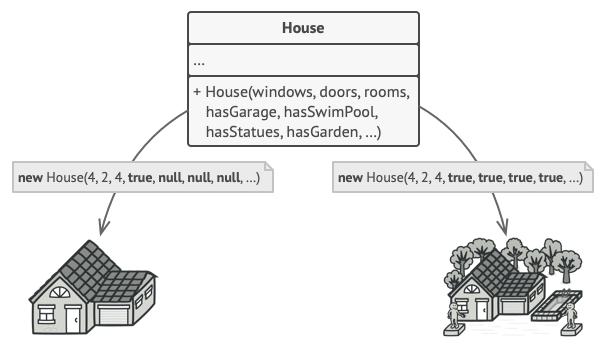
Để xây dựng một ngôi nhà đơn giản, chúng ta cần xây dựng nền móng, 4 bức tường, lắp cửa chính ra vào, lắp cửa số và mái nhà. Nhưng nếu muốn xây dựng một căn nhà lớn hơn với các tiện nghi như hồ bơi, sân vườn,…

- Cách làm 1: Mở rộng lớp cơ sở, tạo một tập hợp các lớp con bao gồm các tham số mở rộng và các tham số chung thừa kế từ lớp cơ sở.

=> Vấn đề: Một số lượng lớn các lớp con được tạo ra, vấn đề được lan rộng ra khi các tham số mở rộng tiếp tục phân lớp (Ví dụ: Các kiểu hồ bơi, các kiểu sân vườn,…)

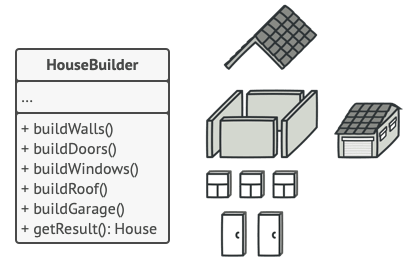
- Cách làm 2: Tạo một hàm tạo không lồ với tất cả các tham số có thể có bên trong lớp cơ sở.

=> Vấn đề: Tham số quá nhiều trong hàm tạo và không phải lúc nào cũng cần các tham số. Trong hầu hết các trường hợp, hầu hết các tham số sẽ không được sử dụng, làm cho hàm tạo phình to và khá xấu về hình thức (Ví dụ: Chỉ một phần nhỏ các ngôi nhà có bể bơi -> các tham số liên quan đến bể bơi sẽ trở nên vô dụng trong các lần tạo các ngôi nhà không có bể bơi).



1. **Giải pháp:**

Mẫu Builder gợi ý rằng ta sẽ trích xuất mã xây dựng đối tượng ra khỏi lớp của chính nó và di chuyển đến các đối tượng riêng biệt được gọi là builder.



Điều này cho phép xây dựng các đối tượng phức tạp theo từng bước, chỉ sử dụng những bước ta thực sự cần.

Mẫu Builder sắp xếp việc xây dựng đối tượng thành một tập hợp các bước (buildWalls, buildDoor, v.v.). Để tạo một đối tượng, ta thực hiện một loạt các bước này trên một đối tượng builder. Phần quan trọng là ta không cần gọi tất cả các bước và chỉ có thể gọi những bước cần thiết để tạo ra một cấu hình cụ thể của một đối tượng.

Một số bước của builder có thể yêu cầu việc thực hiện khác nhau khi ta cần xây dựng các hình thức khác nhau của một đối tượng (Ví dụ: tường của một ngôi nhà bình thường có thể dựng bằng gỗ, nhưng tường của một tòa cao ốc phải được xây bằng gạch, bê tông)

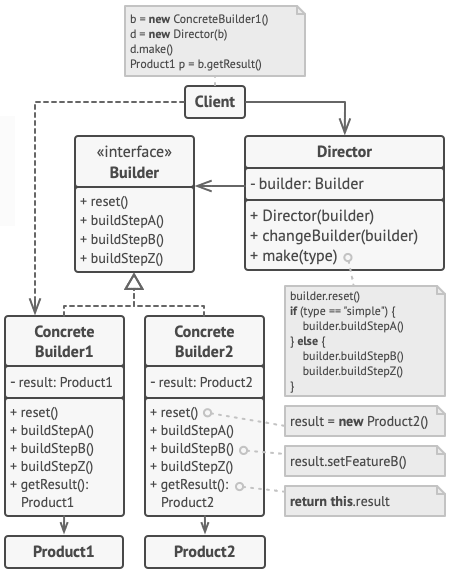
1. **Lợi ích**

Builder Pattern cho phép xây dựng đối tượng (sản phẩm) theo từng bước, có thể trì hoãn việc thực hiện một số bước mà không làm hỏng sản phẩm cuối cùng.

Có thể gọi đệ quy các bước, điều này hữu ích khi cần xây dựng một cây đối tượng.

Builder không để lộ sản phẩm chưa hoàn thành khi đang chạy các bước xây dựng. Điều này ngăn không cho mã máy khách tìm nạp kết quả không đầy đủ.

1. **Cách sử dụng:**

****

**Mô tả:**

**-** Giao diện Builder khai báo các bước xây dựng cho các loại trình xây dựng đối tượng (ConcreteBuilder1, ConcreteBuilder2)

- Các ConcreteBuilder cung cấp các cách triển khai khác nhau của các bước xây dựng.

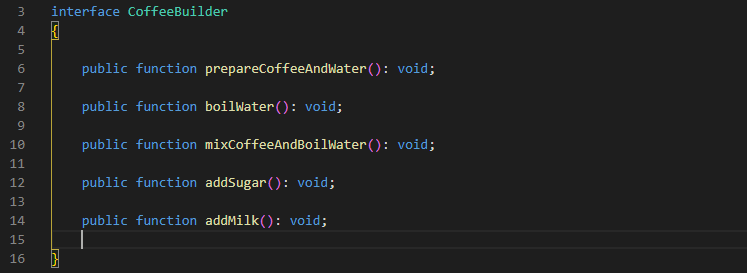
- Procuct là đối tượng kết quả.

- Director là lớp do chúng ta tùy chỉnh, lớp này có trách nhiệm gọi lại các bước xây dựng cho một đối tượng cụ thể, chúng ta có các cách khác nhau để tạo ra một đối tượng sản phẩm trong class này.

- Code của client phải liên kết một trong các đối tượng Builder với Director, vì vậy cả 2 đối tượng Builder và Director. Thông thường điều đó chỉ cần thực hiện một lần (thông qua các tham số của hàm tạo của Director). Sau đó, Director sử dụng đối tượng Builder đ1o cho tất cả các phương thức xây dựng tiếp theo.

**Ví dụ:**

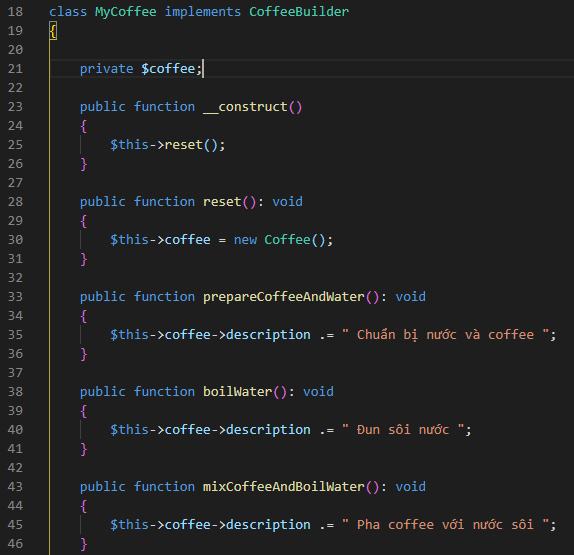
Khai báo giao diện cho việc pha một ly cà phê. Bên trong giao diện khai báo tất cả các phương thức có thể có của quá trình pha cà phê như chuần bị nước và cà phê, đun sôi nước, pha cà phê với nước sôi, thêm đường, thêm sữa.



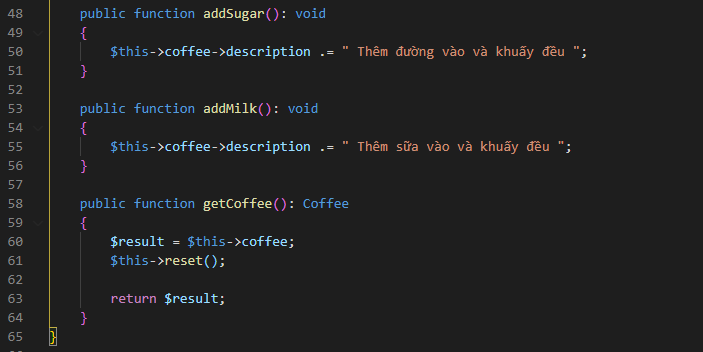
Class MyCoffee (ConcreteBuilder) sẽ thực thi giao diện trên, cung cấp các cách triển khai của các phương thức đã khai báo ở giao diện.

Class MyCoffee chứa một đối tượng $coffee, được sử dụng trong suốt quá trình xây dựng.

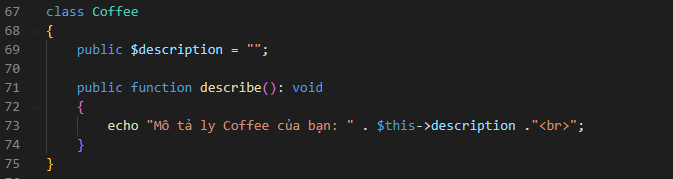
Phương thức reset() làm nhiệm vụ tạo ra một đối tượng mới, phương thức này được Director gọi sau khi đã hoàn thành việc tạo và lấy ra một đối tượng cụ thể, việc này giúp cho việc tạo liên kết giữa Builder và Director chỉ cần thực hiện một lần.



Phương thức getCoffee() trả về kết quả là đối tượng $coffee.

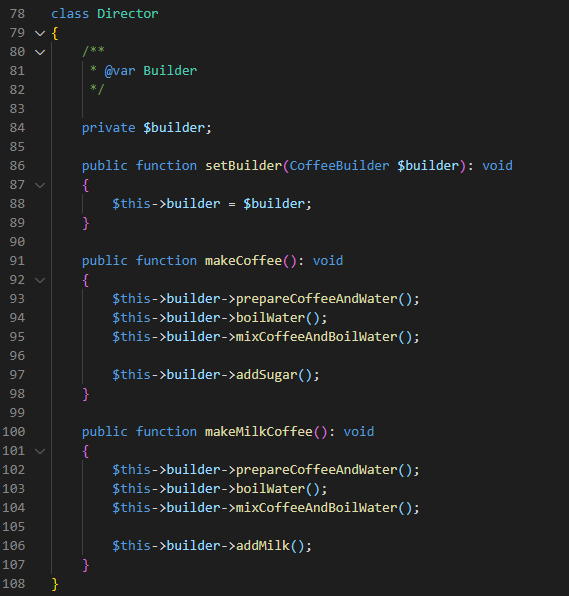


Class Coffee có phương thức in ra mô tả để xem quá trình xây dựng của ly cà phê đó.

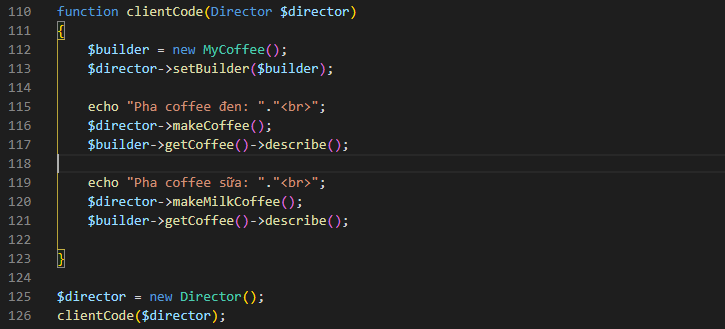


Class Director chứa các phương thức xây dựng các thể hiện khác nhau của đối tượng Builder, với Builder là CoffeeBuilder, ta thử đi xây dựng 2 phương thức một là “pha một ly cà phê bình thường” và hai là “pha một ly cà phê sữa”.

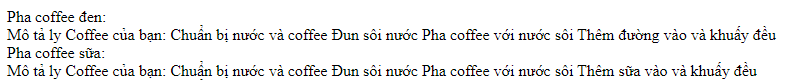
Phương thức makeCoffee() - pha một ly cà phê bình thường ngoài việc gọi lại những phương thức cơ bản như “chuẩn bị cà phê và nước”, “đun sôi nước” và “pha cà phê với nước sôi” thì còn có thêm phương thức addSugar()-thêm đường, với phương thức makeMilkCoffee() - pha một ly cà phê sữa thì ta có gọi thêm phương thức addMilk()-thêm sữa.



Phương thức clientCode(): Khai báo một biến đối tượng thuộc class MyCoffee, tạo liên kết giữa Director và Builder thông qua phương thức setBuilder(). Tạo ra thể hiện cụ thể của đối tượng $builder bằng việc gọi các phương thức xây dựng cụ thể của Director.



**Kết quả:**



1. **Tóm gọn:**

Mẫu Builder Pattern có thể được áp dụng khi việc xây dựng các mô tả khác nhau của một đối tượng sản phẩm bao gồm các bước tương tự chỉ khác nhau về chi tiết.

Builder interface xác định tất cả các bước xây dựng có thể có, các ConcreteBuilder thực thi giao diện Builder interface và đi triển khai các bước này để xây dựng các đại diện cụ thể của đối tượng sản phẩm. Trong khi đó, lớp Director hướng dẫn trình tự của các công việc trong quá trình xây dựng.

1. **Ưu và nhược điểm:**

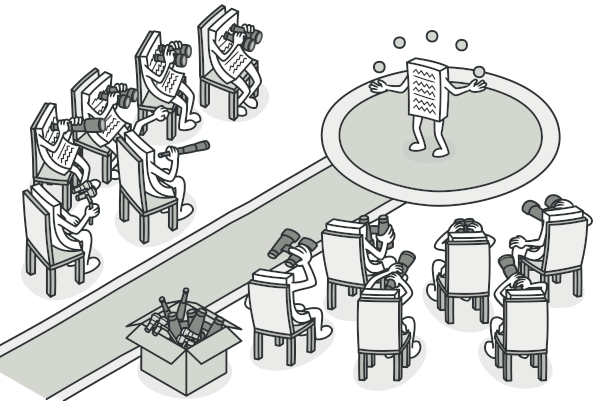
- Ưu: Các lợi ích đã nêu ở trên

- Nhược: Độ phức tạp tăng lên vì mẫu yên cầu tạo thêm nhiều lớp mới.

**Observer Pattern**

1. **Định nghĩa**

Observer là một behavioral pattern, cho phép ta xác định cơ chế đăng ký để thông báo cho nhiều đối tượng về bất kỳ sự kiện nào xảy ra với đối tượng mà họ đang quan sát.



1. **Đặt vấn đề**

Giả sử ta có 2 loại đối tượng khách hàng và cửa hàng, khách hàng rất quan tâm đến một thương hiệu sản phẩm cụ thể (giả sử là một mẫu giày mới, một mẫu điện thoại mới) sẽ sớm có mặt trong cửa hàng. Khách hàng có thể ghé thăm cửa hàng mỗi ngày để xem sản phẩm đã có tại cửa hàng chưa. Nhưng trong khi sản phẩm vẫn đang trên đường nhập về thì hầu hết những chuyến đi này của khách hàng sẽ trở nên vô nghĩa.

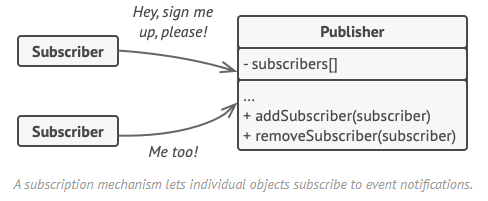
Mặt khác, cửa hàng có thể gửi hàng tá mail ( có thể được coi là thư rác) cho tất cả khách hàng mỗi khi có sản phẩm mới. Điều này tuy giúp một số khách hàng quan tâm thoát khỏi cảnh phải đi đến cửa hàng thường xuyên. Tuy nhiên điều này sẽ làm khó chịu những khách hàng khác không quan tâm đến sản phẩm mới.

**=> Vấn đề:** Khách hàng lãng phí thời gian đi đến cửa hàng để kiểm tra xem sản phẩm đã có mặt tại cửa hàng chưa, còn cửa hàng thì lãng phí nguồn lực để thông báo sai cho khách hàng.

1. **Giải pháp**

Gọi các đối tượng có một số trạng thái nhất định là publisher, tất cả các đối tượng khác muốn theo dõi các thay đổi với trạng thái của publisher được gọi là subcribers.

Mẫu Observer gợi ý rằng, ta nên thêm cơ chế đăng ký vào lớp publisher để các đối tượng riêng lẻ có thể đăng ký hoặc hủy đăng ký khỏi luồng sự kiện đến từ publisher đó.



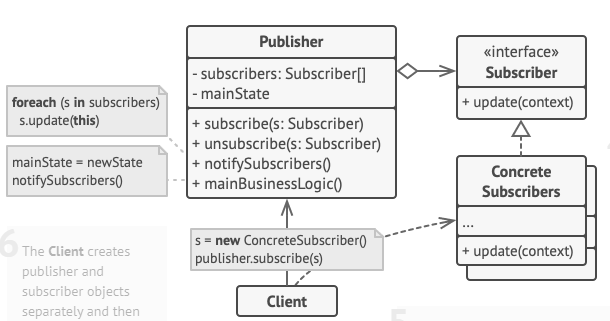
Giờ đây, bất cứ khi nào một sự kiện quan trọng xảy ra với publisher, publisher sẽ xem xét các subcriber của mình và gọi phương thức thông báo cụ thể trên các đối tượng của họ.

1. **Lợi ích**

Có thể thấy nếu ta đăng ký nhận thông báo về tình hình xuất bản của một tờ báo hay tạp chí tại một cửa hàng, ta không cần phải đến cửa hàng để kiểm tra xem số tiếp theo đã ra hay chưa. Thay vào đó, cửa hàng sẽ gửi các số báo mới trực tiếp đến hộp thư của chúng ta ngay sau khi đăng bán tờ báo hay tạp chí mà chúng ta quan tâm.

Đồng thời Cửa hàng cũng duy trì danh sách những người đăng ký và biết họ quan tâm đến tờ báo nào. Người đăng ký có thể rời khỏi danh sách bất kỳ lúc nào khi họ muốn ngăn cửa hàng gửi các số báo hoặc tạp chí mới cho họ.

1. **Cách sử dụng**



**- Publisher:** Đưa ra các sự kiện mà các đối tượng khác quan tâm. Publisher có cơ chế cho người dùng mới đăng ký tham gia và người đã đăng ký trước đó thực hiện hủy đăng ký và rời khỏi danh sách.

**- Subcriber:** Giao diện Subcriber đăng ký một phương thức cập nhật - update() duy nhất (Trong hầu hết các trường hợp). Phương thức này có thể có một số tham số cho phép nhà xuất bản chuyển một số chi tiết sự kiện cùng với bản cập nhật.

**- Concrete Subcriber:** Lớp này thực thi giao diện Subscriber, thực hiện một số hành động để phản hồi lại các thông báo do Publisher đưa ra.

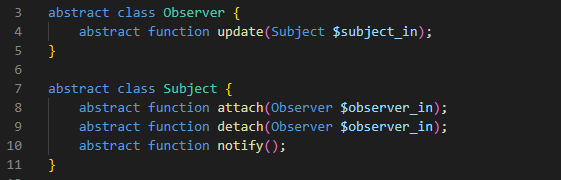
**- Client:** Tạo các đối tượng publisher và subcriber riêng biệt và sau đó đăng ký subcriber cho các bản cập nhật của nhà xuất bản.

**Ví dụ:**

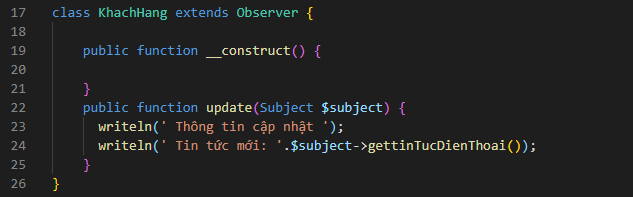
Lấy ví dụ phía trên, một cửa hàng kinh doanh điện thoại sẽ để cho khách hàng đăng ký nhận thông báo về thông tin mẫu điện thoại mới mà họ đang quan tâm.

Sau khi đăng ký, bất cứ thông tin gì về mẫu điện thoại mà cửa hàng thay đổi sẽ đều được cập nhật đến khách hàng. Khi khách hàng hủy đăng ký, họ sẽ không nhận được các thông báo mới nữa.

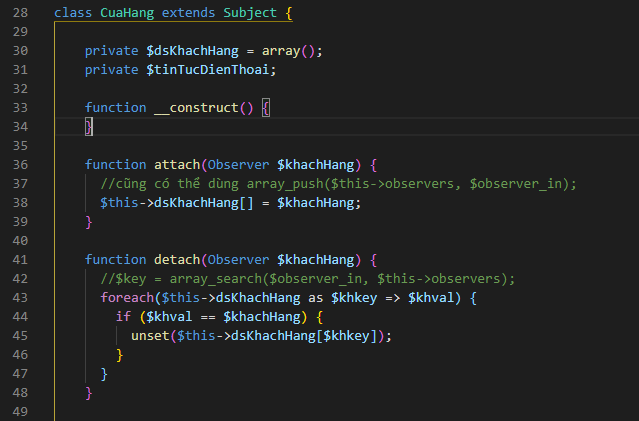
Tạo hai lớp abstract Observer và Subject, như mô tả ở trên thì Observer tương đương với Subcriber, còn Subject tương đương với Publisher. Khai báo các phương thức cần thiết cho 2 abstract class trên.



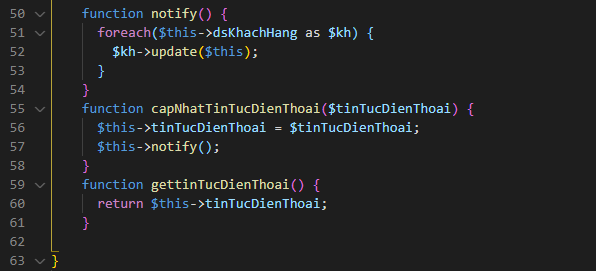
Class KhachHang extends từ Observer, thực hiện việc triển khai cụ thể cho phương thức update().



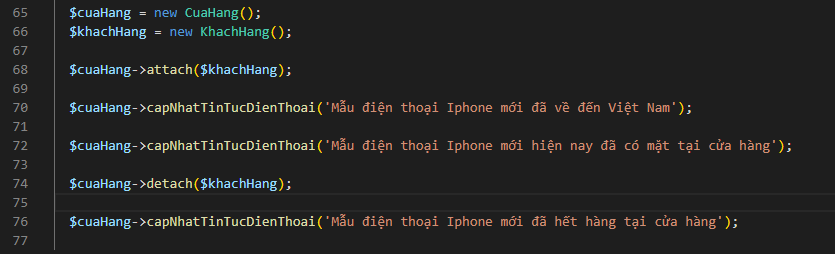
Class CuaHang extends từ Subject, khai báo môt danh sách các khách hàng, hai phương thức attach() và detach() sẽ thao tác với danh sách khách hàng này.



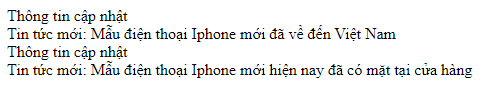
Phương thức notify sẽ thực hiện việc thông báo đến cho từng khách hàng trong danh sách, ở đây mỗi khách hàng sẽ thực hiện phương thức update của chính nó.



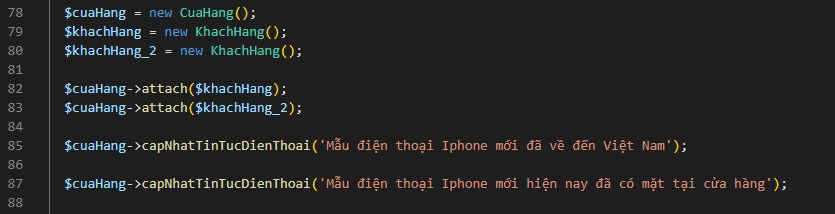
Tạo 2 biến riêng biệt $cuaHang và $khachHang là 2 đối tượng cụ thể của 2 class CuaHang va KhachHang, thực hiện việc đăng ký thông qua phương thức attach(), sau khi đăng ký thì bất cứ khi nào Cửa hàng cập nhật tin tức về mẫu điện thoại mới thì khách hàng đăng ký sẽ nhận được thông báo, ngược lại khi hủy đăng ký thông qua phương thức detach() thì khách hàng sẽ không còn nhận được các thông báo từ cửa hàng nữa.



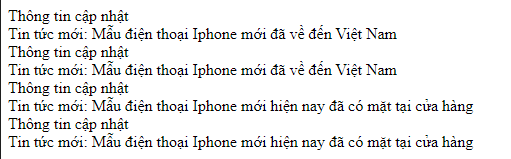
Kết quả



Khi có 2 khách hàng thực hiện đăng ký



Kết quả:



1. **Tóm gọn**

Sử dụng mẫu observer khi một số đối tượng trong ứng dụng của bạn phải quan sát những đối tượng khác. Danh sách đăng ký là dynamic, vì vậy người đăng ký có thể tham gia hoặc rời khỏi danh sách bất cứ khi nào họ cần.

1. **Ưu và nhược điểm**

**Composite Pattern**

1. **Định nghĩa:**

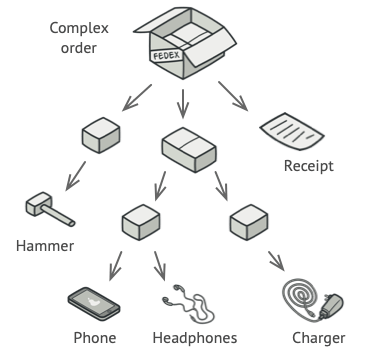
Composite là một structural pattern, cho phép sắp xếp các đối tượng thành cấu trúc cây và sau đó làm việc với các cấu trúc này như thể chúng là các đối tượng riêng lẻ.

1. **Đặt vấn đề:**

Thử tưởng tượng ta có 2 loại sản phẩm và hộp đựng, một hộp đựng có thể chứa một số sản phẩm cũng như chứa được một số hộp nhỏ hơn, những hộp nhỏ này cũng có thể chứa một số sản phẩm hoặc thậm chí một số hộp nhỏ hơn nữa...

Giả sử ta tạo một hệ thống đặt hàng sử dụng những lớp này. Đơn đặt hàng có thể chứa các sản phẩm đơn giản, cũng như các hộp chứa sản phẩm khác,...

=>**Vấn đề:** Làm sao xác định được tổng giá của một đơn đặt hàng như vậy?



(Một đơn hàng có thể bao gồm nhiều sản phẩm khác nhau, đucợ đóng gói trong hộp, được đóng gói trong hộp lớn hơn,...Toàn bộ cấu trúc trông giống như một cái cây lộn ngược)

(Bạn có thể thử cách tiếp cận trực tiếp: mở tất cả các hộp, xem qua tất cả các sản phẩm và sau đó tính tổng. Điều đó có thể làm được trong thế giới thực, nhưng trong một chương trình, nó không đơn giản như chạy một vòng lặp. Bạn phải biết trước các loại Sản phẩm và Hộp mà bạn đang trải qua, cấp độ lồng vào nhau của các hộp và các chi tiết khó chịu khác. Tất cả những điều này làm cho cách tiếp cận trực tiếp trở nên quá khó khăn hoặc thậm chí là không thể.)

1. **Giải pháp:**

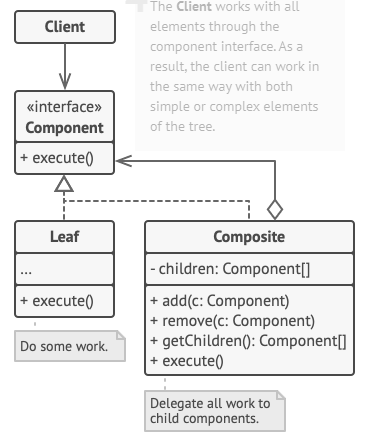
Mẫu Composite đề xuất rằng ta nên làm việc với Sản phẩm và Hộp thông qua một giao diện interface chung khai báo phương thức tính tổng giá.

Cách hoạt động của phương pháp này: Đối với một sản phẩm, nó chỉ cần trả lại giá của sản phẩm đó. Đối với một hộp thì nó sẽ xem xét từng mục trong hộp, hỏi giá của nó và sau đó trả lại tổng số cho hộp này. Nếu một trong những mặt hàng này là một hộp nhỏ hơn, hộp đó cũng sẽ bắt đầu xem xét nội dung của nó cho đến khi tính được giá của tất cả các thành phần bên trong. (Một hộp có thể thêm một số chi phí bổ sung vào giá cuối cùng, chẳng hạn như chi phí đóng gói)

1. **Lợi ích:**

Lợi ích lớn nhất của phương pháp này là bạn không cần quan tâm đến các lớp cụ thể của các đối tượng tạo nên cây. Bạn không cần biết một vật là một sản phẩm đơn giản hay một chiếc hộp phức tạp. Ta có thể xử lý tất cả chúng như nhau thông qua giao diện chung. Khi ta gọi một phương thức, các đối tượng tự truyền yêu cầu xuống cây.

1. **Cách sử dụng:**



- **Component:** Giao diện Component mô tả các hoạt động phổ biến cho các phần tử đơn giản và phức tạp của cây.

- **Leaf:** Leaf (lá) là thành phần cơ bản nhất của cây, không có phân tử con hay phần tử phụ, thông thường các phần tử leaf thực hiện các công việc thực sự, vì chúng không có ai để ủy quyền công việc.

- **Conposite:** Là một phần tử có các phần tử phụ: leaf hoặc các thùng chứa khác. Một thùng chứa không biết các lớp cụ thể của các lớp con của nó. Nó chỉ hoạt động với tất cả phần tử con thông qua giao diện Component.

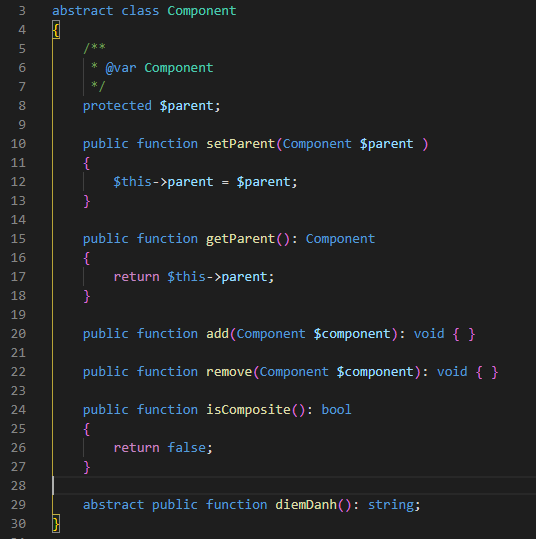
Khi nhận được yêu cầu, một composite ủy quyền công việc cho các phần tử con của nó, xử lý kết quả trung gian và sau đó trả lại về kết quả cuối cùng cho client.

- **Client:** Làm việc với tất cả các phần tử thông qua giao diện Component. Do đó client có thể làm việc theo cùng một cách với cả các phần tử đơn giản hoặc phức tạp của cây.

**Ví dụ:** Điểm danh trong quân đội là một việc thực hiện hàng ngày mà ta có thể thấy, giả sử trong một Đại đội, người chỉ huy có trách nhiệm điểm danh quân số không thể nào đi xuống từng Trung đội, từng Tiểu đội để đếm từng người lính, với những bộ phận quân đội có quy mô lớn hơn thì việc này đòi hỏi nhiều công sức.

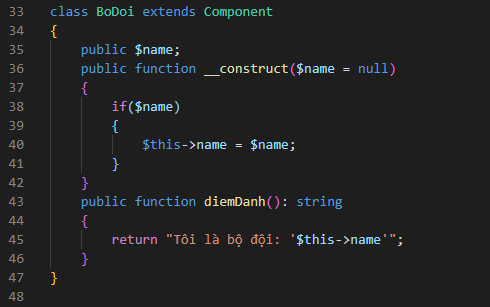
Với cách xử lý của mẫu Composite, việc này có thể thực hiện theo cách, mỗi người lính (cấp thấp nhất) trong một tiểu đội sẽ điểm danh với tiểu đội trưởng, các tiểu đội trưởng trong một trung đội sẽ điểm danh lại với trung đội trưởng, và các trung đội trưởng sẽ điểm danh quân số lại cho đại đội trưởng.

Lớp abtract Component khai báo các phương thức hoạt động của các phần tử cơ bản.



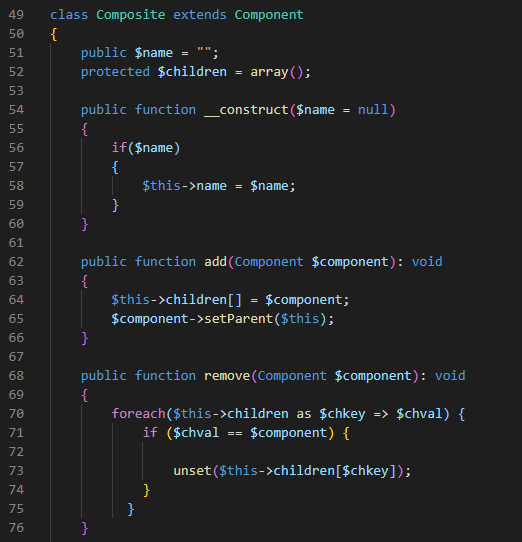
Lớp Bộ đội extends từ Component và đi triện khai cụ thể phương thức diemDanh(), phương thức này đơn giản in ra tên của người lính bộ đội.

Vì là Class đối tượng cơ bản (leaf) nên Class Bộ Đội sẽ không có phương thức add() và remove().

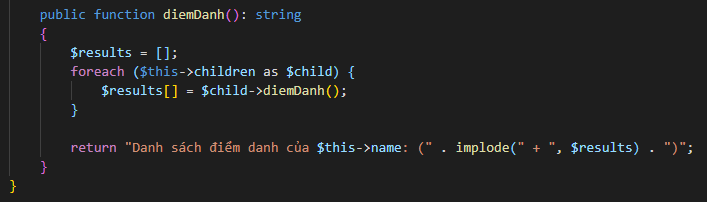


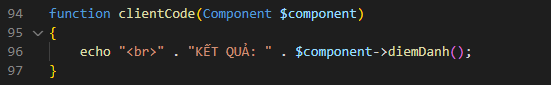
Class Composite extends lại Component và đi triển khai cụ thể cho các phương thức, biến $children là một mảng chứa các đối tượng con, phương thức add() và remove() sẽ thao tác với mảng này, tham số của chúng là một thể hiện cụ thể của Component.

(Composite hiểu đơn giản ở đây có thể là một người lính bộ đội, hoặc có thể là một tiểu đội, một trung đội...)



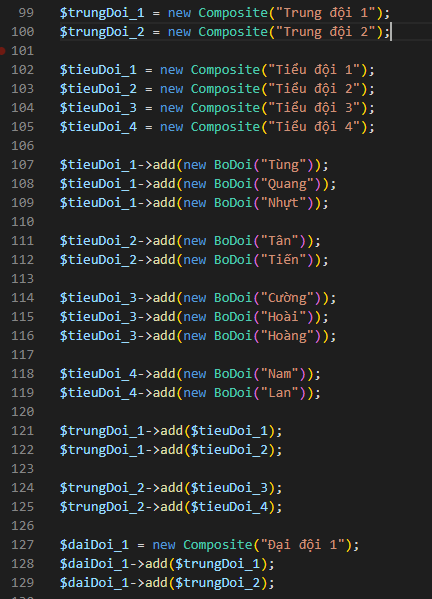
Phương thức điểm danh sẽ gọi lại phương thức điểm danh của các thành phần con.

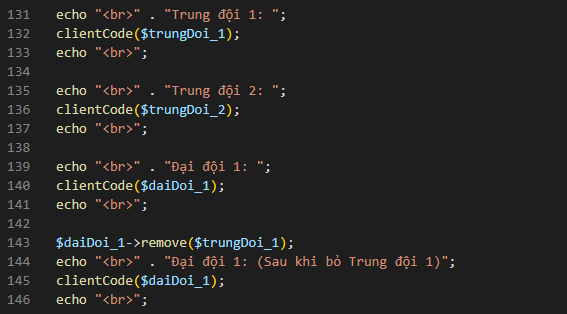




Chạy code:

Thực hiện các khai báo, add các composite để tạo một cây đối tượng.





Kết quả:

