

❖ Adapter pattern

❖ Bridge pattern

❖ Composite pattern

❖ Decorator pattern

❖ Façade pattern

❖ Proxy pattern

3

Example

Lemonade



Lemonade



5

Decorator pattern

2

Example: Cost of Beverage

Beverage

description

getDescription()

cost()

// Other useful methods...

HouseBlend

cost()

DarkRoast

cost()

Decaf

cost()

Espresso

cost()

The description instance variable is set in each subclass and holds a description of the beverage, like "Most Excellent Dark Roast". The getDescription() method returns the description.

The cost() method is abstract; subclasses abstractly subclasses need to define their own implementation.

Beverage is an abstract class subclassed by all beverages offered in the coffee shop.

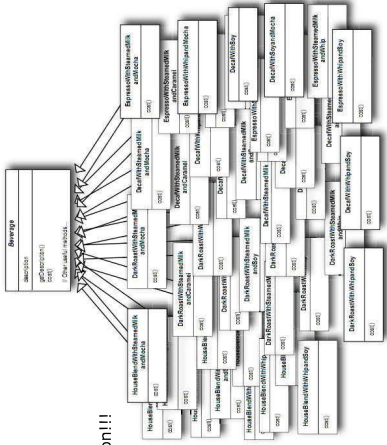
Thêm một số thành phần phụ vào 4 loại nước đã có ?

Each subclass implements cost() to return the cost of the beverage.

6


❖ 1st idea

Class explosion!!!



7

❖ The Open-Closed Principle:



Design Principle
Classes should be open for extension, but closed for modification.

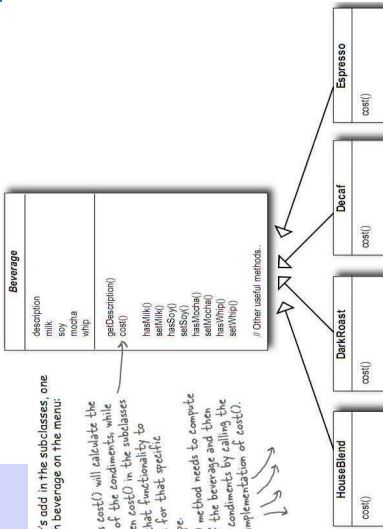
9

❖ 2nd idea

Now let's add in the subclasses, one for each beverage on the menu:

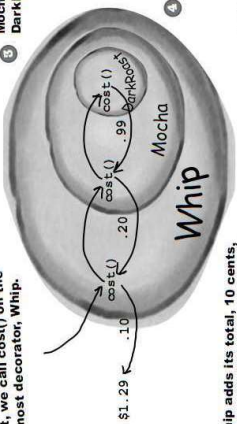
The superclass `cost()` will calculate the costs for all of the condiments, while the overridden `cost()` in the subclasses will extend that functionality to include costs for that specific beverage type.

Each `cost()` method needs to compute the cost of the beverage and then add in the condiments by calling the superclass implementation of `cost()`.



8

❖ 3rd idea



1 First, we call `cost()` on the outmost decorator, Whip.

2 Whip calls `cost()` on Mocha.

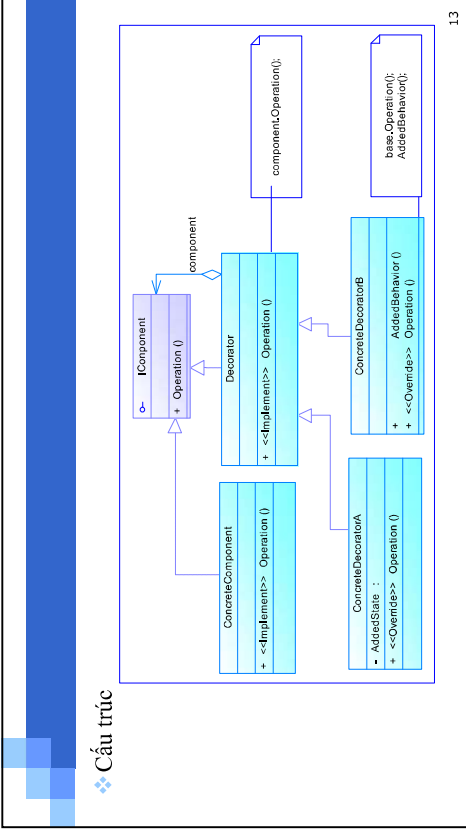
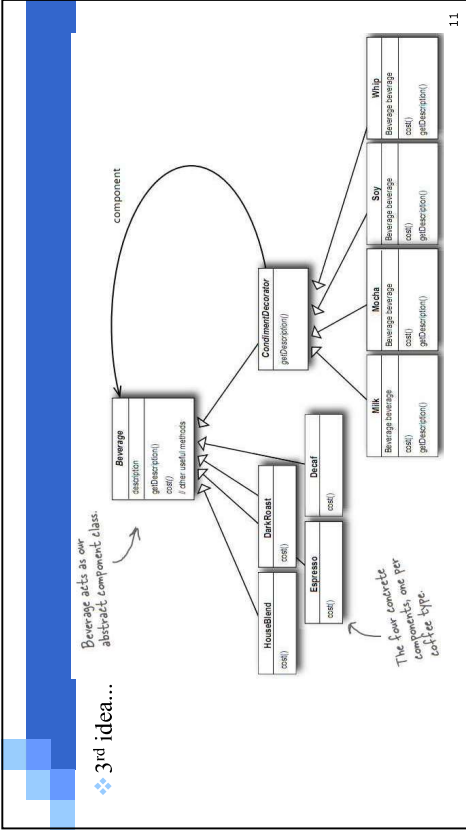
3 Mocha calls `cost()` on DarkRoast.

4 DarkRoast returns its cost, 99 cents.

5 Mocha adds its cost, 20 cents, to the result from DarkRoast, and returns the new total, \$1.19.

6 Whip adds its total, 10 cents, to the result from Mocha, and returns the final result—\$1.29.

10



Decorator pattern

❖ Mục đích:

- Cho phép thêm mới các trạng thái và hành vi vào một đối tượng lúc run-time bằng cách dùng kỹ thuật subclassing để mở rộng các chức năng của lớp.

12

Questions

- Vai trò của lớp Decorator, có thể không cần dùng lớp Decorator được không?
- Nếu mối liên hệ giữa ConcreteComponent và ConcreteDecorator
- Trường hợp sử dụng của mẫu Decorator
 - Client sử dụng một component theo một interface không thay đổi nhưng muốn sử dụng các phiên bản mở rộng của component, nhưng:
 - Không thể mở rộng thành phần component bằng cách thừa kế, hoặc;
 - Việc mở rộng thành phần component có thể dẫn đến việc bùng nổ lớp

14

03/09/2023

Behavioral patterns

- ❖ Chain of Responsibility.
- ❖ Command pattern
- ❖ Interpreter pattern
- ❖ Iterator pattern
- ❖ Mediator pattern
- ❖ Memento pattern
- ❖ Observer pattern
- ❖ State pattern
- ❖ Strategy pattern
- ❖ Template Method
- ❖ Visitor pattern

3

03/09/2023

observer pattern

- ❖ Mục đích: Định nghĩa một phụ thuộc one-to-many giữa các đối tượng sao cho khi một đối tượng thay đổi trạng thái, tất cả các đối tượng phụ thuộc nó được thông báo và được cập nhật một cách tự động.


5

Observer pattern

03/09/2023

Các ví dụ về observer pattern

- ❖ Chương trình bảng tính Excel
- ❖ Data binding
- ❖ Đặt mua báo dài hạn
 - Đọc giả đăng ký mua báo dài hạn với tòa soạn
 - Khi có một tờ báo mới xuất bản thì nó được đại lý phân phối đến đọc giả.
 - Publishers + Subscribers = Observer Pattern



6

03/09/2023

Five minute drama: a subject for observation

In today's act, two part-bubbles software developers encounter a real live head hunter...

Software Developer #1

1 This is
2 Ben, I'm looking for a
3 job. I've got five years
4 experience and I'm
5 ready to start. I'll
6 call you!

Headhunter (Subject)

7 Uh, yeah,
8 you are my list at 3:00
9 PM. I'll call you
10 when I find a job for
11 you.

Software Developer #2

12 I'll call you when I
13 find a job for you.
14 I'll call you when I
15 find a job for you.

Headhunter (Subject)

16 I'll call you when I
17 find a job for you.
18 I'll call you when I
19 find a job for you.

Software Developer #1

20 I'll call you when I
21 find a job for you.
22 I'll call you when I
23 find a job for you.

Software Developer #2

24 I'll call you when I
25 find a job for you.
26 I'll call you when I
27 find a job for you.

7

03/09/2023

Software Developer #1

1 This is
2 Ben, I'm looking for a
3 job. I've got five years
4 experience and I'm
5 ready to start. I'll
6 call you!

Headhunter (Subject)

7 Uh, yeah,
8 you are my list at 3:00
9 PM. I'll call you
10 when I find a job for
11 you.

Software Developer #2

12 I'll call you when I
13 find a job for you.
14 I'll call you when I
15 find a job for you.

Headhunter (Subject)

16 I'll call you when I
17 find a job for you.
18 I'll call you when I
19 find a job for you.

Software Developer #1

20 I'll call you when I
21 find a job for you.
22 I'll call you when I
23 find a job for you.

Software Developer #2

24 I'll call you when I
25 find a job for you.
26 I'll call you when I
27 find a job for you.

8

03/09/2023

Observer pattern

Cấu trúc

```
classDiagram
    class Subject {
        <<abstract>>
        +Attach(Observer observer)
        +Detach(Observer observer)
        +Notify()
    }
    class ConcreteSubject {
        -subjectState
        +GetState()
    }
    class Observer {
        +Update()
    }
    class ConcreteObserver {
        -ObserverState
        +<<Implement>> Update()
    }
    Subject <|-- ConcreteSubject
    Observer <|-- ConcreteObserver
    Subject "1" -- "*" Observer : observers
    ConcreteSubject --> ConcreteObserver : subject
    ConcreteSubject --> ConcreteObserver : observerState=subject.getState();
```

9

03/09/2023

Questions

- Giải thích vai trò của hàm Update() từ đó biết vai trò của IObservable
- Có thể thay thế Observer pattern bằng mô hình các đối tượng dùng chung dữ liệu được không?
- Loosely coupled design: Hãy phân tích mối quan hệ giữa 2 ConcreteSubject và ConcreteObserver
- Viết mã lệnh cho cấu trúc của Observer pattern

10

5

03/09/2023

Trường hợp sử dụng của Observer Pattern

- ❖ Sự thay đổi dữ liệu của một đối tượng có thể kéo theo sự cập nhật trạng thái của các đối tượng khác (Ví dụ: thay đổi dữ liệu kéo theo việc cập nhật giao diện tự động để hiển thị dữ liệu).
- ❖ Sự thay đổi một đối tượng kéo theo một hiệu ứng phụ nào đó tùy thuộc vào thành phần client sử dụng.
- ❖ Viết/thiết kế các callback trong các library/framework.

11

03/09/2023

Bài tập

- ❖ Tìm hiểu các mô hình lập trình, framework, library có sử dụng Observer Pattern

12

03/09/2023

Trường hợp sử dụng

43

Chain of Responsibility

03/09/2023

Chain of Responsibility

❖ Mục đích:

- Tách rời thành phần gọi và đối tượng thực hiện request

❖ Ý tưởng:

- Kết nối các đối tượng thực hiện request thành một chuỗi
- Chuyển request dọc theo chuỗi cho đến khi gặp được đối tượng có khả năng xử lý được nó

45

03/09/2023

Question

❖ Có thể thay đổi Chain of Responsibility bằng cách dùng cấu trúc lệnh if...else của các ngôn ngữ lập trình được không?

47

03/09/2023

Chain of Responsibility

❖ Cấu trúc:

```
classDiagram
    class Client
    class Handler {
        +HandleRequest()
        +SetSuccessor(Handler successor)
    }
    class ConcreteHandler1 {
        +HandleRequest()
        +SetSuccessor(Handler successor)
    }
    class ConcreteHandler2 {
        +HandleRequest()
        +SetSuccessor(Handler successor)
    }
    Client --> Handler
    Handler <|-- ConcreteHandler1
    Handler <|-- ConcreteHandler2
    ConcreteHandler1 --> ConcreteHandler2 : SetSuccessor
    ConcreteHandler2 --> ConcreteHandler2 : SetSuccessor (this.successor=null;)
```

46

03/09/2023

Sử dụng

❖ “Người gọi” không biết phải gọi request cho “người nhận” nào trong tập các “người nhận”

❖ Chuỗi các “người nhận” có thể thay đổi lúc run-time

48

Iterator pattern

03/09/2023

Iterator

03/09/2023

Vấn đề:

- Các tập hợp khác nhau được biểu diễn theo các cách khác nhau
- Client truy cập tới các phần tử của tập hợp theo một cách duy nhất
- Client không cần biết cấu trúc của từng tập hợp cụ thể

51

Iterator pattern

03/09/2023

They want to use my Pancake House menu as the breakfast menu and the Diner's menu as the lunch menu. We've agreed on an implementation for the menu items...

Low

Me!

...but we can't agree on how to implement our menus. That joker over there used an ArrayList to hold his menu items, and I used an Array. Neither one of us is willing to change our implementations... we just have too much code written that depends on them.

50

Iterator

03/09/2023

Mục đích:

- Cung cấp một cách truy cập các phần tử của một tập hợp một cách tuần tự mà không cần biết cấu trúc của tập hợp đó.

Cấu trúc

52


03/09/2023

Questions

- ❖ Có thể gộp chung hai lớp Iterator và Aggregate được không? Nếu ru và khuyết điểm của cách làm này.
- ❖ Vận dụng mẫu Iterator: Mẫu Iterator đã được cài đặt cho hầu hết các tập hợp, việc sử dụng mẫu Iterator chính là sử dụng các cái đặt sẵn có của iterator trên các tập hợp này. Sau đây là một số khuyến nghị:
 - Iterator là giao diện chung cho tất cả các tập hợp cài đặt một phương pháp duyệt tập hợp thứ hai. Do đó, để tránh trường hợp client phải phụ thuộc vào một tập hợp cụ thể, nên sử dụng tập hợp qua giao diện Iterator nếu có thể.

53

03/09/2023



Design Principle

A class should have only one reason to change.

Mỗi class chỉ nên thiết kế với một single responsibility

54

Mẫu decorator

- Ý nghĩa là mẫu có 1 lớp cha là lớp trừu tượng/ giao diện
- 2 lớp con kế thừa bao gồm :
 - 1 lớp đại diện yêu cầu cần thực hiện
 - 1 lớp là lớp yêu cầu. Lớp này là lớp trừu tượng. Lớp có các lớp con và các lớp con là các yêu cầu của bài toán
- Ví dụ Lớp 1 chiếc pc bao gồm: Ram, CPU, Màn hình
 - Phân tích--
 - Lớp cha là lớp PC và gồm 2 lớp con
 - Lớp 1: là lớp Concrete: thực hiện yêu cầu lắp
 - Lớp 2: là lớp Decorater: là lớp yêu cầu gồm 3 lớp con bao gồm: RAM,CPU và màn hình

Mẫu Chain: là mẫu chuyển các công việc cho lớp có thể xử lý công việc cho lớp có thể xử lý được yêu cầu đó.

Mẫu gồm 3 lớp:

- Lớp Cha – Lớp đại diện cho 2 lớp con và chứa các phương thức thực hiện yêu cầu của bài, 1 phương thức là Lớp Cha ketiep(LopCha v); Chịu trách nhiệm chuyển lên đối tượng có thể thực hiện yêu cầu.

- Lớp con 1: là Lớp dùng để tạo các đối tượng cấp dưới và kế thừa các phương thức của lớp Cha.

Có 1 biến là Lớp cha: phụ trách chuyển đối tượng lên đối tượng cấp cao hơn có thể giải quyết yêu cầu.

- Lớp con 2: là Lớp dùng để tạo duy nhất 1 đối tượng là cấp cao nhất, kế thừa các phương thức của lớp Cha.

Mẫu Observer

- **Subject:** là abstract/interface chịu trách nhiệm quản lý và thông báo đến các observer nào đã đăng ký khi có sự thay đổi.

Bao gồm 3 phương thức: Attach(), Detach(),Notify() và chứa 1 biến là đại diện observer(**Lớp Observer**).

Subject còn có các lớp con kế thừa là ConcreteSubject.

- **Observer:** là: là abstract/interface chịu trách nhiệm cập nhập thông tin thay đổi cho các **ConcreteObserver** và có phương thức Update()

- ConcreteObserver: sẽ kế thừa từ lớp Observer và có 1 biến là (Lớp **ConCreateSubject**) và có các phương thức khác như (Đăng ký, Hủy đăng ký).

Mẫu Iterator

Định nghĩa mẫu Iterator sẽ chịu trách nhiệm duyệt các phần tử trong tập hợp.

Tập có thể là 1 List, Mảng hoặc 1 tập hợp do người dùng quy định

Mẫu iterator được java hỗ trợ với class có tên là iterator bao gồm 2 phương thức chính:

hasNext() và Next

hasNext(): Chịu trách nhiệm duyệt các phần tử có trong mảng

Next(): Chịu trách nhiệm lấy giá trị của các phần tử đã được duyệt

Các Phương pháp làm:

- Phương pháp 1: Duyệt 1 mảng có sẵn

- Phương pháp 2: Duyệt 1 tập có sẵn.

- Phương pháp 3: Duyệt 1 tập hợp do người dùng định nghĩa. Cách này khó vì phải người dùng phải tự

các iterator.

- Phương pháp 1: Làm trong hàm main

- Tạo 1 List<Kiểu dữ liệu> = khởi tạo mảng mà add phần tử vào

- Tạo iterator trùng với kiểu dữ liệu mảng và add iterator cho list

- Thiết lập điều kiện dừng cho iterator và sử lý yêu cầu bên trong điều kiện dừng

- Phương pháp 2:

- Sau khi tạo xong hàm cấu trúc cho đối tượng thì chúng ta bắt đầu tạo 1 hàm để duyệt phần tử

- Hàm này phải là static và tham số truyền vào sẽ là 1 Iterator<Lớp cấu trúc>

- Trong điều kiện duyệt phần tử sẽ phải tạo 1 đối tượng của Lớp cấu trúc

và cho Iterator lấy dữ liệu trong đối tượng.

- Sau khi lấy được dữ liệu sẽ bắt đầu sử lý yêu cầu

- Sau đó sẽ viết hàm main giống với cách 1 nhưng chỉ khác là sẽ dùng hàm đã tạo để duyệt phần tử.

Tài liệu UML:

I Mối quan hệ giữa các lớp

1. Kế thừa:

- Trong UML Class Diagram, mối quan hệ giữa các lớp được thể hiện bằng các mũi tên, và quan hệ kế thừa được thể hiện bằng **mũi tên đầu tam giác rỗng nét liền, với đầu mũi tên chỉ về phía lớp cha.**

Ký hiệu: Lớp con <|-- Lớp kế thừa

Ảnh minh họa:



2. Triển khai và phụ thuộc

- **Triển khai:** Mối quan hệ triển khai giữa một lớp và một giao diện được thể hiện bằng **một mũi tên đầu tam giác rỗng nét đứt.**

Ký hiệu: Lớp triển khai(Con) --> Lớp bị triển khai(Cha): **1 biến gì đó??**

Ảnh minh



Chú ý: khi sử dụng --> thì sẽ có thêm 1 biến ở ngay trên mũi tên

- **Phụ thuộc:** những mối quan hệ phụ thuộc khác giữa hai lớp đối tượng sẽ được thể hiện bằng **mũi tên mở nét liền.(Hoặc chứa các thành phần của lớp bị phụ thuộc)**

Ký hiệu: Lớp phụ thuộc(Con) ..|> Lớp bị phụ thuộc (Cha)

Ảnh minh



II. Các quan hệ giữa hai đối tượng

- **sự liên hợp (association):**

Khi một đối tượng này tương tác với đối tượng khác, chẳng hạn đọc thuộc tính, gọi phương thức ..., quan hệ giữa chúng được gọi là sự liên hợp (association), và **được thể hiện bằng mũi tên rỗng nét liền.**

Ký hiệu: -->: **1 biến gì đó??**

Ảnh minh



Chú ý: khi sử dụng --> thì sẽ có thêm 1 biến ở ngay trên mũi tên

- sự phụ thuộc (dependency)

Khi một đối tượng này có biết đến đối tượng kia, chẳng hạn qua tham số đầu vào, nhưng không tạo ra tương tác, quan hệ giữa chúng được gọi là sự phụ thuộc (dependency), và được thể hiện bằng **mũi tên mờ nét đứt**.

Ký hiệu: ..->

Ảnh minh  họa:

- sự tổ hợp (composition)

Khi một đối tượng là một thành phần để cấu thành nên một đối tượng khác, và nó chỉ có thể tồn tại như là một thành phần của đối tượng cha, thì quan hệ giữa chúng được gọi là sự tổ hợp (composition) giữa hai đối tượng, và được thể hiện bằng **mũi tên liền hợp đi kèm với một hình quả trám đặc làm gốc**.

- sự tụ tập (aggregation)

Khi một đối tượng là một thành phần cấu thành nên một đối tượng khác, nhưng vẫn có thể tồn tại mà không cần đến sự tồn tại của đối tượng cha, quan hệ giữa chúng được gọi là tụ tập (aggregation) và được thể hiện bằng **mũi tên mờ nét liền với một hình quả trám rỗng làm gốc**.

Ký hiệu: o-->: **listerner**

Ảnh minh  họa:

chú ý : Khi sử dụng o--> sẽ có thêm 1 biến nữa để lắng nghe: **listerner**