



# *Structural pattern*

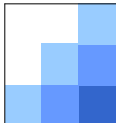
**Giảng viên: Huỳnh Tuấn Anh**  
**Khoa CNTT - Đại học Nha Trang**



## **Structural pattern**

- ❖ Liên quan tới cách tổ chức các đối tượng để hình thành các cấu trúc lớn hơn
- ❖ Mô tả cách để kết hợp các đối tượng để thực hiện một chức năng mới

2

- 
- ❖ Adapter pattern
  - ❖ Bridge pattern
  - ❖ Composite pattern
  - ❖ Decorator pattern
  - ❖ Façade pattern
  - ❖ Proxy pattern

3



# *Decorator pattern*

## Example

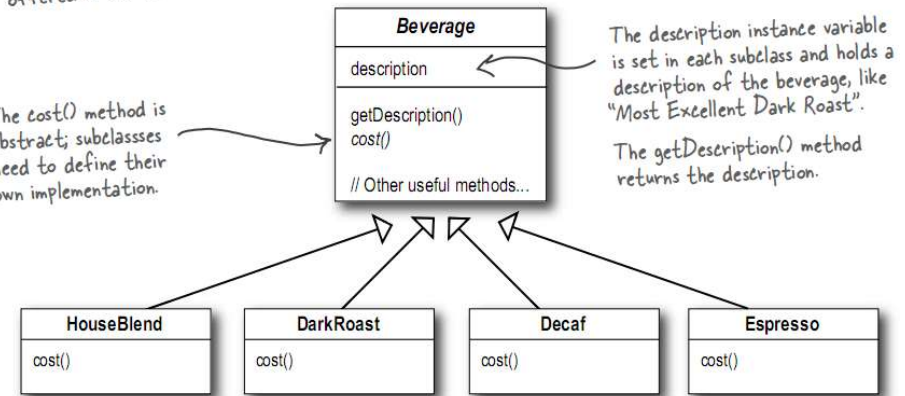


5

## Example: Cost of Beverage

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

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



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.

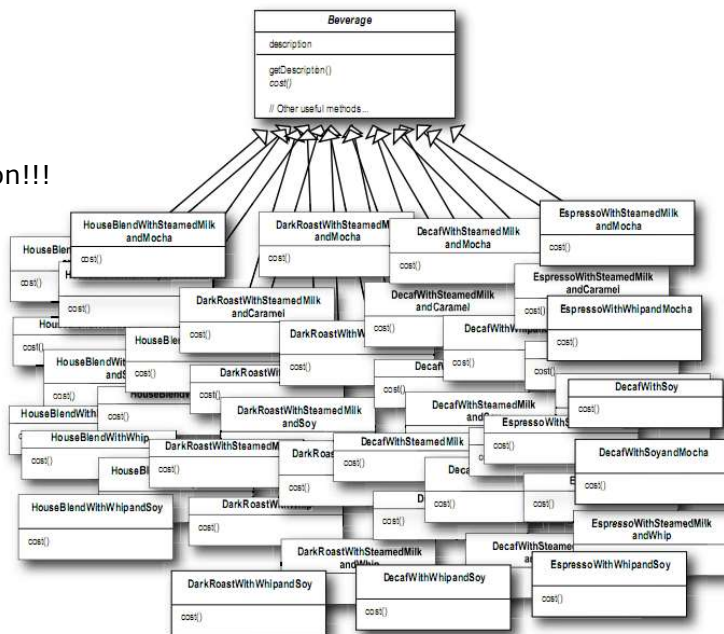
Thêm một số thành phần phụ vào 4 loại nước đã có ?

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

6

❖ 1<sup>st</sup> idea

Class explosion!!!



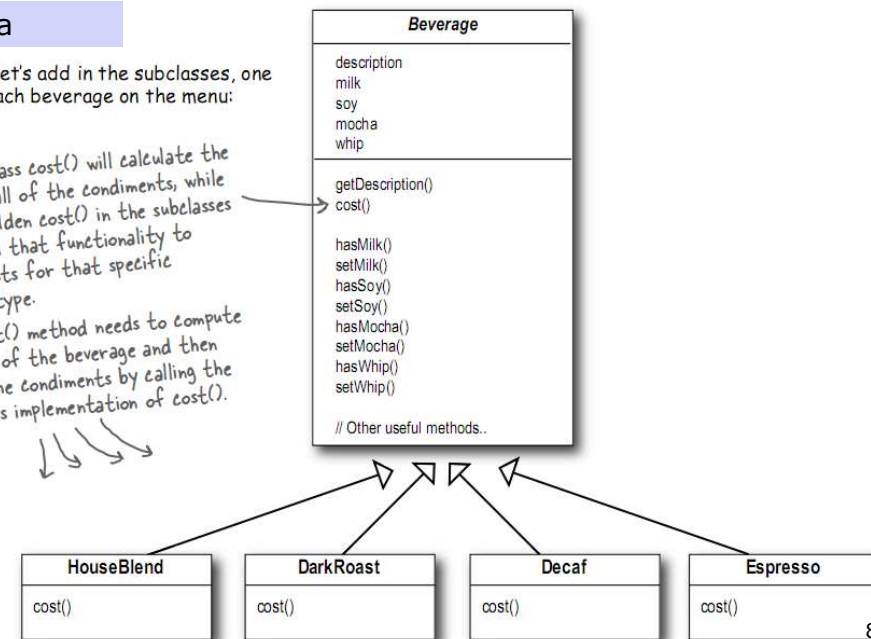
7

2<sup>nd</sup> 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

## ❖ The Open-Closed Principle:

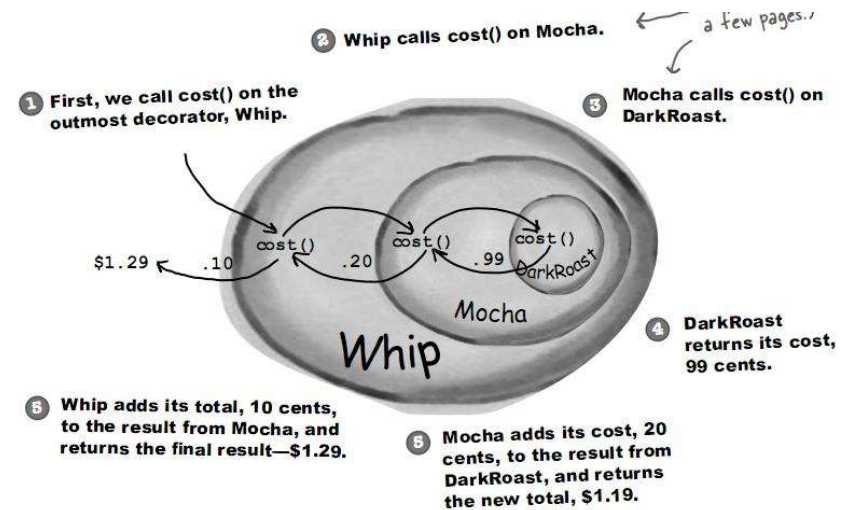


### Design Principle

*Classes should be open for extension, but closed for modification.*

9

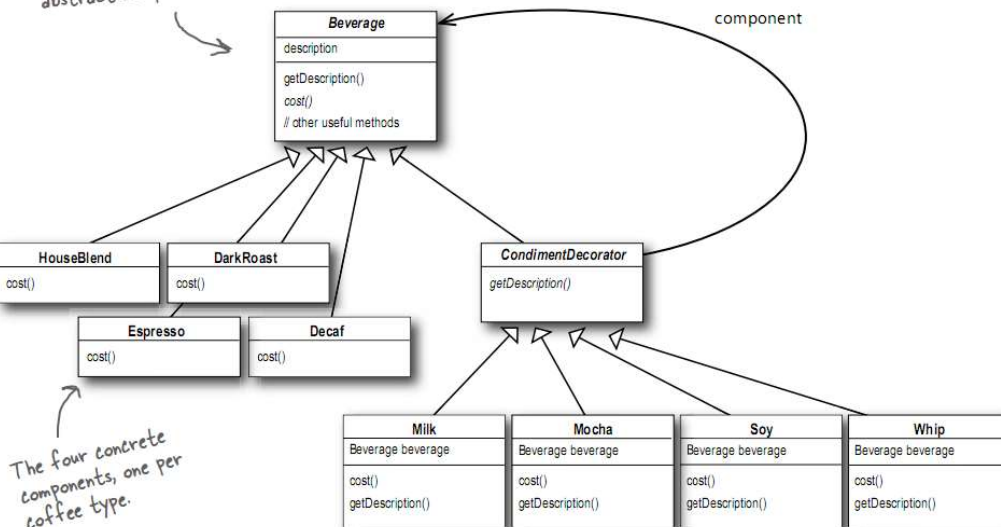
## ❖ 3<sup>rd</sup> idea



10

*Beverage acts as our abstract component class.*

component



11

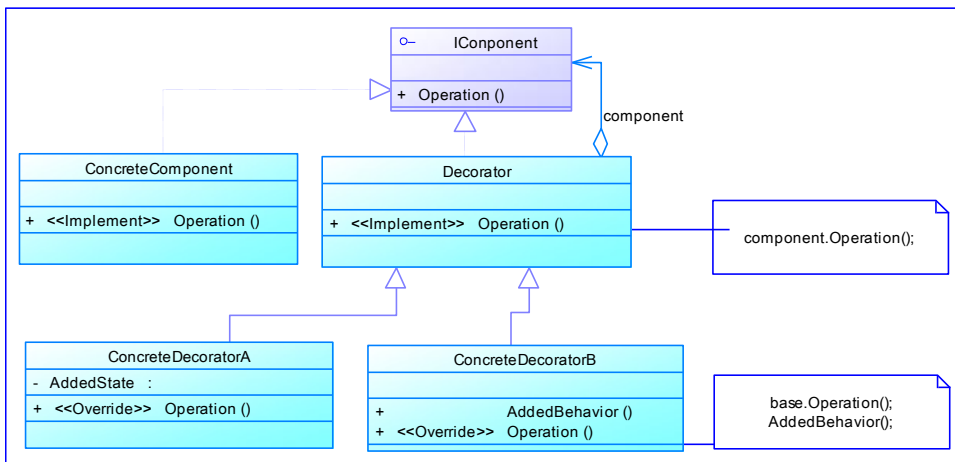
## Decorator pattern

### ❖ 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

## ❖ Cấu trúc



13

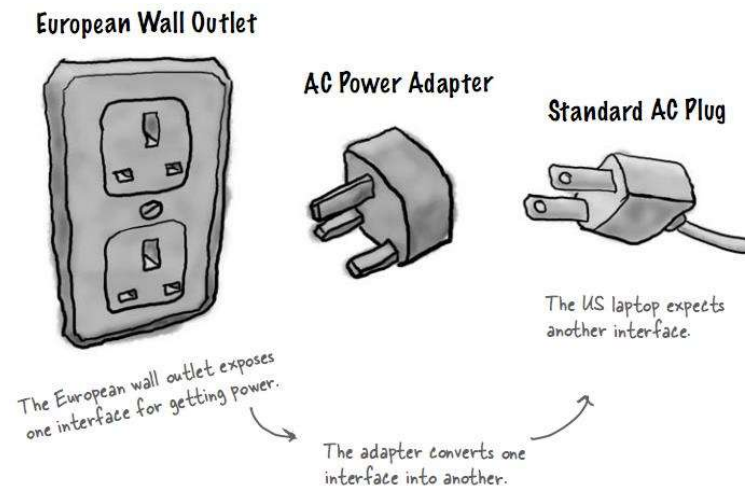
## Questions

- ❖ Vai trò của lớp Decorator, có thể không cần dùng lớp Decorator được không?
- ❖ Nêu mối liên hệ giữa ConcreteComponent và ConcreteDecorator

14

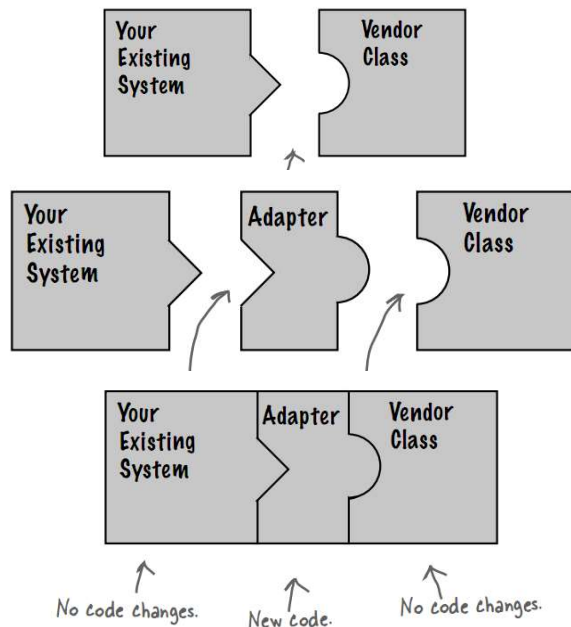
## Adapter pattern

## Adapter pattern: Example



16

## Adapter pattern: Example



17

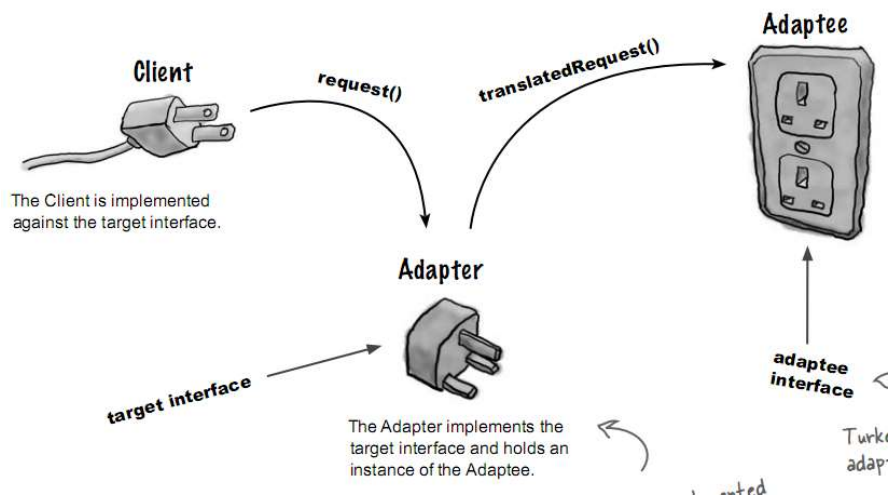
## Adapter pattern

### ❖ Mục đích:

- Chuyển giao diện của một lớp thành một giao diện khác mà client sử dụng
- Cho phép các lớp có giao diện không tương thích cùng làm việc với nhau.

18

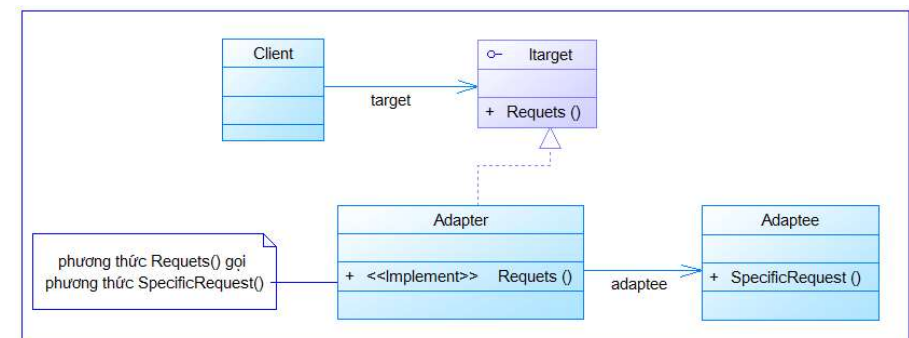
## Adapter pattern: Idea



19

## Adapter pattern

### ❖ Cấu trúc:



20

## Questions

- ❖ Vai trò của lớp Adapter ?
- ❖ Adapter chỉ được sử dụng cho một lớp Adaptee duy nhất?

21

## *Façade pattern*

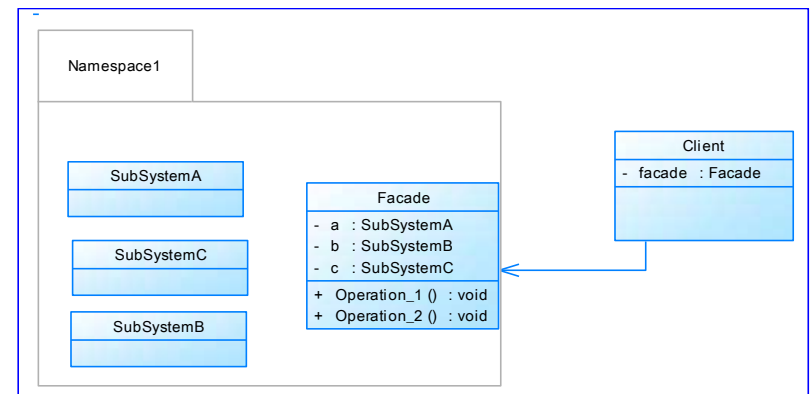
## Façade pattern

- ❖ Mục đích
  - Cung cấp một interface hợp nhất cho một tập các interface trong một subsystem
  - Định nghĩa một interface ở mức cao làm cho việc sử dụng subsystem trở nên dễ dàng hơn

23

## Façade pattern

- ❖ Cấu trúc:



24

- ❖ Each unit should only talk to its friends; don't talk to strangers.
- ❖ Reduce the interactions between objects to just a few close “friend



### **Design Principle**

*Principle of Least Knowledge -  
talk only to your immediate friends.*

25

## **Proxy pattern**

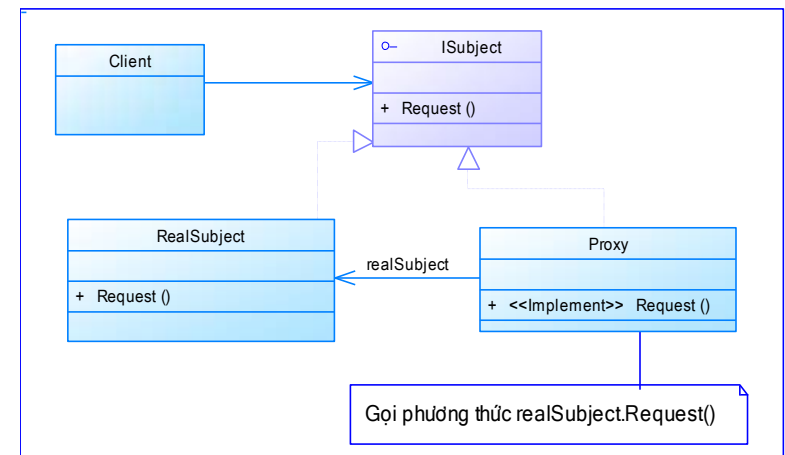
## **Proxy pattern**

- ❖ Mục đích:
  - Cung cấp một đối tượng thay thế hay một trình giữ chỗ cho một đối tượng khác để kiểm soát việc truy cập tới đối tượng đó.
  - Sử dụng Proxy pattern để tạo một đối tượng đại diện để kiểm soát truy cập tới một đối tượng khác:
    - Ở xa
    - Expensive to create
    - Cần được bảo vệ

27

## **Proxy pattern**

### ❖ Cấu trúc



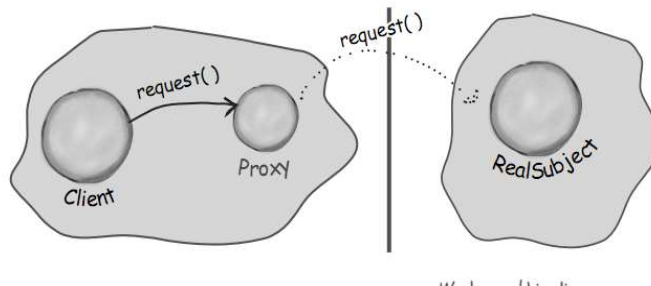
28



## Remote proxy

### Remote Proxy

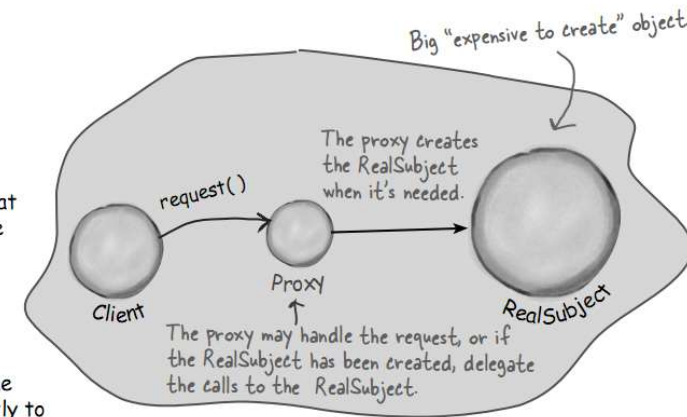
With Remote Proxy, the proxy acts as a local representative for an object that lives in a different JVM. A method call on the proxy results in the call being transferred over the wire, invoked remotely, and the result being returned back to the proxy and then to the Client.



29

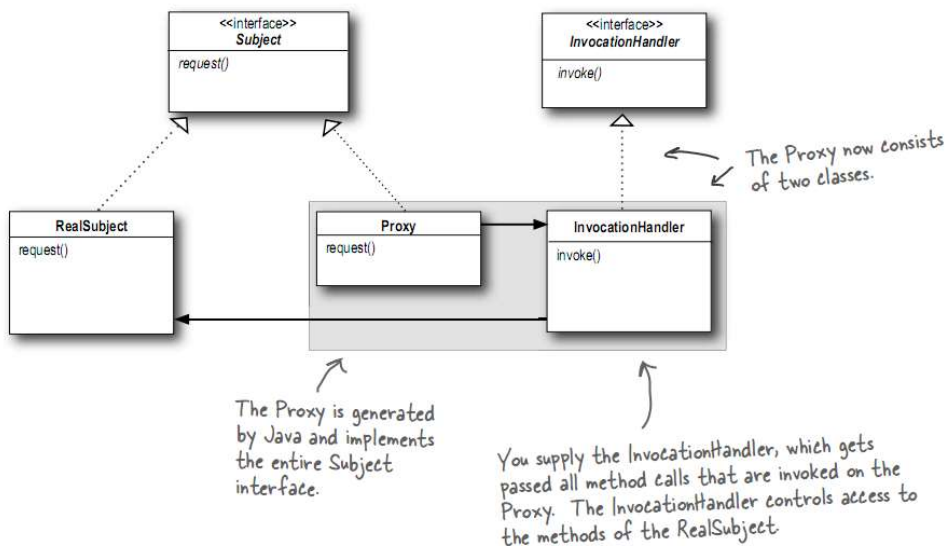
### Virtual Proxy

Virtual Proxy acts as a representative for an object that may be expensive to create. The Virtual Proxy often defers the creation of the object until it is needed; the Virtual Proxy also acts as a surrogate for the object before and while it is being created. After that, the proxy delegates requests directly to the RealSubject.



30

## Protection proxy



31

## Questions

32



# Bridge pattern

## Example: Remote Control

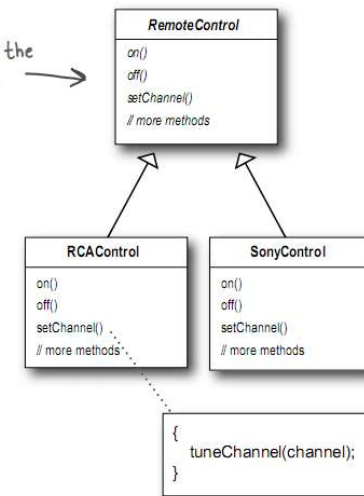
- ❖ Các loại remote TV có cùng chung một giao diện trừu tượng và nhiều cách thực thi khác nhau cho nhiều loại TV

Every remote has the same abstraction.

- ❖ Vấn đề:

- Remote lần đầu tiên thiết kế không chuẩn và cần phải cải tiến
  - Cả TV và Remote đều thay đổi

Lots of implementations, one for each TV.

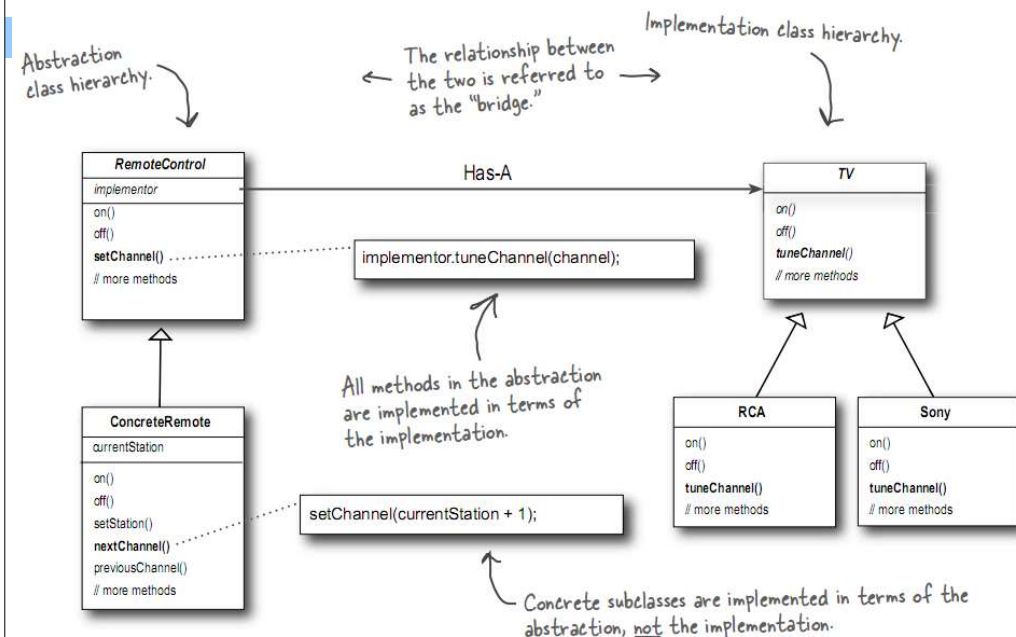


34

## Bridge pattern

- ❖ Mục đích:

- Tách rời phần trừu tượng ra khỏi sự thực thi của nó sao cho cả hai có thể biến đổi không phụ thuộc nhau



35

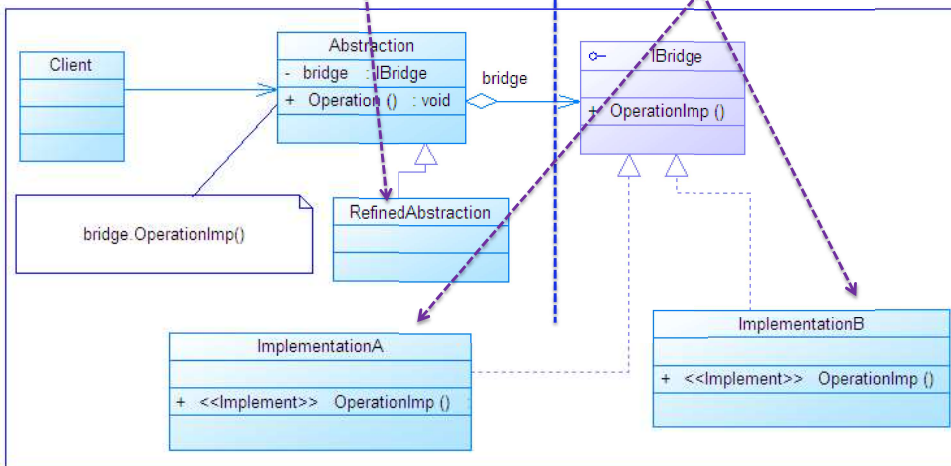
36

## Bridge pattern

### ❖ Cấu trúc

Biến đổi phần trừu tượng (IBridge)

Biến đổi phần thực thi



37

## Bridge pattern

### ❖ Ưu điểm:

- Tách rời sự thực thi sao cho chúng không kết nối vĩnh viễn với một giao diện
- Phần Abstraction và phần Implementation có được thể mở rộng không phụ thuộc nhau
- Thay đổi các lớp concrete abstraction không ảnh hưởng đến client

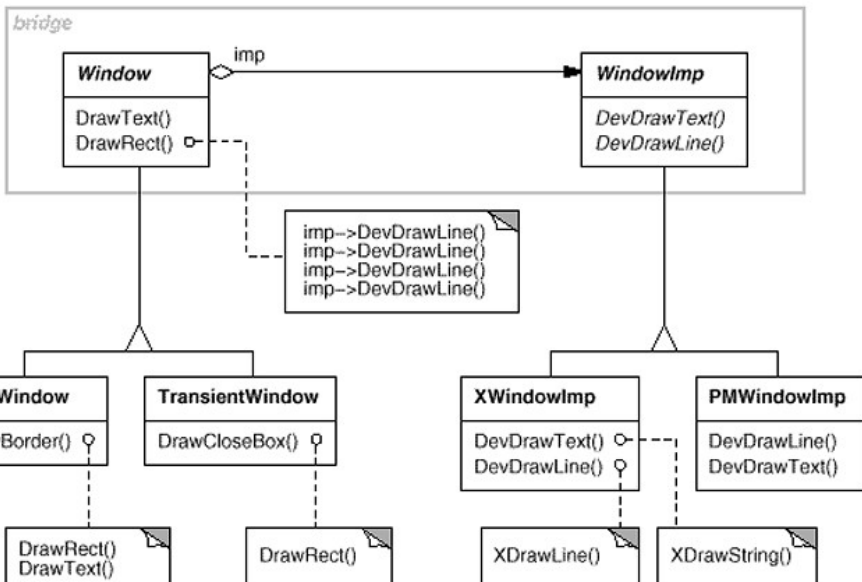
### ❖ Hạn chế:

- Gia tăng sự phức tạp

### ❖ Sử dụng:

- Sử dụng khi cần biến đổi interface và implementation theo các cách khác nhau
- Trong các hệ thống đồ họa và hệ thống của windows phải chạy trên nhiều nền tảng khác nhau.

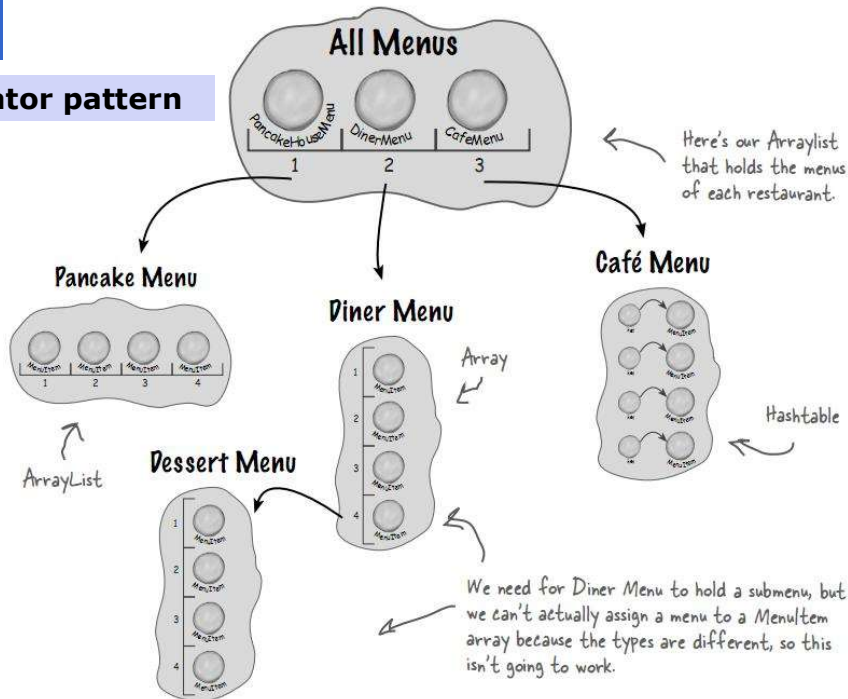
38



39

## Composite pattern

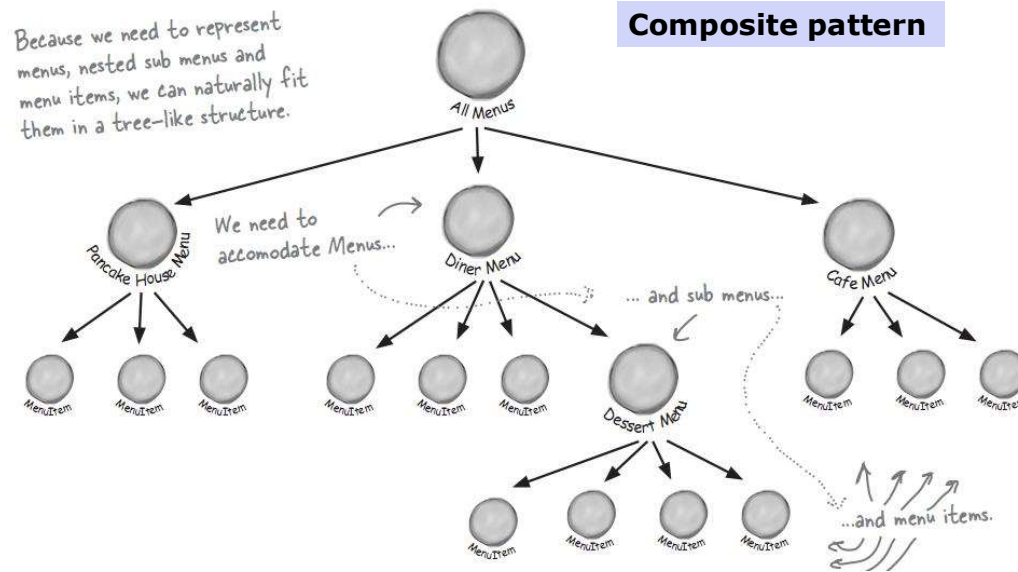
## Iterator pattern



41

## Composite pattern

Because we need to represent menus, nested sub menus and menu items, we can naturally fit them in a tree-like structure.



42

## Composite pattern

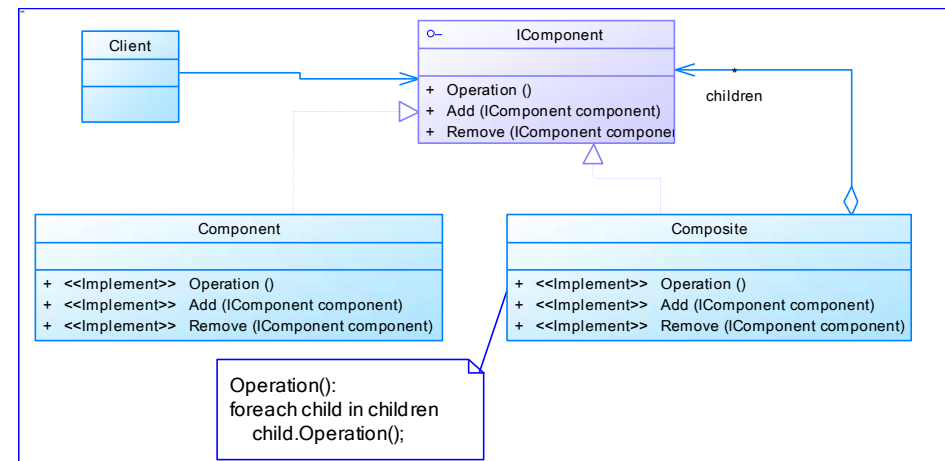
### ❖ Mục đích:

- Sắp xếp các đối tượng vào các cấu trúc cây để biểu diễn các phân cấp part-whole giữa các đối tượng
- Đãi xử với các đối tượng riêng lẻ và nhóm các đối tượng theo các cách giống nhau

43

## Composite pattern

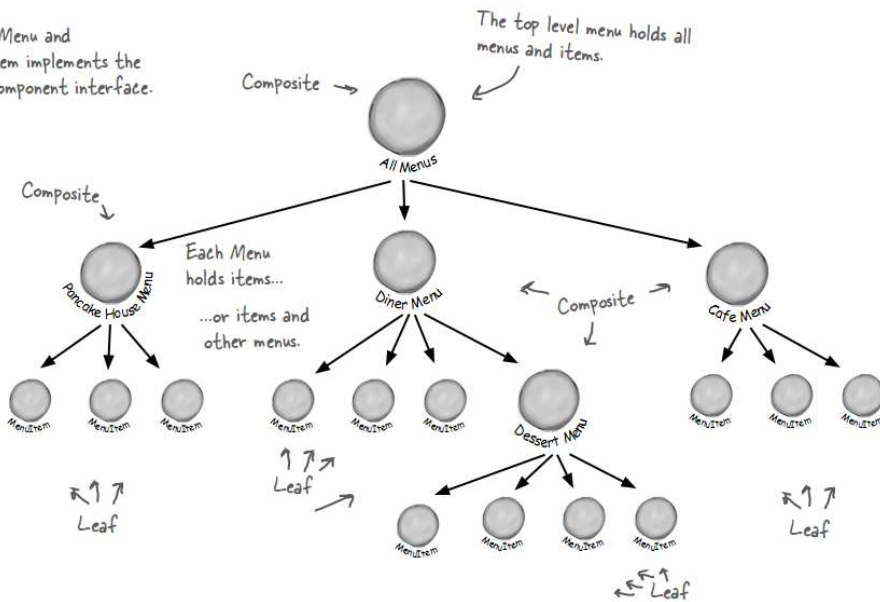
### ❖ Cấu trúc:



44

## Composite pattern: Example

Every Menu and MenuItem implements the MenuComponent interface.



45

## Composite pattern

- ❖ Composite Pattern cho phép xây dựng các cấu trúc của các đối tượng dưới dạng cây với các node là:
  - composition of objects
  - individual objects
- ❖ Sử dụng cấu trúc Composite ta có thể áp dụng cùng một Operation cho cả hai loại đối tượng Composite và Individual  
→ Làm mất đi sự khác biệt đối với hai loại đối tượng

46

## Questions

- ❖ So sánh giữa Iterator Pattern và Composite Pattern. Có thể thay thế Composite Pattern bằng Iterator Pattern được không

47

## Tài liệu tham khảo

- ❖ Eric Freeman, Elisabeth Freeman, Kathy Sierra, Bert Bates. Head First Design pattern. O'Reilly 2006.
- ❖ **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.** Design Patterns Elements of Reusable Object-Oriented Software. Addison-Wesley 1995
- ❖ <http://www.dofactory.com/Patterns/Patterns.aspx>

48