Behavioral patterns

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

Behavioral patterns

- Liên quan tới các giải thuật và các quan hệ giữa các đối tượng, cách các đối tượng giao tiếp với nhau

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

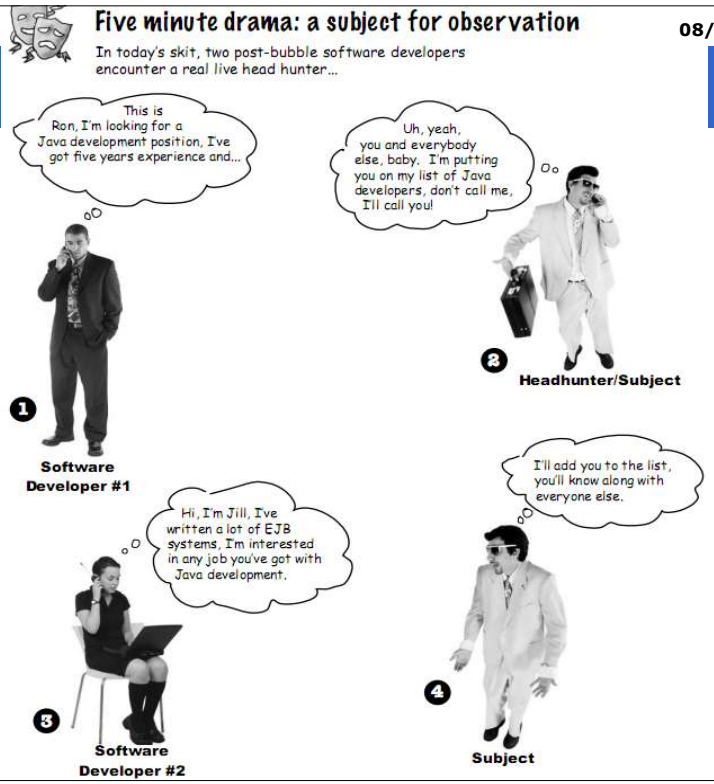
Observer pattern

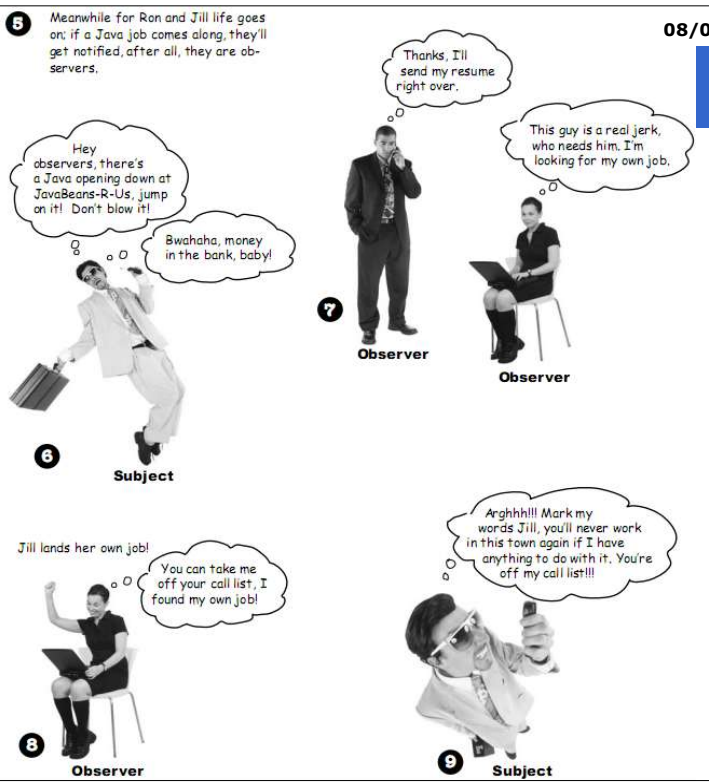
observer pattern

- 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.

Các ví dụ về observer pattern

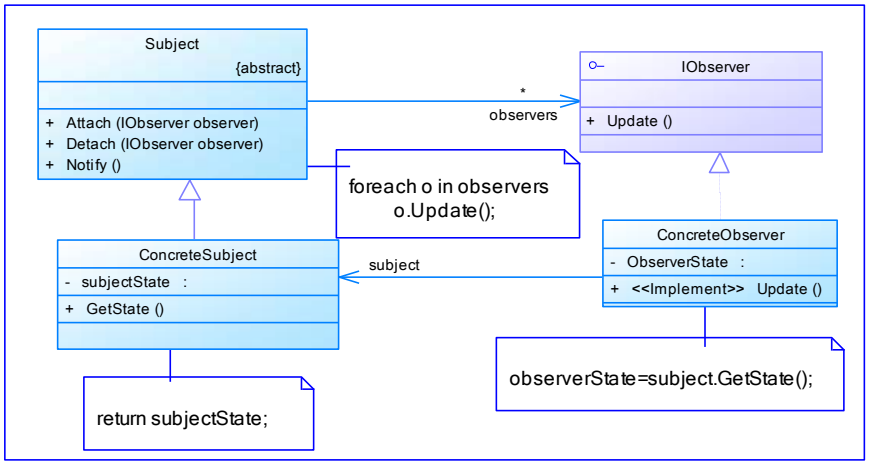
- Chương trình bảng tính Excel
- Data binding
- Đặt mua báo dài hạn
  - Độc giả đăng ký mua báo dài hạn với tòa soạn
  - Khi có một tờ báo mới xuất bản thì nó được đại lý phân phối đến đọc giả.
  - **Publishers + Subscribers = Observer Pattern**





Observer pattern

- Cấu trúc



Questions

- Giải thích vai trò của hàm Update() từ đó cho biết vai trò của IObserver
- 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

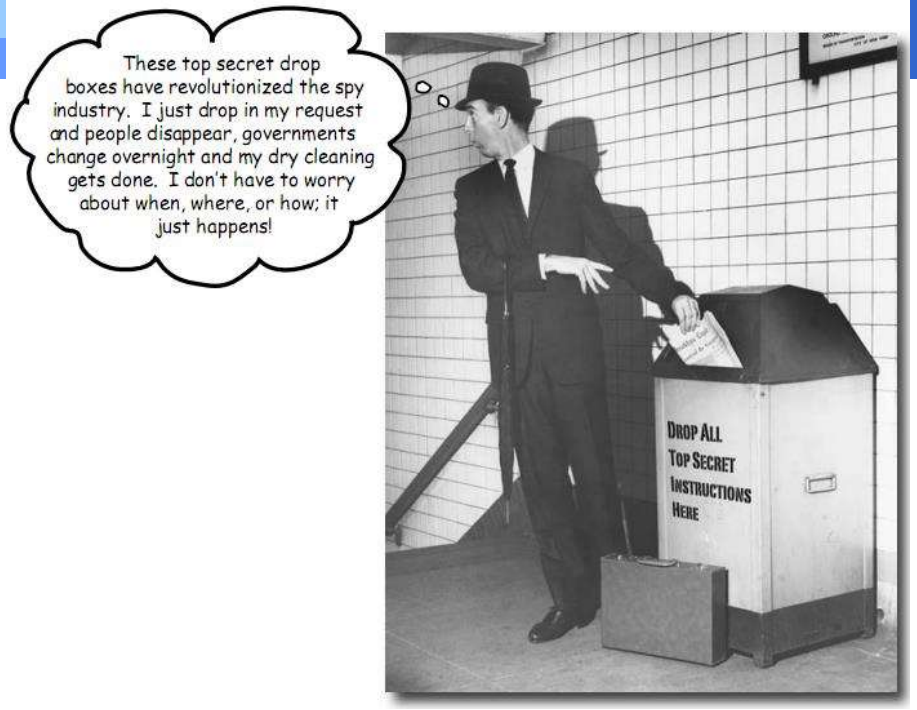
Bài tập

- Tìm hiểu mô hình lập trình RxJava
- Tìm hiểu mô hình lập trình RxJava cho Android

Design Principle

Strive for loosely coupled designs between objects that interact.

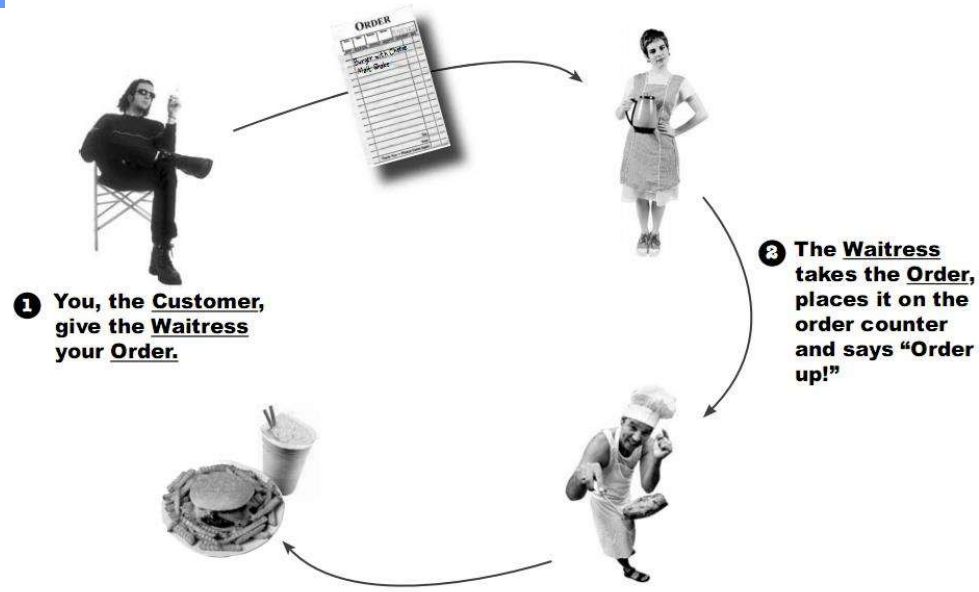
Command pattern

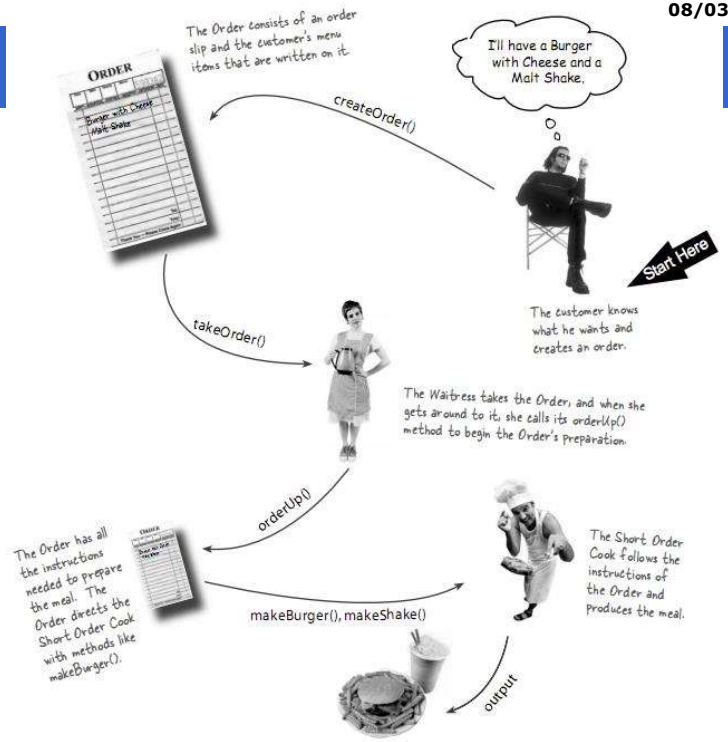


Command pattern

- Mục đích:
  - Đóng gói requets thành một đối tượng.
  - Cho phép tham số hóa các client với các requets khác nhau
  - Tách rời request một hành động ra khỏi đối tượng thực hiện hành động đó

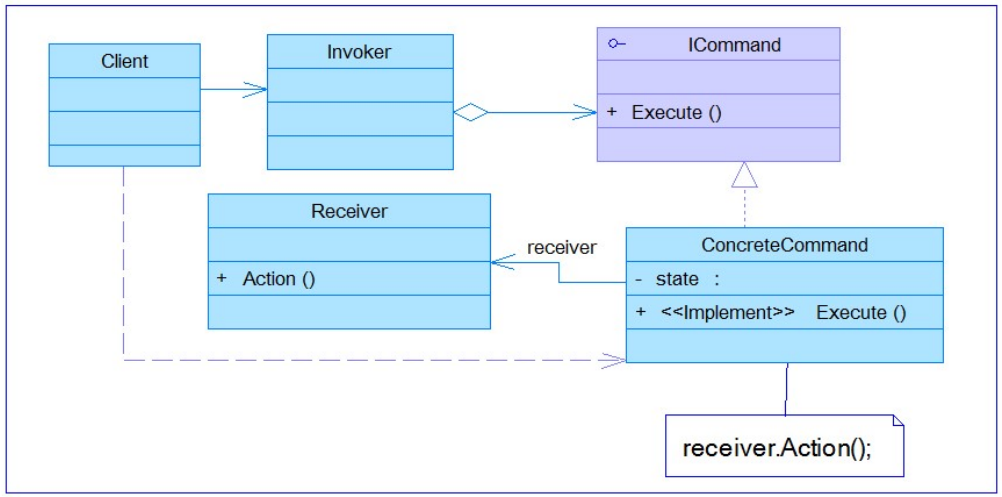
Example: Diner operates



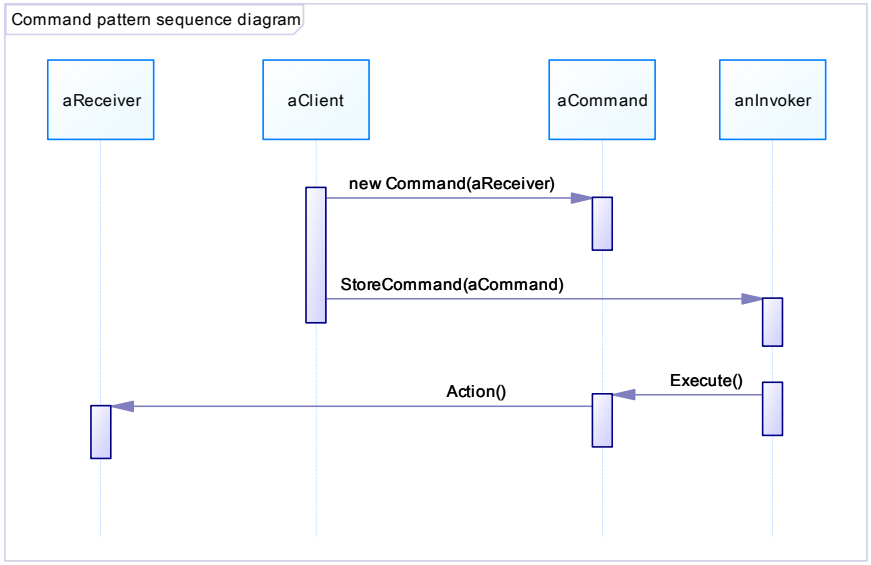


Command pattern

- Cấu trúc:

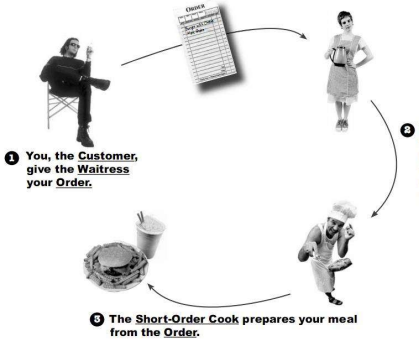


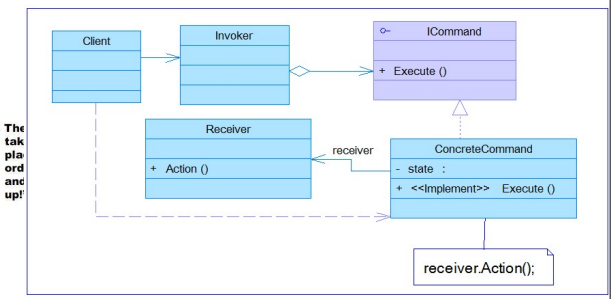
Collaboration



Matching between Command & Diner Example

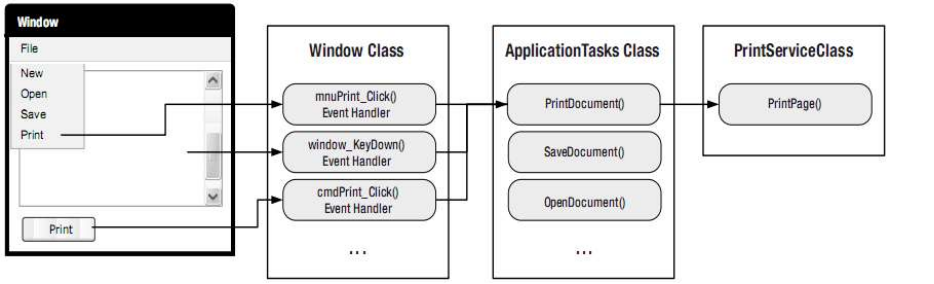
Diner Example	Command pattern
Customer	Client
Order	Command
Waitress	Invoker
TakeOrder()	setCommand()
Short Order Cook	Receiver
orderUp()	Execute()



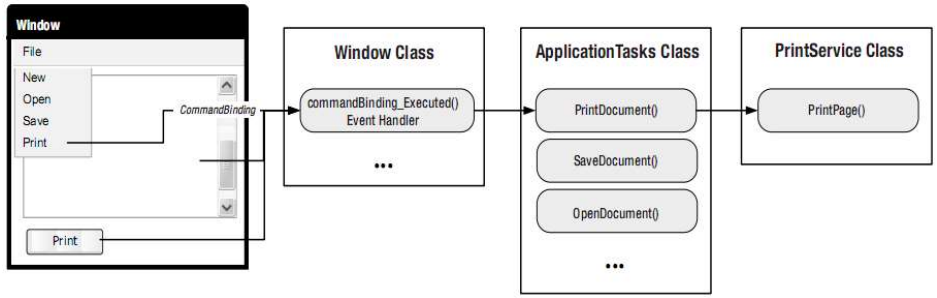


Ứng dụng

Windows Form



WPF

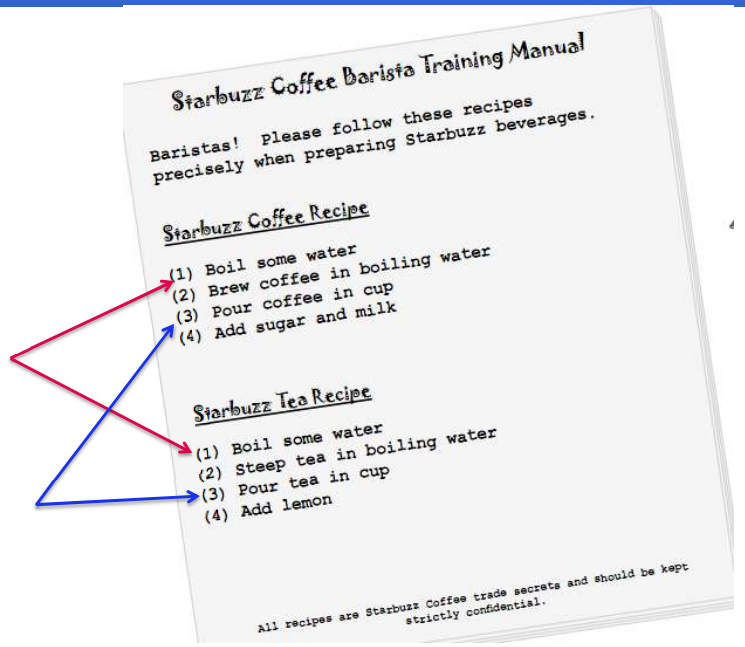


Questions

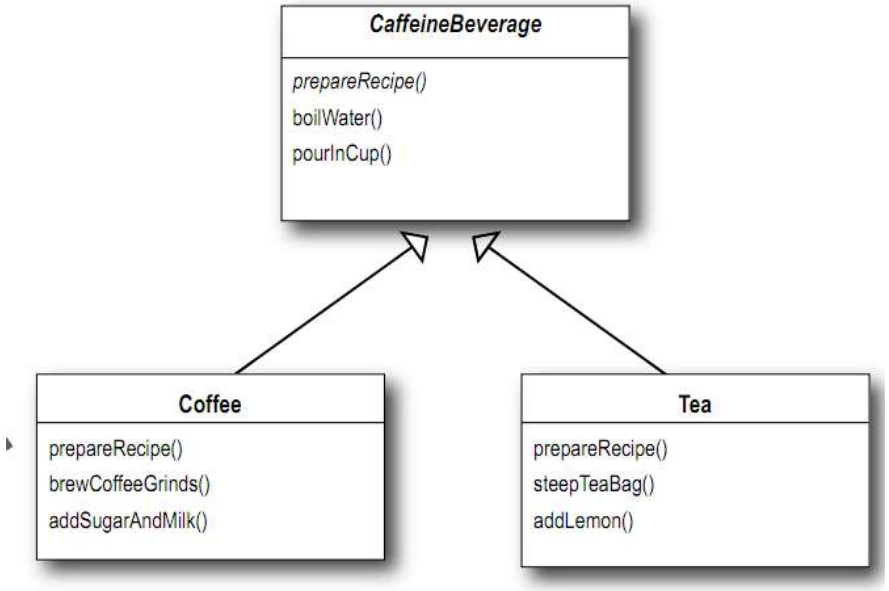
- Làm thế nào Command pattern tách rời sự phụ thuộc giữa *invoke of a request và receiver of the request?*

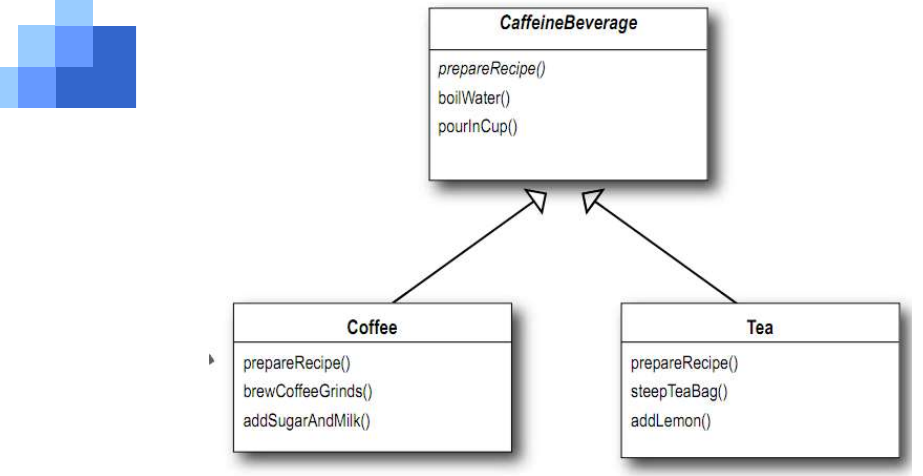
Template method

Tea and Coffee examples



Tea and Coffee example





Coffee

```
void prepareRecipe() {
    boilWater();
    brewCoffeeGrinds();
    pourInCup();
    addSugarAndMilk();
}
```

Tea

```
void prepareRecipe() {
    boilWater();
    steepTeaBag();
    pourInCup();
    addLemon();
}
```


Tea and Coffee example…

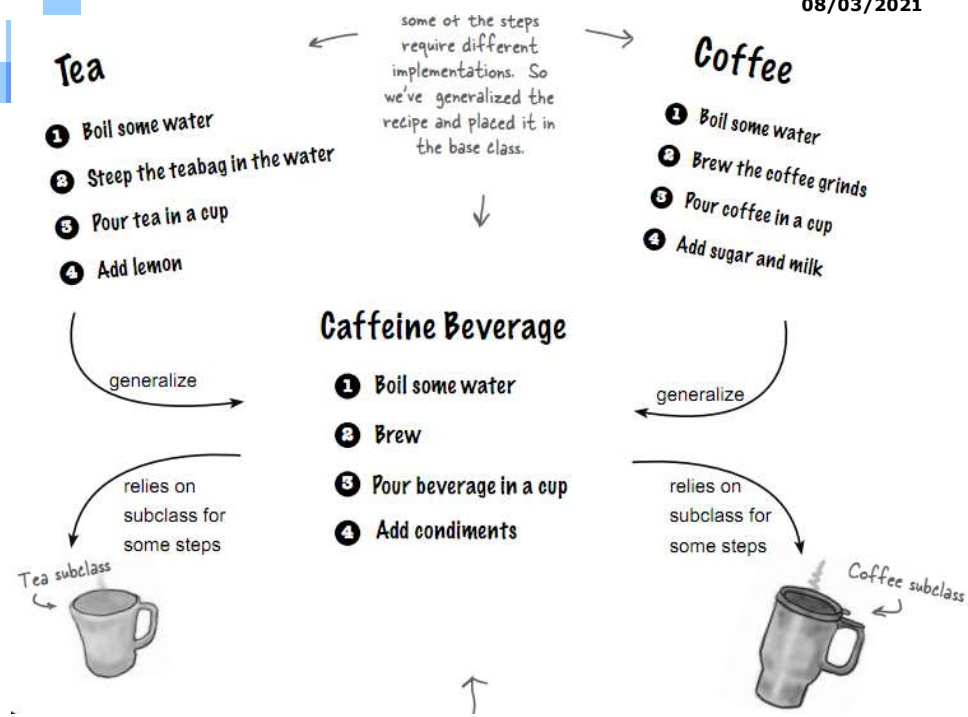
Coffee

```
void prepareRecipe() {
    boilWater();
    brewCoffeeGrinds();
    pourInCup();
    addSugarAndMilk();
}
```

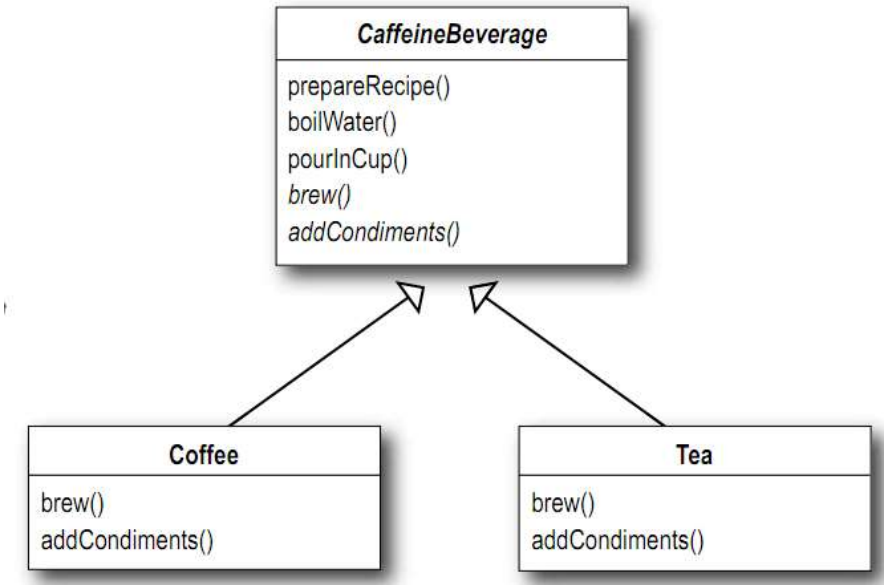
Tea

```
void prepareRecipe() {
    boilWater();
    steepTeaBag();
    pourInCup();
    addLemon();
}
```

```
void prepareRecipe() {
    boilWater();
    brew();
    pourInCup();
    addCondiments();
}
```

Tea and Coffee example…

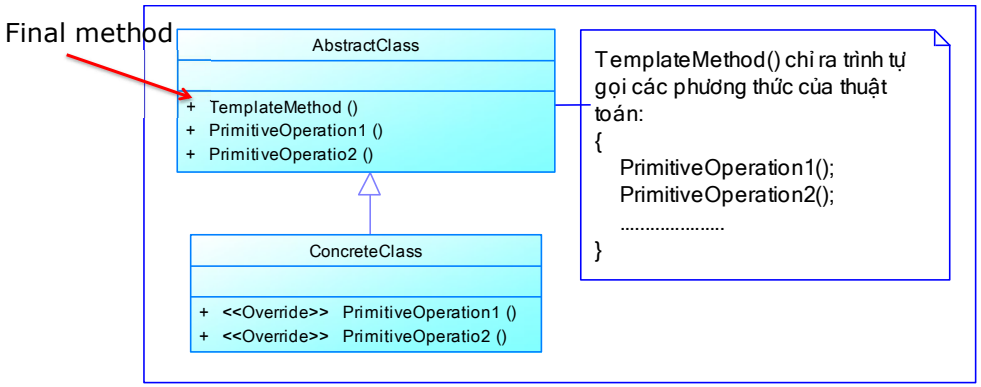


template method

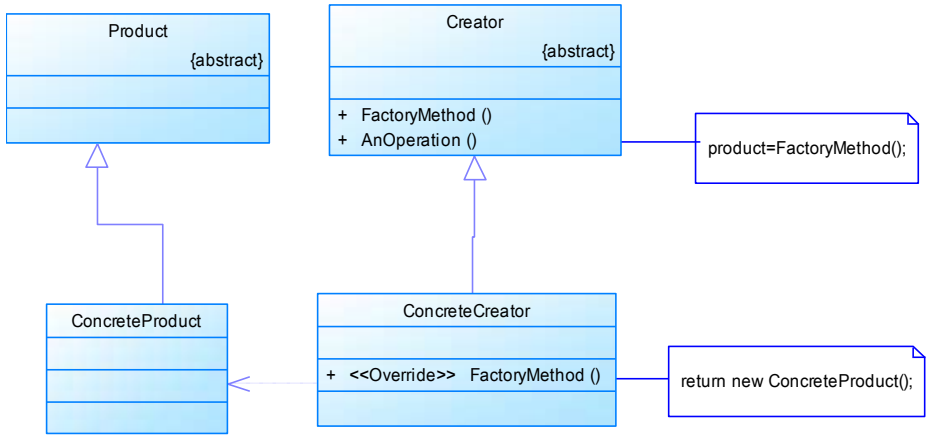
- Mục đích:
  - Định nghĩa khung sườn của một thuật toán bao gồm nhiều bước trong một phương thức.
    - Một số bước được khai báo abstract ở lớp cơ sở và ủy quyền cho lớp con cài đặt
    - Việc cài đặt các bước ở lớp con không làm ảnh hưởng đến cấu trúc của thuật toán.

Template method

- Cấu trúc

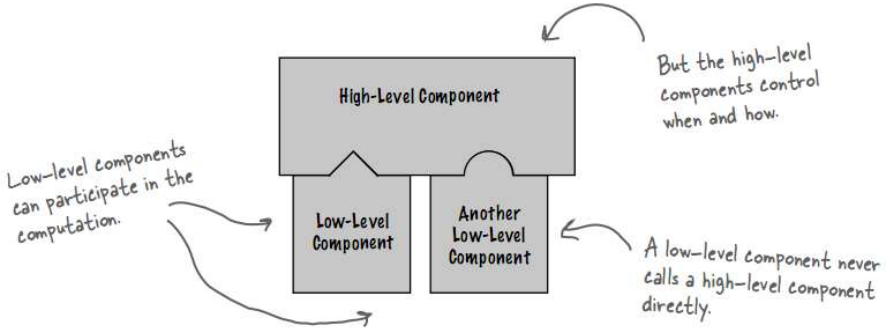


Liên hệ giữa factory method và template method



The Hollywood Principle

Don't call us, we'll call you.



Questions

- Có thể dùng interface để thay thế AbstractClass không? Nếu được hãy vẽ lại sơ đồ cấu trúc của template method?
- Các ConcreteClass có cần phải thực thi toàn bộ các phương thức của lớp AbstractClass?
- So sánh Template method với Strategy pattern, factory method

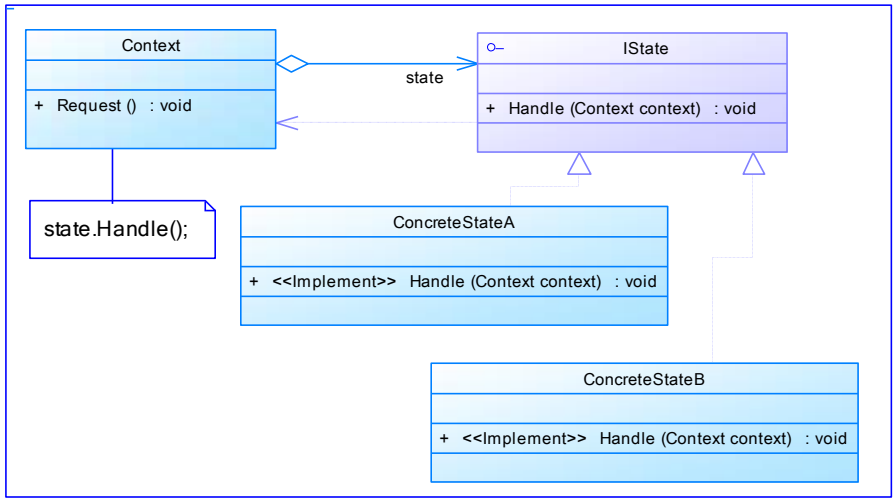
State pattern

State pattern

- **Mục đích:** Cho phép một đối tượng thay đổi hành vi của nó khi trạng thái bên trong của nó thay đổi. Đối tượng đó sẽ xuất hiện để thay đổi lớp của nó.

State pattern

- Cấu trúc:



State pattern: Sử dụng

- Một đối tượng có nhiều trạng thái
  - Hành vi của đối tượng phụ thuộc vào các trạng thái của đối tượng và có thể thay đổi lúc run-time
  - Nhiều operation phụ thuộc vào trạng thái của đối tượng. Thông thường chỉ vài operation cùng phụ thuộc vào trạng thái của đối tượng.
- Sử dụng State pattern:
  - Đối tượng → Context
  - Mỗi trạng thái →ConcreteState
    - ConcreteState chứa các operation phụ thuộc vào trạng thái
  - Chuyển trạng thái của đối tượng: ConcreteState, Context
    - Handle(Context context)

Questions

- Trong State pattern class diagram, IState là một interface, có thể thay thế IState bằng một abstract class được không? Nếu được hãy chỉ ra khi nào thì thay thế
- State pattern làm tăng số lớp cài đặt của ứng dụng. Tại sao ta vẫn sử dụng State pattern trong OOD (Object Oriented Design)
- Giả sử trong Context có biến s kiểu IState. Giải thích vì sao một lớp ConcreteState có thể thay đổi trạng thái của biến s.
- Client có thể tương tác với biến kiểu IState trong context được không? giải thích.

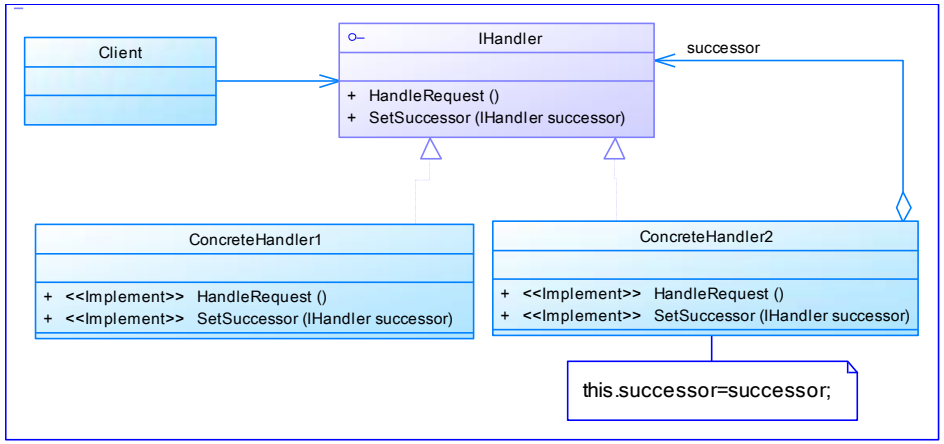
Chain of Responsibility

Chain of Responsibility

- Mục đích:
  - Tách rời người gởi và người 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

Chain of Responsibility

- Cấu trúc:



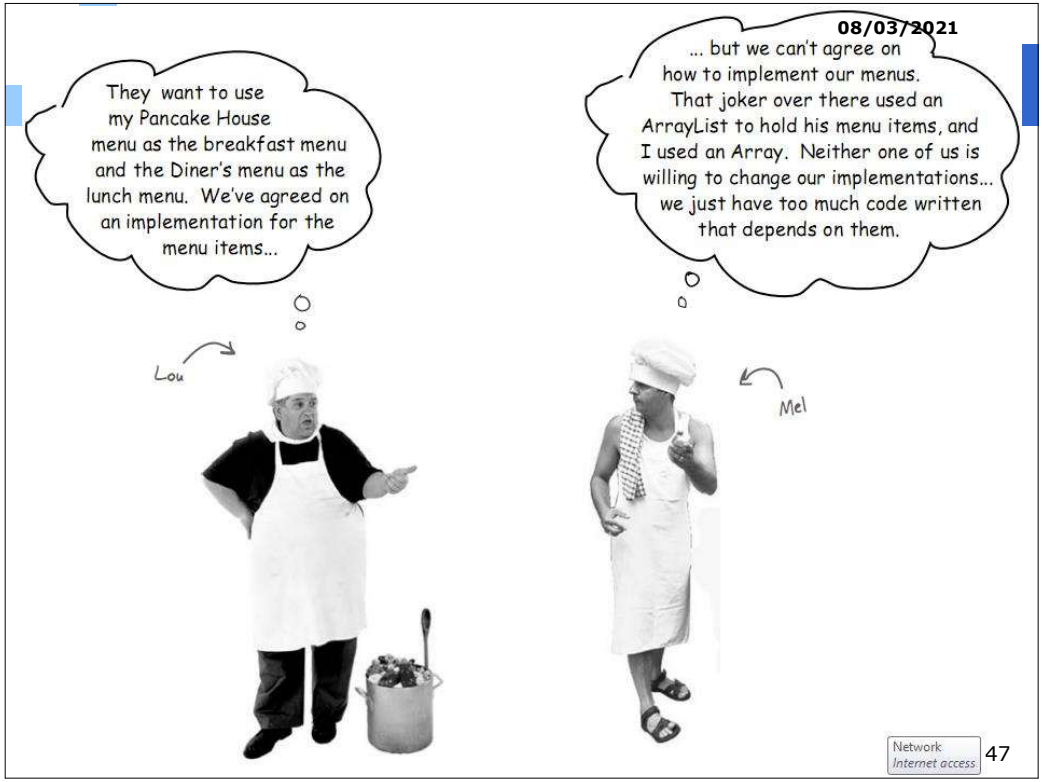
Question

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

Sử dụng

- "Người gởi" không biết phải gởi request cho "người nhận" nào trong tập các "người nhận"
- Chuỗi các "người nhận" có thể thay đổi lúc run-time

Iterator pattern

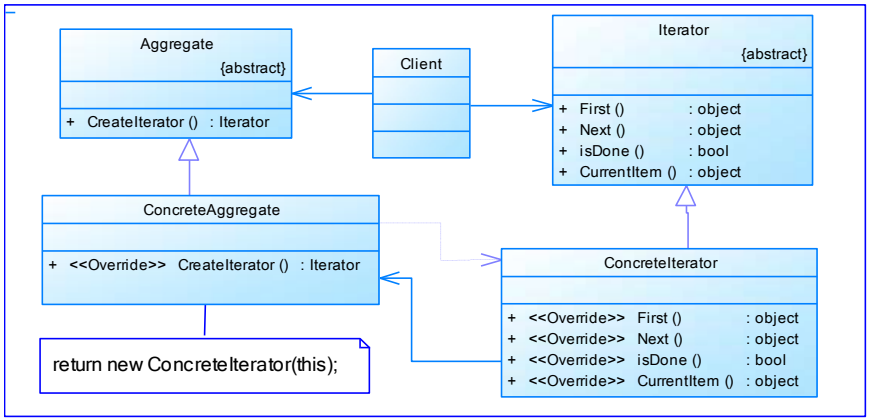


Iterator

- 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

Iterator

- 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



Questions

- Có thể gộp chung hai lớp Iterator và Aggregate được không? Nêu ưu và khuyết điểm của cách làm này

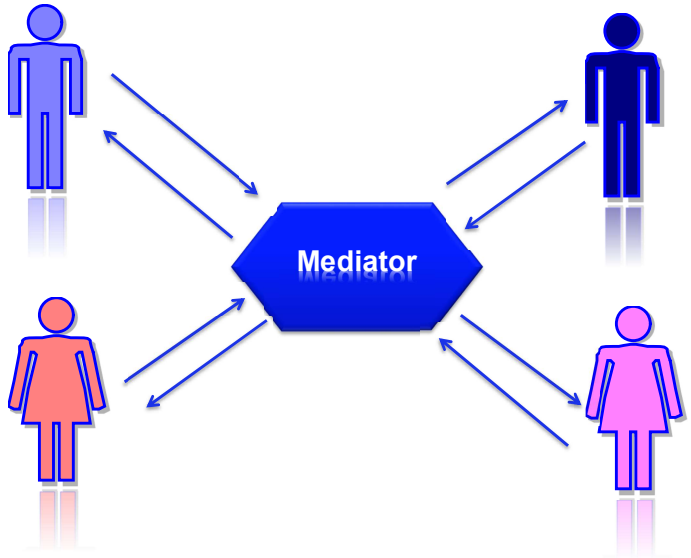
Design Principle

A class should have only one reason to change.

Mỗi class chỉ nên thiết kế với một single responsibility

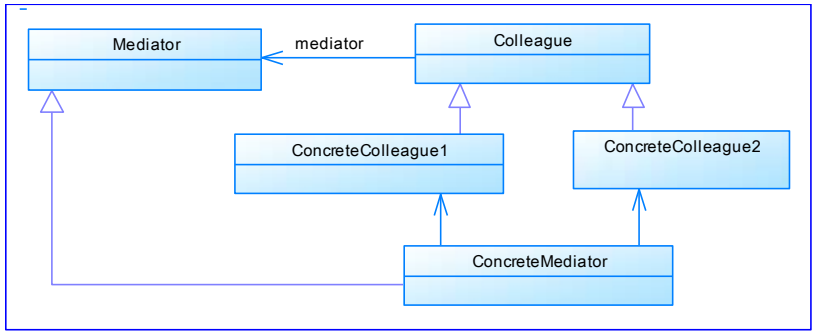
Mediator pattern

Mediator: Example



Mediator pattern

- Mục đích:
  - Đóng gói các cách tương tác giữa một tập các đối tượng



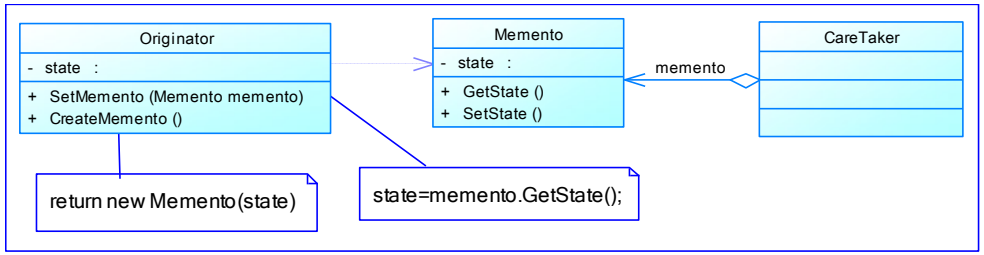
Mediator

- Ưu điểm
  - Gia tăng việc tái sử dụng các đối tượng được hỗ trợ bởi Mediator bằng cách tách rời chúng ra khỏi hệ thống
  - Đơn giản hóa việc duy trì hệ thống bằng cách tập trung logic điều khiển
  - Đơn giản hóa và giảm sự thay đổi các message được gởi giữa các đối tượng trong hệ thống
- Hạn chế
  - Mediator có thể rất phức tạp nếu không được thiết kế một cách phù hợp.
- Sử dụng
  - Mediator thường được sử dụng để phối hợp các thành phần GUI với nhau

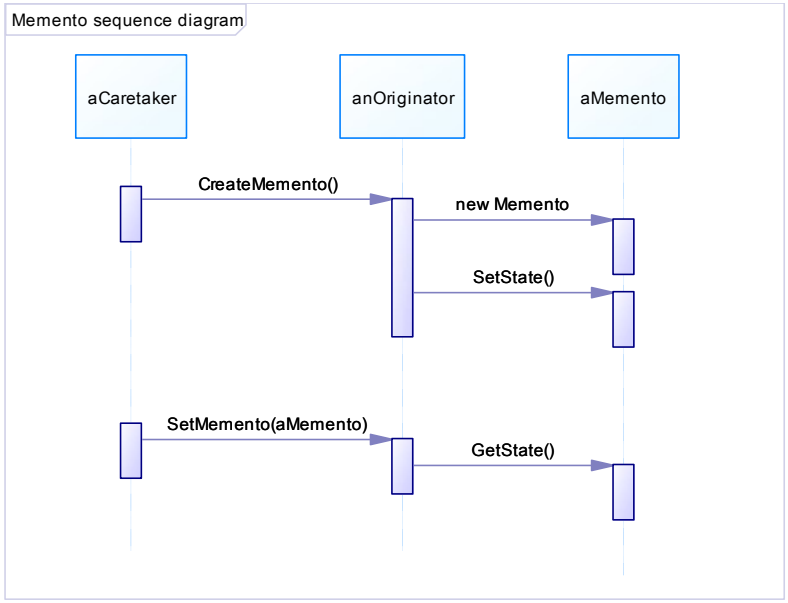
Memento pattern

Memento pattern

- Mục đích:
  - Lưu lại trạng thái của một đối tượng để khôi phục lại sau này mà không vi phạm nguyên tắc đóng gói.



Collaboration



Visitor pattern

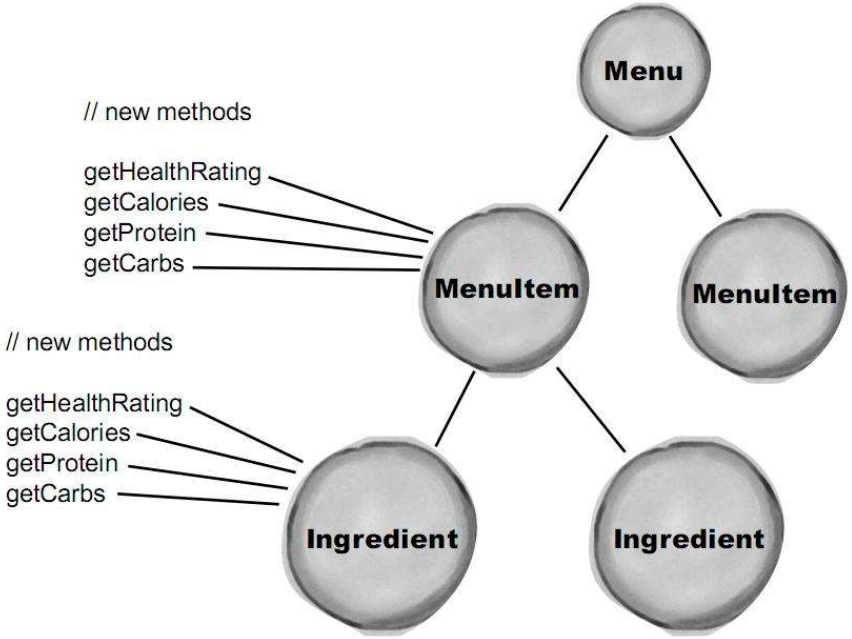
Vấn đề

- Xử lý thông tin của một phần tử trong một cấu trúc đối tượng cho trước
- Việc xử lý thông tin như thế nào chưa thể xác định lúc compile-time

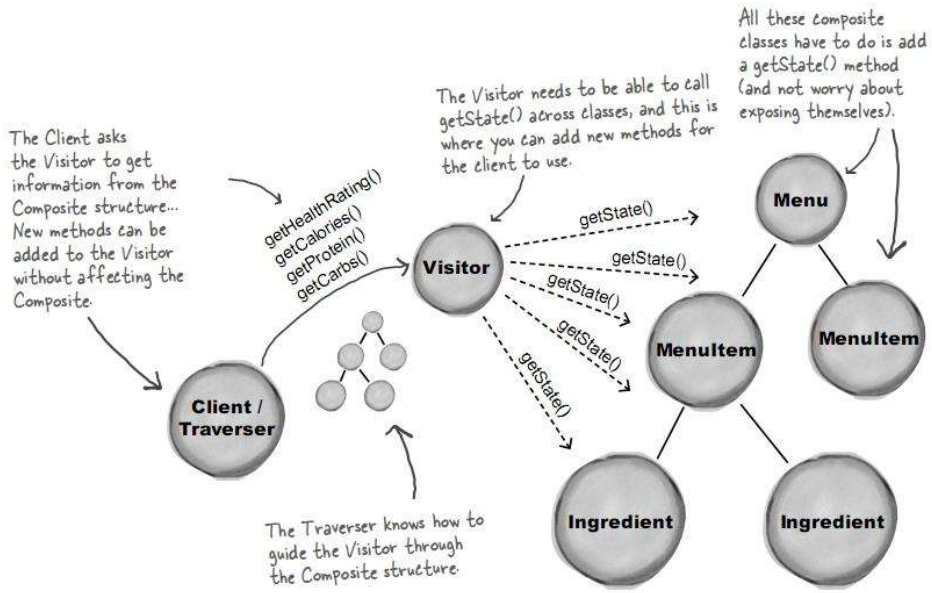
Visitor pattern

- Mục đích:
  - Thực hiện một thao tác trên một phần tử của một cấu trúc phức hợp (composite structure)
  - Định nghĩa phương thức mới thao tác trên một phần tử của cấu trúc mà không cần thay đổi các lớp đã được định nghĩa trên cấu trúc đó

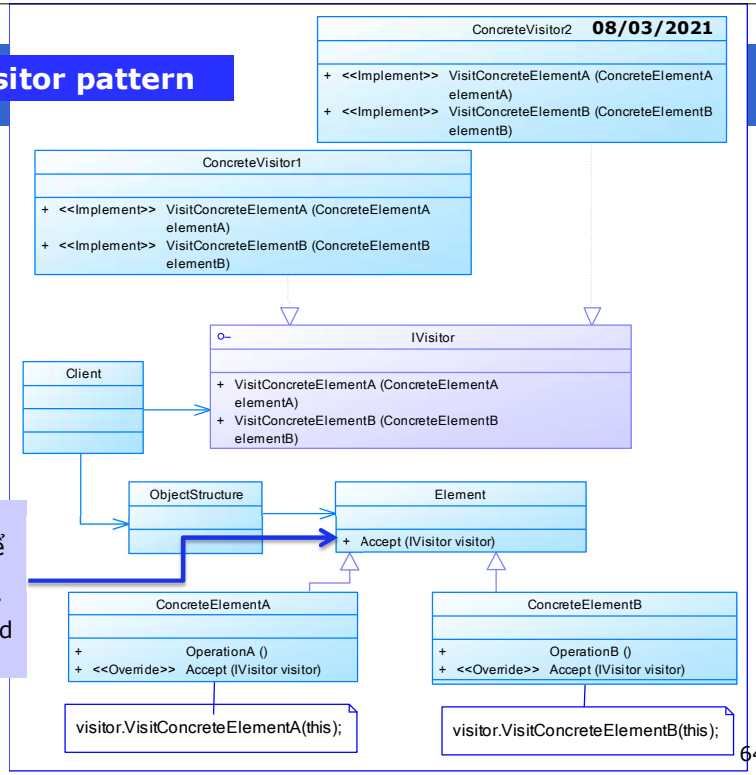
Visitor pattern: Example



Visitor pattern: Example



Giải pháp: Visitor pattern



Định nghĩa trước một khuôn dạng để cho phép các đối tượng dạng IVisitor truy cập tới các field bên trong

Visitor pattern

- Ưu điểm
  - Cho phép thêm các thao tác xử lý tới một Composite structure mà không cần thay đổi cấu trúc đó
  - Việc thêm mới một operation khá dễ dàng
  - Mã lệnh của các operation được thực hiện bởi Visitor được tập trung
- Hạn chế
  - Tính chất đóng gói (encapsulation) của các lớp bị phá vỡ khi sử dụng Visitor
  - Việc thay đổi nó để phù hợp với cấu trúc Composite gặp nhiều khó khăn do phải dùng hàm duyệt cấu trúc của Composite.

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