Lập trình Java – Design Pattern

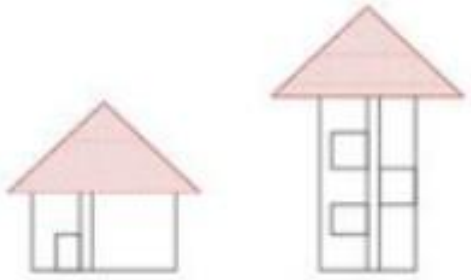
Ths. Vũ Duy Khương

Nội dung

1. Giới thiệu Design Pattern
2. Phân loại
3. Proxy
4. Observer
5. Singleton

Giới thiệu

- Ngữ cảnh
  - Kiến trúc sư Christopher Alexander ("Timeless way of building", 1979) đã phát triển ý tưởng để xác định các đối tượng khi đưa ra: Một giải pháp cho một vấn đề trong một ngữ cảnh – "A solution to a problem in a context"



  - Để xác định một mẫu (pattern), cần đưa ra lý do và cách thức để xây dựng từng giải pháp

Giới thiệu

- Design Pattern lần đầu được giới thiệu bởi Gang of Four (GoF) : Gamma Erich (PhD thesis), Richard Helm, Ralph Johnson và John Vlissides (1995).
- Đây là một nét "văn hóa mới" trong cộng đồng lập trình
- Pattern là thiết bị cho phép các chương trình chia sẻ kiến thức về thiết kế. Khi xây dựng chương trình, có nhiều vấn đề gặp phải và nó xuất hiện lại thường xuyên.
- Thư viện các pattern là nới để chúng ta có thể tìm thấy "Các giải pháp lập trình tốt nhất".

Giới thiệu

❑ Khái niệm

- Mẫu thiết kế (Design Pattern) là vấn đề thông dụng cần giải quyết và là cách giải quyết vấn đề đó trong một ngữ cảnh cụ thể
- Mẫu thiết kế không đơn thuần là một bước nào đó trong các giai đoạn phát triển phần mềm mà nó đóng vai trò là sáng kiến để giải quyết một vấn đề thông dụng nào đó.
- Mẫu thiết kế sẽ giúp cho việc giải quyết vấn đề nhanh, gọn và hợp lý hơn.
- Mẫu thiết kế còn được sử dụng nhằm cô lập các thay đổi trong mã nguồn, từ đó làm cho hệ thống có khả năng tái sử dụng cao.

Giới thiệu Design Pattern

❑ Lý do nên dung Design Pattern

- Tái sử dụng: Việc thiết kế một phần mềm hướng đối tượng phục vụ cho mục đích dùng lại là rất khó; cần phải xác định được có những lớp đối tượng nào, quan hệ giữa chúng ra sao, có kế thừa hay không,… Thiết kế phải đảm bảo không chỉ giải quyết được các vấn đề hiện tại, mà còn có thể tiến hành mở rộng trong tương lai. Vì vậy, nếu phần mềm không có một thiết kế tốt, việc sau này khi mở rộng phần mềm lại phải thiết kế lại từ đầu rất có thể xảy ra.

Giới thiệu Design Pattern

❑ Lý do nên dung Design Pattern

- Kinh nghiệm quý báu: Design Pattern là những kinh nghiệm đã được đúc kết từ những người đi trước, việc sử dụng Design Pattern sẽ giúp chúng ta giảm được thời gian và công sức suy nghĩ ra các giải pháp để giải quyết những vấn đề đã có lời giải.

Phân loại Design Pattern

- **Hệ thống các mẫu design pattern được chia thành 3 nhóm dựa theo vai trò**

Creational	Structural	Behavioral
Abstract Factory, Factory, Builder, Prototype, Singleton	Adapter, Bridge, Composite, Decorator, Facade, Proxy, Flyweight	Interpreter, Template Method, Chain of Responsibility, Command, Iterator, Mediator, Memento, Observer, State, Strategy, Visitor.

Phân loại Design Pattern

- **Trong đó:**
  - **Nhóm Creational (nhóm kiến tạo)**
    - Hỗ trợ cho việc khởi tạo đối tượng trong hệ thống: khởi tạo một đối tượng cụ thể từ một định nghĩa trừu tượng (abstract, class, interface).
    - Giúp khắc phục những vấn đề khởi tạo đối tượng, hạn chế sự phụ thuộc vào platform

Phân loại Design Pattern

- **Trong đó:**
  - **Nhóm Structural (nhóm cấu trúc)**
    - Các lớp đối tượng kết hợp với nhau tạo thành cấu trúc lớn hơn
    - Cung cấp cơ chế xử lý những lớp không thể thay đổi, ràng buộc muộn (late binding) và giảm kết nối (lower coupling) giữa các thành phần và cung cấp các cơ chế khác để kế thừa.
    - Diễn tả một cách có hiệu quả cả việc phân chia hoặc kết hợp các phần tử trong một ứng dụng

Phân loại Design Pattern

- **Trong đó:**
  - **Nhóm Behavioral (nhóm hành vi)**
    - Cách thức để các lớp và đối tượng có thể giao tiếp với nhau
    - Che giấu hiện thực của đối tượng, che giấu giải thuật, hỗ trợ việc thay đổi cấu hình đối tượng một cách linh động.
    - Có liên quan đến luồng điều khiển của hệ thống. Một vài cách của tổ chức điều khiển bên trong một hệ thống có thể mang lại các lợi ích cả về hiệu suất lẫn khả năng bảo trì hệ thống đó

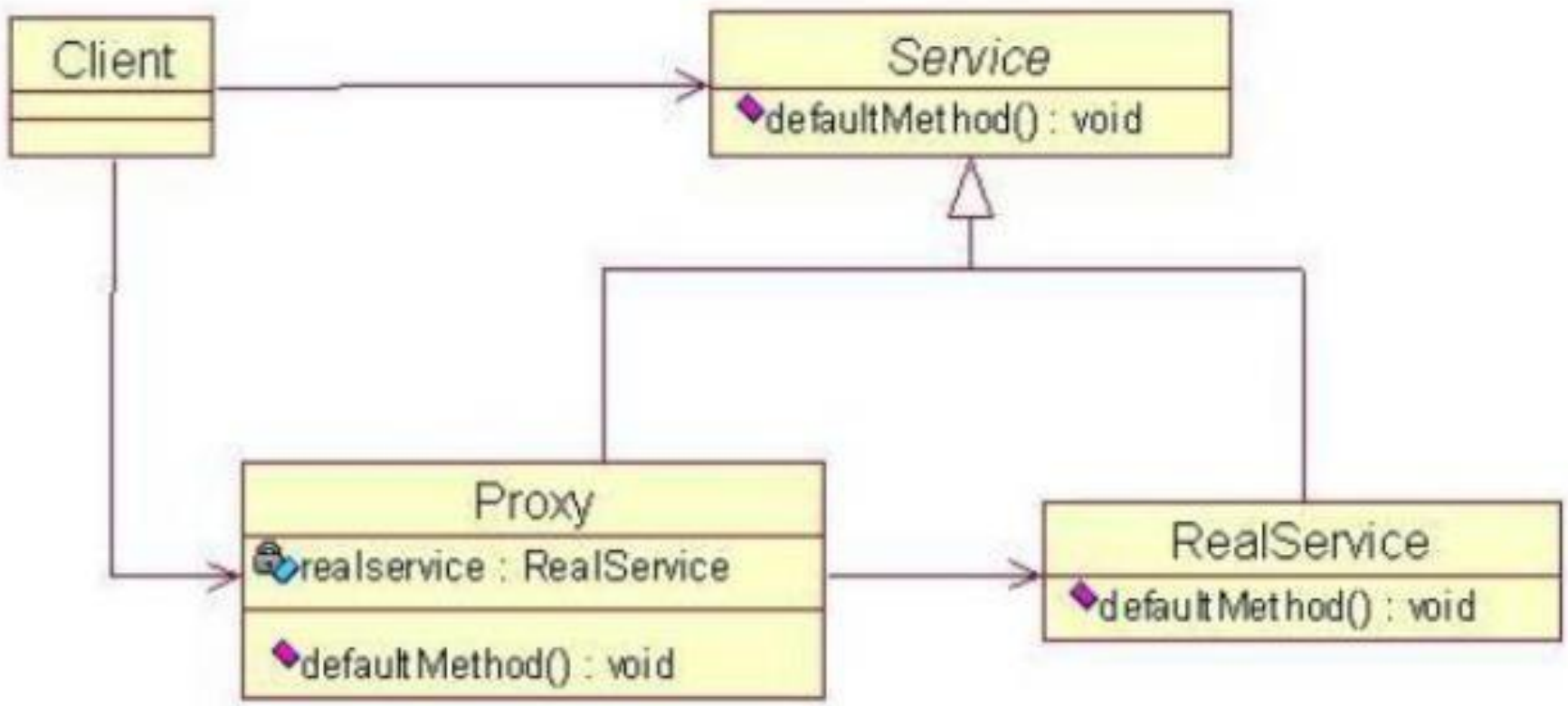
Proxy

❑ Giới thiệu

- Đại diện một đối tượng phức tạp bằng một đối tượng đơn giản, vì các mục đích truy xuất, tốc độ và bảo mật
- Là một mẫu thiết kế mà ở đó tất cả các truy cập trực tiếp một đối tượng nào đó sẽ được chuyển hướng vào một đối tượng trung gian

Proxy

❑ Cấu trúc mẫu



Proxy

```
public interface Service {
    void defaultMethod();
}
```

```
public class RealService
implements Service {
    // khởi tạo RealService
    public RealService(…){
    …
    }
    @Override
    public void defaultMethod() {
    // code của việc sẽ thực hiện
    trong RealService
    }
    …
}
```

```
public class Proxy implements
Service{
    private RealService
    realService;
    public Proxy (…){
    …
    }
    @Override
    public void defaultMethod() {
    if(realService == null){
        realService = new
        RealService(…);
    }
    realService.defaultMethod ();
    }
}
```

Proxy

- Trong đó:
  - **Service**: là giao tiếp định nghĩa các phương thức chuẩn cho một dịch vụ nào đó
  - RealService: là một thực thi của giao tiếp Service, lớp này sẽ khai báo tường minh các phương thức của Service, lớp này xem như thực hiện tốt tất cả các yêu cầu từ Service
  - **Proxy**: kế thừa Service và sử dụng đối tượng của RealService

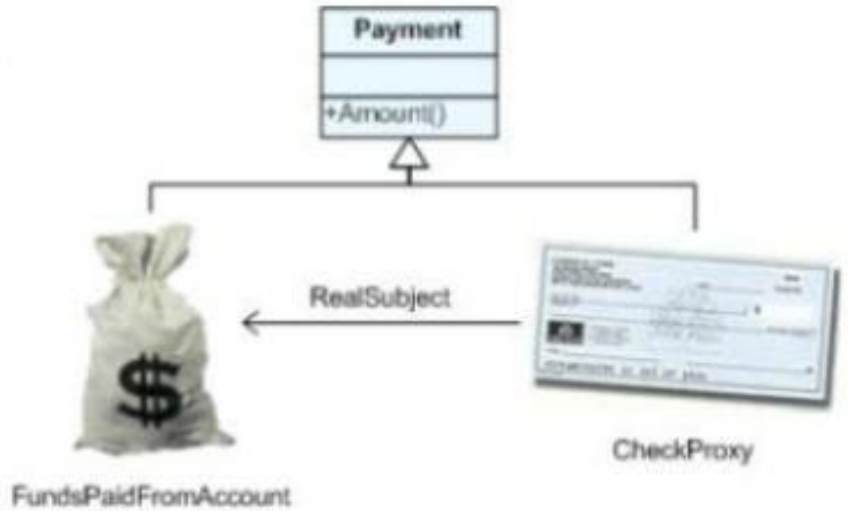
Proxy

❑ Áp dụng

- Sử dụng mẫu Proxy khi bạn cần một tham chiếu phức tạp đến một đối tượng thay vì chỉ một cách bình thường
- Remote proxy – sử dụng khi bạn cần một tham chiếu định vị cho một đối tượng trong không gian địa chỉ (JVM)
- Virtual proxy – lưu giữ các thông tin thêm vào về một dịch vụ thực vì vậy chúng ta có thể hoãn lại sự truy xuất vào dịch vụ này
- Protection proxy – xác thực quyền truy xuất vào một đối tượng thực

Proxy

- **Ví dụ:** Proxy cung cấp một vật thay thế hoặc một nơi lưu trữ để cung cấp quyền truy cập vào một đối tượng. Một tấm séc hay ngân phiếu là một proxy cho các quỹ trong một tài khoản. Tấm séc có thể được sử dụng thay cho tiền mặt khi mua sắm và kiểm soát truy cập tài khoản của người phát hành



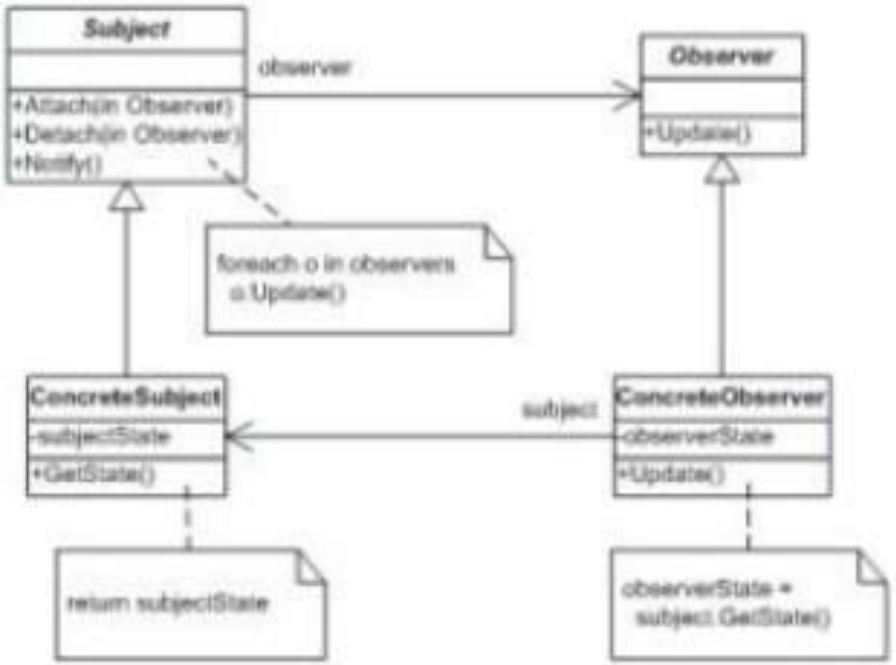
Observer

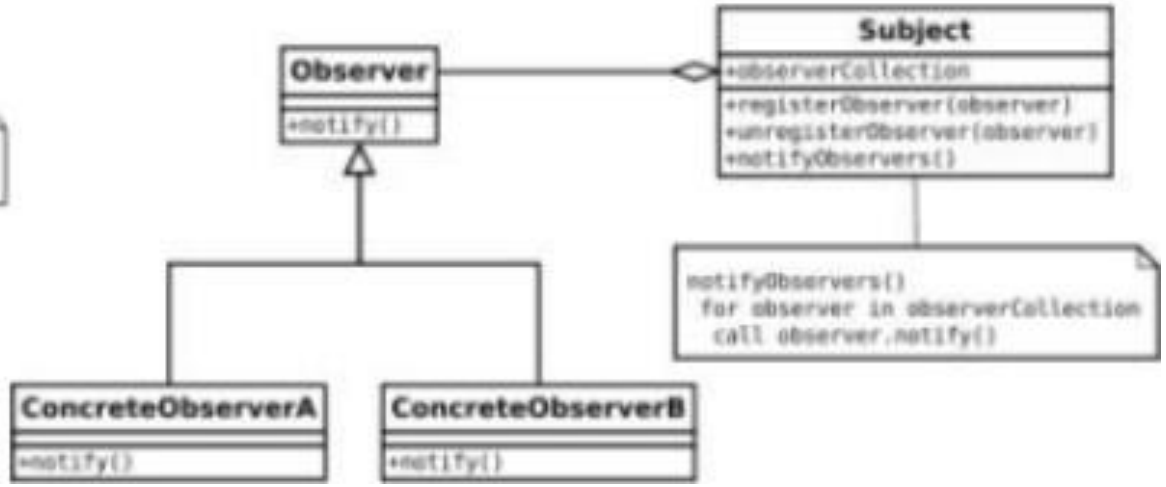
❑ Giới thiệu

- Observer là một pattern được dung trong trường hợp ta muốn cài đặt những lớp – đối tượng phụ thuộc vào đối tượng khác, gọi là các đối tượng quan sát(observer) và đối tượng được quan sát (observable).
- Khi trạng thái của đối tượng được quan sát thay đổi thì những đối tượng quan sát sẽ thực hiện hành động nào đó.
- Pattern này cũng được dung khá phổ biến.

Observer

❑ Cấu trúc mẫu





Observer

- **Trong đó**
  - Observable – interface hoặc abstract class xác định các hoạt động để gắn hoặc gắn lại observer cho client, còn được gọi là Account
  - ConcreteObservable – concrete Observable class. Nó duy trì trạng thái của đối tượng, khi có một thay đổi trong trạng thái xuất hiện, nó sẽ thông báo cho các Observer khác kèm theo
  - Observer – interface hoặc abstract class định nghĩa các operation được sử dụng để thông báo cho đối tượng này.
  - ConcreteObserverA, ConcreteObserverB – lớp cụ thể được kế thừa từ lớp Observer

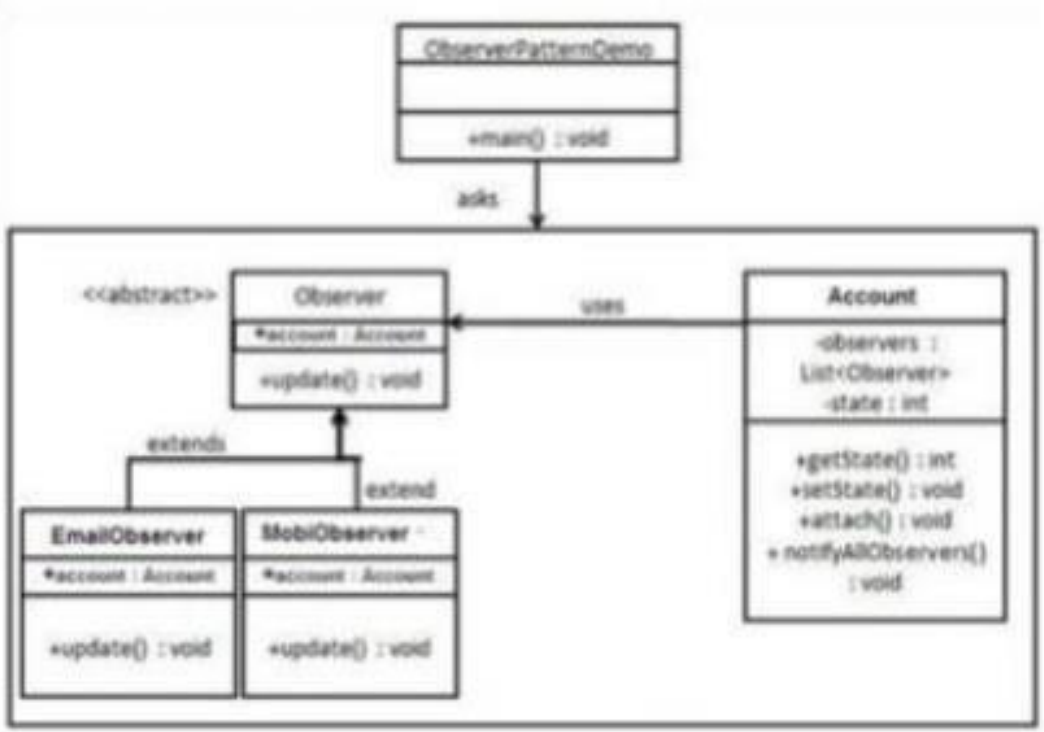
Observer

❑ Áp dụng

- Khi một trừu tượng (abstraction) có hai phía cạnh (aspect), cái nọ phụ thuộc vào cái kia
- Đóng gói các khía cạnh này trong các đối tượng riêng biệt cho phép thay đổi và tái sử dụng chúng một cách độc lập
- Khi có một thay đổi được đến một đối tượng yêu cầu thay đổi các đối tượng khác.

Observer

- Ví dụ: Đối tượng được quan sát sẽ là đối tượng Account, khi nó có thay đổi trạng thái thì các đối tượng quan sát là MobiPhone và Email sẽ thực hiện hành động tương ứng là gửi SMS và gửi email thông báo



Observer

```
public class Account {
	private List<Observer> observers = new ArrayList<Observer>();
	private int state;
	public int getState() {
		return state;
	}
	public void setState(int state) {
		this.state = state;
		notifyAllObservers();
	}
	public void attach(Observer observer){
		observers.add(observer);
	}
	public void notifyAllObservers(){
		for (Observer observer : observers) {
		observer.update();
		}
	}
}
```

Observer

```
public abstract class Observer {
    protected Account account;
    public abstract void update();
}
```

```
public class EmailObserver extends
Observer{
    public EmailObserver(Account
    account){
    this.account = account;
    this.account.attach(this);
    }
    @Override
    public void update() {
    // nội dung của Email
    }
}
```

```
public class MobiObserver extends
Observer{
    public MobiObserver(Account
    account){
    this.account = account;
    this.account.attach(this);
    }
    @Override
    public void update() {
    // nội dung của Mobi
    }
}
```

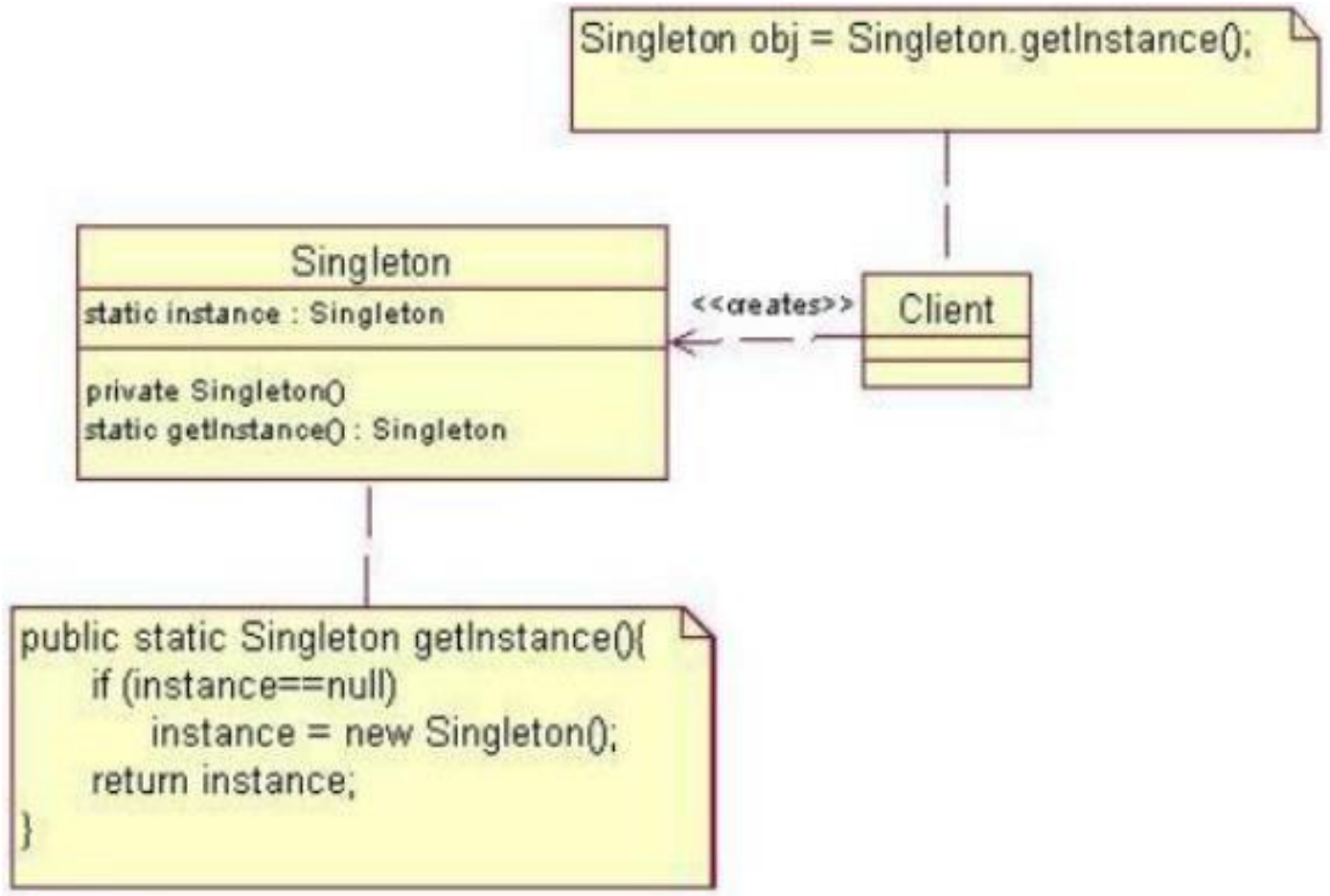
Singleton

❑ Giới thiệu

- Mẫu này được thiết kế để đảm bảo cho một lớp chỉ có thể tạo duy nhất một thể hiện của nó

Singleton

❑ Cấu trúc mẫu



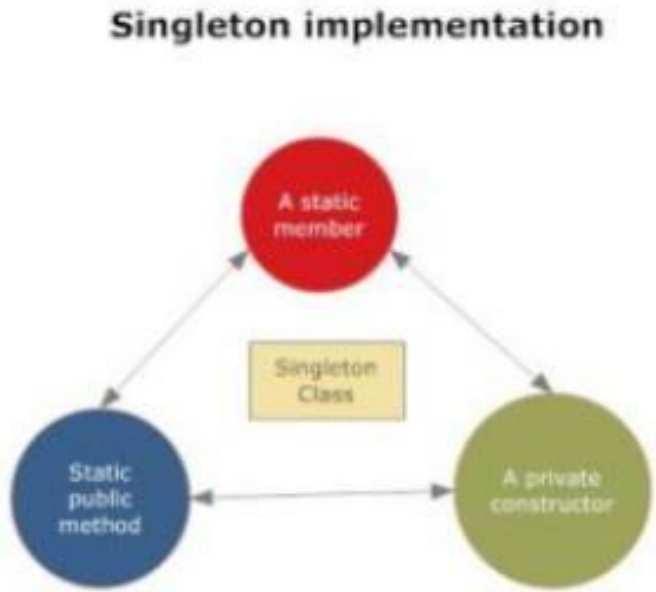
Singleton

```
public class Singleton {
    private static Singleton instance = new Singleton();
    private Singleton(){}
    public static Singleton getInstance(){
        if (instance == null)
         instance = new Sinleton
        return instance;
    }
    ...
}
```

```
//cilent
SingleObject object = SingleObject.getInstance();
...
```

Singleton

- Trong đó
  - Singleton cung cấp một phương thức tạo private , duy trì một thuộc tính tĩnh để tham chiếu đến một thể hiện của lớp Singleton này, và nó cung cấp thêm một phương thức tĩnh trả về thuộc tính tĩnh này

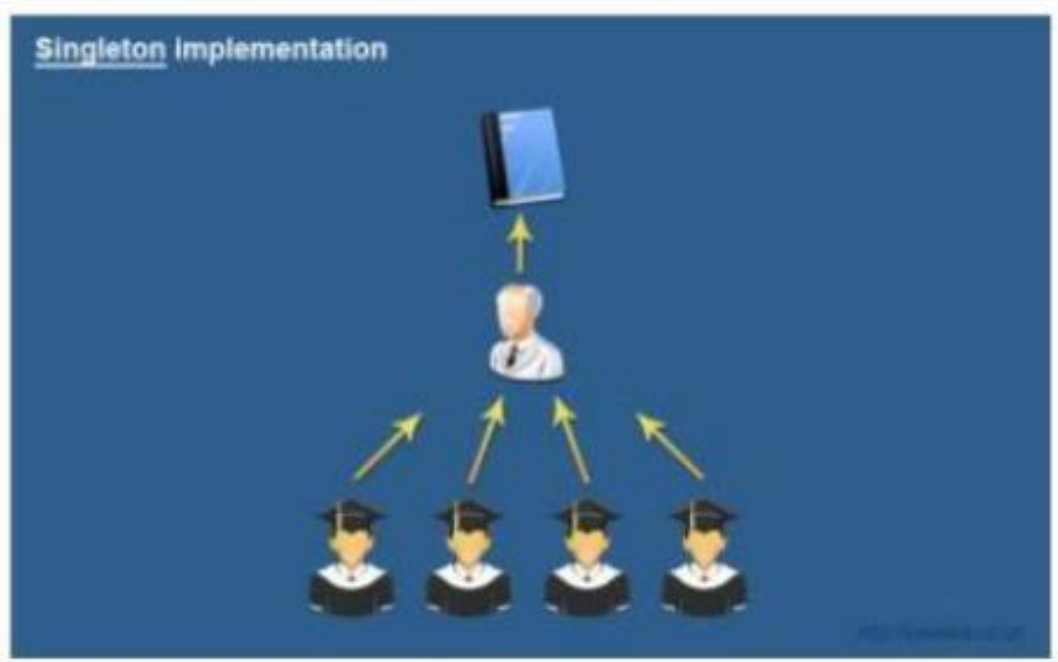


Singleton

- **Áp dụng**
  - Khi muốn lớp chỉ có 1 thể hiện duy nhất và nó có hiệu lực ở mọi nơi

Singleton

- **Ví dụ:**
  - Bằng cách sử dụng mẫu Singleton, lớp Teacher đã được khởi tạo chỉ một lần và sau đó mỗi đối tượng Student nhận được một tài liệu tham khảo của instance đó



Q&A


T3H
IT-Institute
THANK YOU