




ĐẠI HỌC BÁCH KHOA HÀ NỘI  
 VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

# Bài 12: Mẫu thiết kế Design patterns

1

## Nội dung

1. Giới thiệu
2. Mẫu thiết kế Singleton
3. Mẫu thiết kế Factory Method




VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

2

2

## Nội dung

1. Giới thiệu
2. Mẫu thiết kế Singleton
3. Mẫu thiết kế Factory Method




VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

3

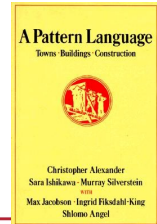
3

## 1. Giới thiệu

- ❖ Vào cuối những năm 70, kiến trúc sư Christopher Alexander đưa khái niệm về pattern (mẫu): mẫu các giải pháp cho một vấn đề cụ thể
- ❖ Christopher Alexander là một kiến trúc sư, các mẫu thiết kế của ông liên quan đến kiến trúc tòa nhà. Đây là ý tưởng cho các mẫu thiết kế phần mềm



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



4

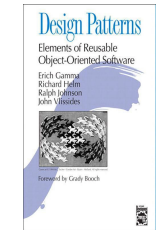
4

## 1. Giới thiệu

- ❖ Mẫu thiết kế - Design patterns là các giải pháp tối ưu nhất, đưa ra bởi các LTV OOP giàu kinh nghiệm
- ❖ Mẫu thiết kế là giải pháp chuẩn cho các vấn đề LTV thường gặp trong quá trình phát triển phần mềm
  - Giải pháp có được sau quá trình tối ưu mã nguồn trong suốt thời gian dài
  - Giúp nâng cao chất lượng mã nguồn, đảm bảo tính high cohesion, low coupling

## Gang of Four (GOF)

- ❖ Năm 1994, 4 tác giả Erich Gamma, Richard Helm, Ralph Johnson và John Vlissides xuất bản cuốn sách có tiêu đề **Design Patterns - Elements of Reusable Object-Oriented Software**, chính thức đề xuất khái niệm mẫu thiết kế - Design Pattern trong phát triển phần mềm



## GoF patterns: 3 nhóm

- ❖ **Nhóm khởi tạo (Creational Patterns)** – trừu tượng hóa quá trình khởi tạo đối tượng
  - Factory Method, Abstract Factory, Singleton, Builder, Prototype
- ❖ **Nhóm cấu trúc (Structural Patterns)** – kết hợp các đối tượng
  - Adapter, Bridge, Composite, Decorator, Façade, Flyweight, Proxy
- ❖ **Nhóm hành vi (Behavioral Patterns)** – các đối tượng giao tiếp với nhau
  - Command, Interpreter, Iterator, Mediator, Observer, State, Strategy, Chain of Responsibility, Visitor, Template Method

## Phân loại mẫu thiết kế

	Purpose	Creation	Structure	Behaviour
Scope				
Class		Factory method	Adapter (class)	Interpreter, Template Method
Object		Abstract Factory	Adapter (object)	Chain of Responsibility
		Builder	Bridge	Command
		Prototype	Composite	Iterator
		Singleton	Decorator	Mediator
			Façade	Memento
			Flyweight	Observer
			Proxy	State, Strategy, Visitor

## Cách các Design Pattern được giới thiệu

- ❖ Name – Alias: Tên, tên gọi khác
- ❖ Classification: Phân loại
- ❖ Intent: Mục đích
- ❖ Motivation: Khi nào cần sử dụng mẫu này
  - Bài toán đặt ra
  - Giải pháp nếu không dùng DP (nếu có)
- ❖ Solution: Giải pháp khi dùng DP (ví dụ và tổng quát)
  - Biểu đồ lớp / Biểu đồ tương tác
  - Mã nguồn minh họa
- ❖ Pros and cons
  - Phân tích ưu nhược điểm khi sử dụng DP này
- ❖ Applicability
  - Các ví dụ ứng dụng trong thực tế, đặc biệt những ví dụ phổ biến

9

## Nội dung

1. Giới thiệu
2. **Mẫu thiết kế Singleton**
3. Mẫu thiết kế Factory Method

10

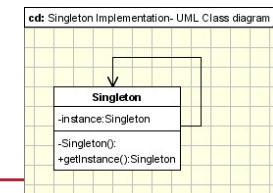
## Mục tiêu

- ❖ Yêu cầu đặt ra: chỉ có **duy nhất một** thể hiện của một lớp được tạo ra?
  - Cần đảm bảo không thể tạo ra được thể hiện thứ hai
- ❖ Quản lý tập trung các tài nguyên của hệ thống: cung cấp điểm truy cập toàn cục tới thể hiện duy nhất đó

11

## Giải pháp

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



12

### “Lazy instantiation” với cơ chế khóa kép

```
class Singleton {
    private static Singleton instance;

    private Singleton(){
        System.out.println("Singleton(): Initializing Instance");
    }

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

    public void doSomething(){
        System.out.println("doSomething(): Singleton does something!");
    }
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

13

13

### “Early instantiation” với trường static

```
class Singleton{
    private static Singleton instance = new Singleton();

    private Singleton(){
        System.out.println("Singleton(): Initializing Instance");
    }

    public static Singleton getInstance(){
        return instance;
    }

    public void doSomething(){
        // ...
    }
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

14

14

### Nội dung

1. Giới thiệu
2. Mẫu thiết kế Singleton
3. **Mẫu thiết kế Factory Method**



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

15

15

### Mục tiêu

- ❖ Khởi tạo các đối tượng mà không để lộ logic khởi tạo cho client.
- ❖ Tham chiếu đến đối tượng vừa được tạo thông qua một giao diện chung

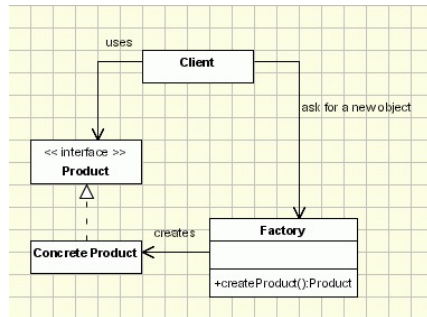


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

16

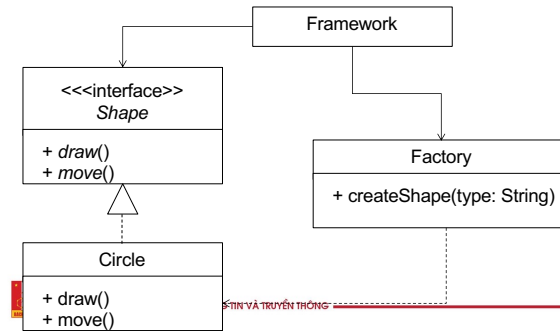
16

## Giải pháp



## Ví dụ: framework vẽ hình với các đối tượng shapes

- ❖ Client: drawing framework
- ❖ Product: Lớp Shape với 2 phương thức draw() và move()

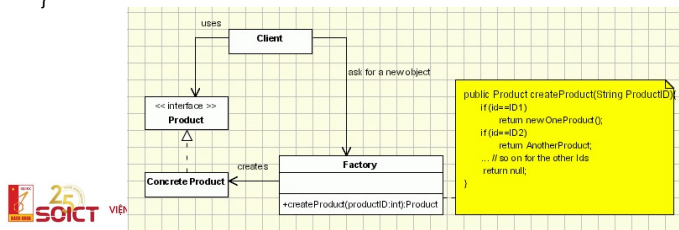


## Switch/case “noob instantiation”

```

public class ProductFactory{
    public Product createProduct(String ProductID){
        if (id==ID1)
            return new OneProduct();
        if (id==ID2) return
            return new AnotherProduct();
        ... // so on for the other ids
    }
    return null; //if the id doesn't have any of the expected values
}
...

```



## Class Registration – Sử dụng Java reflection

```

class ProductFactory {
    private HashMap m_RegisteredProducts = new HashMap();
    public void registerProduct (String productID, Class productClass){
        m_RegisteredProducts.put(productID, productClass);
    }
    public Product createProduct(String productID){
        Class productClass = (Class)m_RegisteredProducts.get(productID);
        Constructor productConstructor = productClass.
            getDeclaredConstructor(new Class[] {String.class});
        return (Product)productConstructor.newInstance(new Object[] { });
    }
}

public static void main(String args[]){
    ProductFactory.instance().registerProduct("ID1", OneProduct.class);
}

class OneProduct implements Product{
    static {
        ProductFactory.instance().registerProduct("ID1",OneProduct.class);
    }
}

```

### Do dùng static block: Cần load các lớp Product

```
class Main {
    static {
        try {
            Class.forName("OneProduct");
            Class.forName("AnotherProduct");
        }
        catch (ClassNotFoundException any){
            any.printStackTrace();
        }
    }
    public static void main(String args[]) {
        ...
    }
}
```

21

### Class Registration – giải pháp không dùng Java reflection

```
abstract class Product {
    public abstract Product createProduct();
    ...
}

class OneProduct extends Product {
    ...
    static {
        ProductFactory.instance().registerProduct("ID1", new OneProduct());
    }
    public OneProduct createProduct() {
        return new OneProduct();
    }
    ...
}

class ProductFactory {
    private HashMap m_RegisteredProducts = new HashMap();

    public void registerProduct(String productID, Product p){
        m_RegisteredProducts.put(productID, p);
    }

    public Product createProduct(String productID){
        ((Product)m_RegisteredProducts.get(productID)).createProduct();
    }
}
```

22