Bài 14
SOLID và Creational Design Pattern

Môn học: WBD-JAVA

Mục tiêu

- Trình bày được ý nghĩa của nguyên lý SOLID
- Trình bày được ý nghĩa của Design Pattern
- Trình bày được ý nghĩa của các Creational Design Pattern
- Triển khai được Singleton Pattern
- Triển khai được Factory Method Pattern
- Triển khai được Object Pool Pattern

Program to an interface, not an concrete implementation

S.O.L.I.D

- SOLID là viết tắt của các từ:
  - **S** - Single-responsiblity principle – Nguyên lý Trách nhiệm Duy nhất
  - **O** - Open-closed principle – Nguyên lý đóng mở
  - **L** - Liskov substitution principle – Nguyên lý Thay thế Liskov
  - **I** - Interface segregation principle – Nguyên lý Phân tách Interface
  - **D** - Dependency Inversion Principle – Nguyên lý Đảo ngược Phụ thuộc
- SOLID bao gồm các nguyên lý quan trọng bậc nhất cần tuân thủ khi thiết kế kiến trúc phần mềm nhằm đạt được mục đích:
  - Dễ mở rộng
  - Dễ bảo trì

Single-responsiblity principle

- Nguyên lý Trách nhiệm Duy nhất
- Mỗi lớp chỉ nên đảm nhiệm một nhiệm vụ duy nhất
- Lí do:
  - Dễ quản lý mã nguồn
  - Các lớp tập trung vào nhiệm vụ của mình
  - Giảm tính phụ thuộc giữa các thành phần
  - Có thể phát triển đồng thời các lớp độc lập với nhau
  - Dễ dàng mở rộng
  - Dễ dàng bảo trì

Open-closed principle

- Nguyên lý Đóng-mở
- Các đối tượng (hoặc thực thể) nên mở đối với việc mở rộng, nhưng đóng đối với việc thay đổi
- Nghĩa là: Có thể dễ dàng mở rộng một lớp mà không cần thay đổi mã nguồn của lớp đó
- Lí do:
  - Dễ mở rộng
  - Dễ thay đổi

Liskov substitution principle

- Nguyên lý Thay thế Liskov
- Các lớp con có thể được sử dụng thay thế cho các lớp cha
- Nghĩa là: Nếu S là một lớp con của T, thì các đối tượng của lớp T có thể được thay thế bằng các đối tượng của lớp S mà không làm ảnh hưởng tới bất cứ hành vi nào của hệ thống
- Lí do:
  - Tránh các sai sót khi mở rộng thiết kế

Interface segregation principle

- Nguyên lý Phân tách Interface
- Không nên bắt buộc phải triển khai một Interface nếu không cần đến nó, cũng không nên bắt buộc phải phụ thuộc vào các phương thức mà không cần đến chúng
- Nghĩa là: Các Interface chỉ nên chứa các phương thức cần thiết vừa đủ với mục đích của nó. Nên tách các Interface thành nhiều Interface nếu các phương thức của chúng không liên quan chặt chẽ đến nhau.

Dependency Inversion Principle

- Nguyên lý Đảo ngược Phụ thuộc
- Trừu tượng (abstraction) không nên phụ thuộc vào chi tiết. Chi tiết nên phụ thuộc vào trừu tượng.
- Các module ở mức trên không nên phụ thuộc vào các module ở mức dưới, mà cả hai nên phụ thuộc vào Trừu tượng (abstraction)

Design Pattern

- Design Pattern (DP - Mẫu Thiết kế) là các giải pháp tổng quát có thể tái sử dụng cho các trường hợp thường gặp khi thiết kế kiến trúc phần mềm.
- Design Pattern không phải là bản thiết kế hoàn chỉnh có thể dùng để chuyển hoá trực tiếp thành mã nguồn.
- Design Pattern là các khuôn mẫu (template) để giải quyết vấn đề trong các tình huống khác nhau.

Lợi ích của Design Pattern

- Đẩy nhanh tốc độ thiết kế và phát triển phần mềm
- Chất lượng của giải pháp đã được minh chứng
- Ngăn ngừa các vấn đề phát sinh nếu thiết kế không tốt
- Có thể áp dụng cho rất nhiều tình huống khác nhau
- Dễ dàng cộng tác, chia sẻ thiết kế và mã nguồn giữa các bên

Phân nhóm Design Pattern

- Creational Design Pattern:
  - Là nhóm các Design Pattern được sử dụng để giải quyết các vấn đề thường gặp đối với việc khởi tạo đối tượng.
- Structural Design Pattern:
  - Là nhóm các Design Pattern được sử dụng để thiết kế các thành phần của lớp và đối tượng giúp việc dễ dàng nhận ra mối quan hệ giữa các thực thể.
- Behavioral Design Pattern:
  - Là nhóm các Design Pattern được sử dụng để giải quyết các vấn đề phổ biến trong giao tiếp giữa các đối tượng.

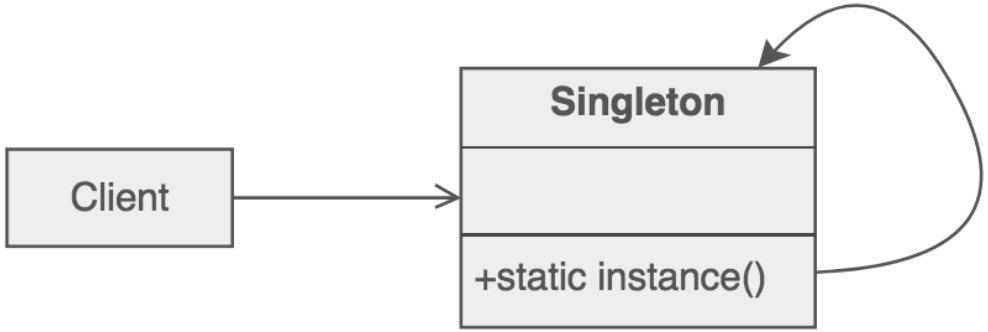
Các Creational Design Pattern thông dụng

- Singleton
  - Giải quyết vấn đề: Ứng dụng chỉ cần duy nhất đối tượng của một lớp nào đó.
- Factory Method
  - Giải quyết vấn đề: Các ứng dụng riêng lẻ xác định các đối tượng miền của riêng chúng và cung cấp sự khởi tạo những đối tượng này.
- Object Pool
  - Giải quyết vấn đề: Giới hạn số lượng các đối tượng và tái sử dụng chúng thay vì phải tạo ra quá nhiều đối tượng khiến tốn kém tài nguyên.

Singleton

- Singleton là một mẫu thiết kế thuộc nhóm creational cho phép bạn đảm bảo rằng một lớp chỉ có một đối tượng thể hiện và cung cấp truy cập đối tượng này với phạm vi toàn ứng dụng.
- Singleton đảm bảo chỉ duy nhất môt thể hiện mới (new instance) được tạo ra và nó sẽ cung cấp cho bạn một phương thức để truy cập đến thực thể đó.
- Một lớp thiết kế theo Singleton Pattern có các đặc điểm:
  - Phương thức khởi tạo private để ngăn cản việc tạo thể hiện của class từ các lớp khác.
  - Biến private static của class, nó là thể hiện duy nhất của lớp.
  - Có một phương thức public static để trả về thể hiện của lớp.

Singleton - 2

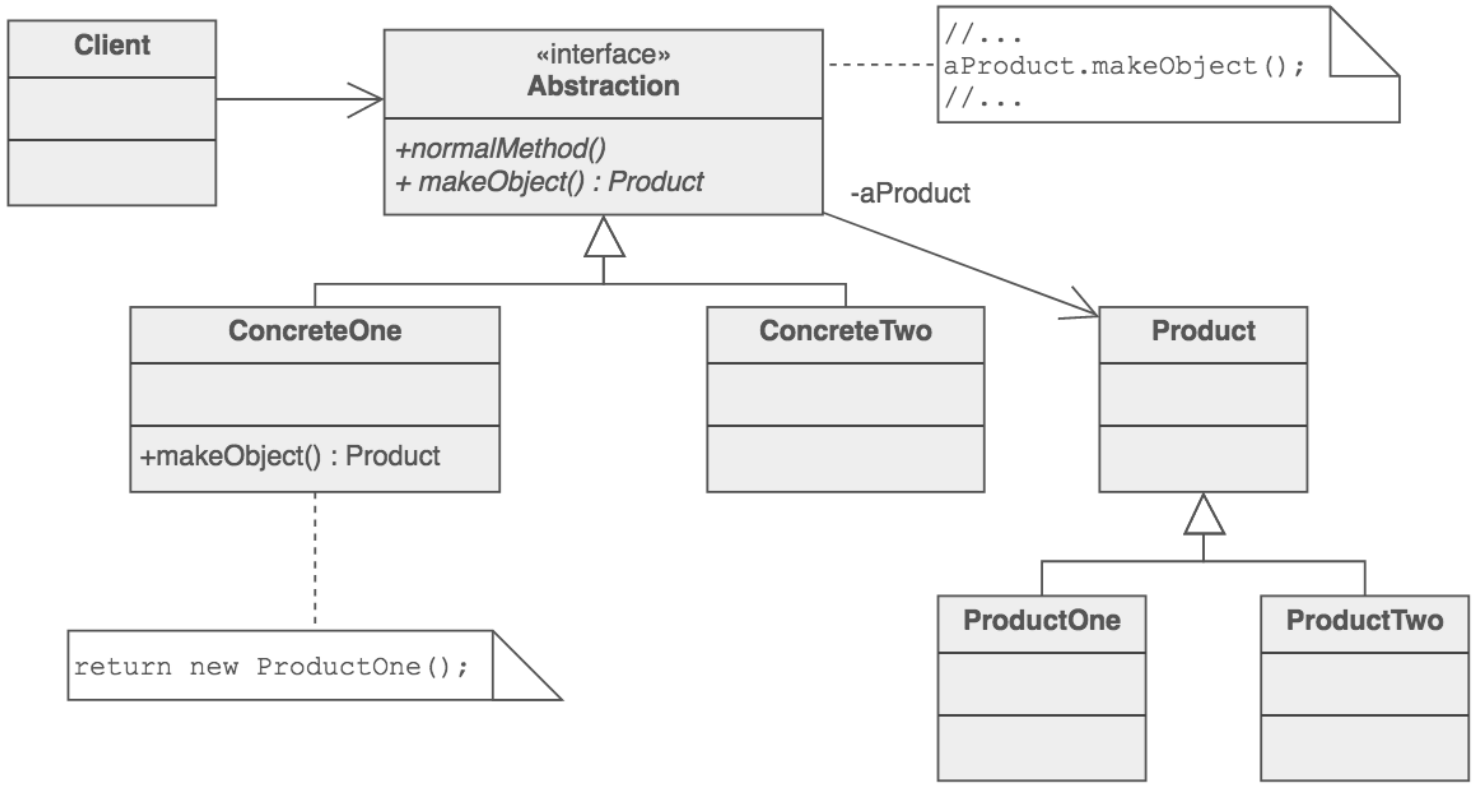



```
public final class Singleton {
    private static volatile Singleton instance;

    private Singleton() {}

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

Factory Method

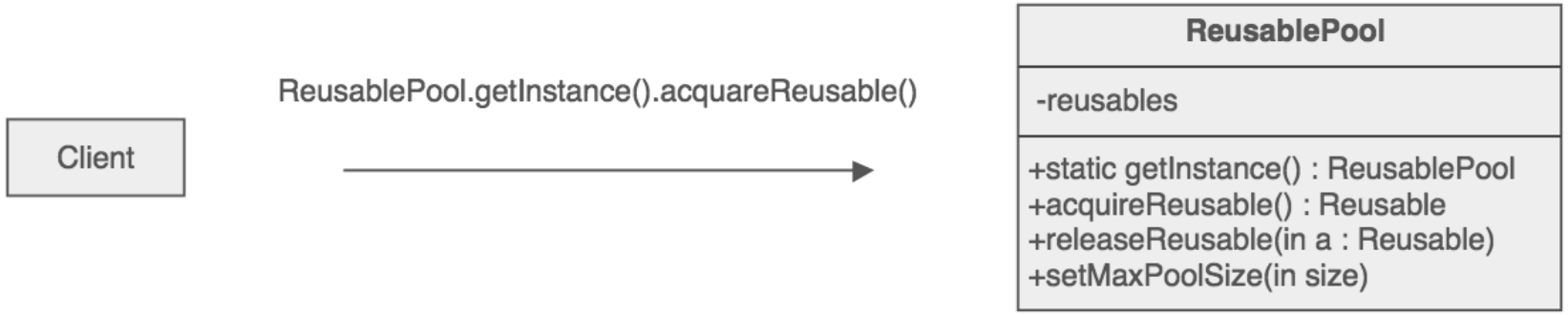
- Đây là mẫu thiết kể có thể xem là được sử dụng phổ biến nhất, với mục đích quản lý và khởi tạo các đối tượng một cách linh hoạt.
- Factory Method định nghĩa một interface dành cho việc tạo ra đối tượng, nhưng để lớp con quyết định lớp nào sẽ được dùng để khởi tạo đối tượng.
- Factory Method tương tự Abstract Factory nhưng khác với Factory Method, Abstract Factory là factory của các factory.

Factory Method - 2



Object Pool

- Object Pool Pattern quản lý một tập hợp các đối tượng sẽ được tái sử dụng trong chương trình.
- Các đối tượng được chứa chung ở một nơi gọi là pool (bể chứa). Khi cần dung, đối tượng được gọi ra từ pool, sử dụng trong một khoảng thời gian nhất định rồi trả về pool. Không thành phần nào có thể sử dụng tận được đối tượng được lấy ra khỏi pool cho tới khi nó được quay trả về.
- Object Pool Pattern thích hợp với tình huống khi có nhiều hơn một đối tượng và số đối tượng được khởi tạo là hạn chế.



[Thực hành] Triển khai Singleton Pattern

[Thực hành] Triển khai Factory Method

[Thực hành] Triển khai Object Pool

[Bài tập] Áp dụng Factory Method

Tổng kết

- SOLID bao gồm các nguyên lý quan trọng bậc nhất cần tuân thủ khi thiết kế kiến trúc phần mềm nhằm đạt được mục đích: Dễ mở rộng và Dễ bảo trì
- Design Pattern (Mẫu Thiết kế) là các giải pháp tổng quát có thể tái sử dụng được trong các trường hợp thường gặp khi thiết kế kiến trúc phần mềm
- Singleton chỉ khởi tạo một đối tượng duy nhất của một lớp
- Factory Method định nghĩa một Interface để khởi tạo đối tượng, nhưng để cho các lớp con quyết định sẽ khởi tạo đối tượng của lớp nào.
- Object Pool nâng cao hiệu năng của ứng dụng thông qua việc tạo ra một tập hạn chế các đối tượng và tái sử dụng chúng thay vì phải tạo ra quá nhiều đối tượng