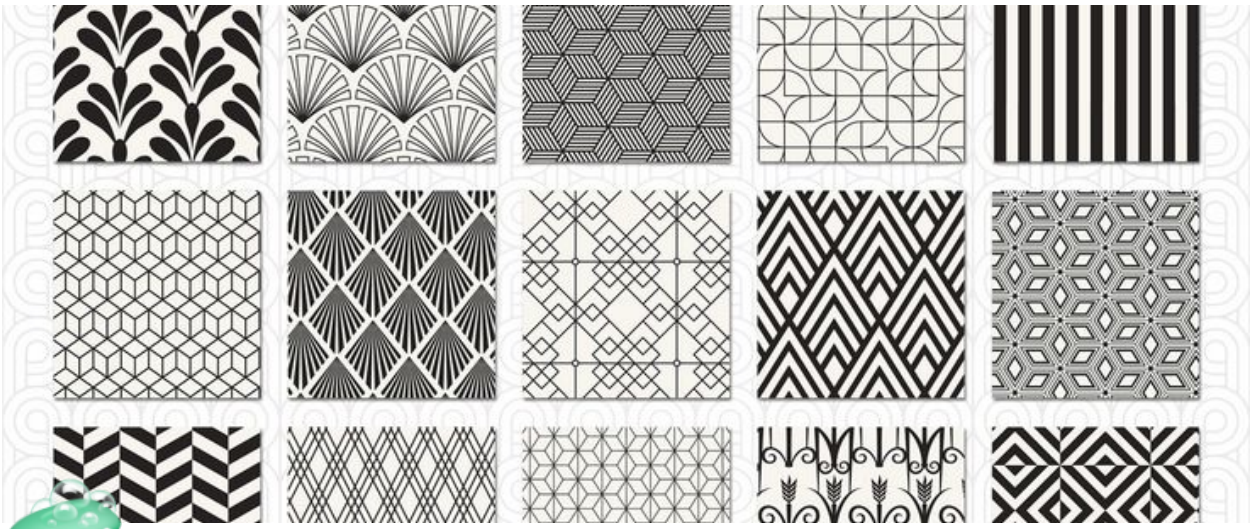

디자인 패턴 - 프록시 패턴

주차	18주차
스터디 일자	@2024/02/29

프록시 패턴 (Proxy Pattern)?

> *프록시 패턴은 객체 지향 디자인 패턴 중 하나로, 객체 간의 간접적인 접근을 가능하게 하는 구조를 제공하는 패턴이다.*

프록시란?

대변인 이라는 의미를 가지고 있어, **프록시 패턴**은 말 그대로 원본 객체를 바로 호출하는 것이 아니라, 원본 객체에 접근할 수 있는 **대리자**를 호출하는 패턴이다.

프록시 패턴을 사용한다면

→ 어떤 객체를 호출할 때, 해당 객체가 메모리에 존재하지 않아도 기본적인 정보를 참조하거나 설정할 수 있다.

→ 실제 객체의 필요 시점까지 객체 생성을 미루는 **지연 초기화(Lazy Initializing)**가 가능하다.

프록시 패턴 종류

원격 프록시

원격 객체에 대한 대변자 역할을 하여 다른 공간에 있더라도 마치 같은 공간에 있는 객체에 접근한 것처럼 동작한다.

가상 프록시 (지연 초기화 방식)

꼭 필요로 하는 시점까지 객체의 생성을 연기하고, 해당 객체가 생성된 것 처럼 동작한다.

보호 프록시

객체에 대한 접근을 제어하거나 권한을 달리 하고 싶은 경우 사용한다.

프록시 패턴 장단점

장점

SOLID 원칙 준수

기존 대상 객체의 코드를 변경하지 않고 새로운 기능을 추가할 수 있다. (**개방 폐쇄 원칙-OCP)**

대상 객체는 자신의 기능에만 집중 하고, 그 이외 부가 기능을 제공하는 역할을 프록시 객체에 위임하여 다중 책임을 회피 할 수 있다. (**단일 책임 원칙-SRP**)

보안성 향상

원본 객체에 직접 접근하지 않기 때문에 보안성이 향상된다.

프록시 객체를 사용하여 원본 객체 접근 권한을 제한할 수 있다.

접근 로그를 남기는 등 보안적인 추가 기능을 적용할 수 있다.

유연성 향상

원본 객체에 간접적으로 접근하므로, 객체 간의 결합도를 줄일 수 있다.

성능 향상

지연 초기화를 통해 객체가 필요한 순간에 필요한 객체만 초기화하여 사용하므로, 불필요한 객체 생성을 줄일 수 있다.

단점

복잡성 증가

- 객체 간의 중간 계층이 추가되어 코드의 복잡성이 증가한다.

성능 저하

- 프록시 객체를 추가로 사용하므로, 객체 생성이 빈번한 경우 성능이 저하될 수 있다.

프록시 패턴 구현 방식

- 인터페이스

```
public interface Image {
    void display();
}
```

- 실제 구현 클래스

```
public class RealImage implements Image {
    private String filename;

    public RealImage(String filename) {
        this.filename = filename;
        loadFromDisk(); // 파일을 로딩중임을 알리는 메서드
    }

    @Override
    public void display() {
        System.out.println("Displaying " + filename);
    }
```

```
    private void loadFromDisk() {
        System.out.println("Loading " + filename);
    }
}
```

- 프록시 클래스

```
public class ProxyImage implements Image {
    private RealImage realImage;
    private String filename;

    public ProxyImage(String filename) {
        this.filename = filename;
    }

    @Override
    public void display() {
        if (realImage == null) { // 로딩해논 이미지 없으면 실제 객
체 만듦
            realImage = new RealImage(filename);
        }
        realImage.display();
    } // 지연로딩의 원리
```

- 클라이언트

```
public class Client {
    public static void main(String[] args) {
        Image image = new ProxyImage("test.jpg");

        // 이미지를 표시
        image.display();

        // 이미지를 다시 표시함
        // 이번에는 이미지를 로드 X
```

```
            image.display();
        }
    }
```

⇒ 실제 구현 클래스에 직접 접근하지 않고 **대리자**를 통해 동작을 수행하고 있다!

☝️ 프록시는 인터페이스를 구현함으로써 생성되는 방식이기 때문에, 자바의 동적 프록시는 클래스가 하나 이상의 **인터페이스를 구현할 때**만 동작한다 ~~!