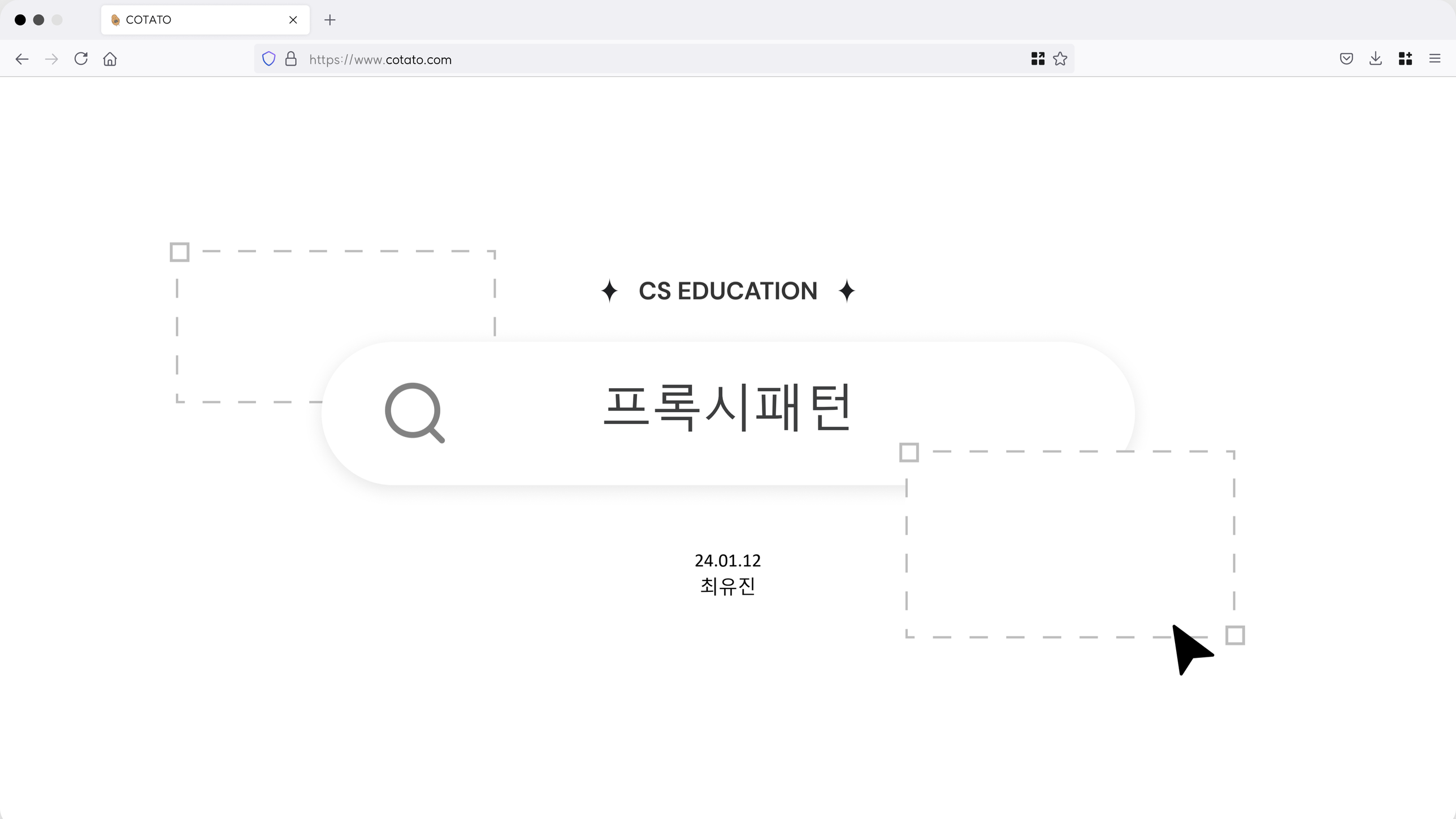

COTATO
https://www.cotato.com
CS EDUCATION
프록시패턴
24.01.12
최유진

교육 목표

-프록시패턴
-프록시패턴의 로직
-프록시패턴의 구조
-프록시패턴의 장단점

디자인패턴

:특정 문맥과 상황에서 반복해서 일어나는 문제에 대한 해결방안을 정리한 것
-> **보편적인 문제상황**에서의 **해결방안**

모든 디자인패턴은 어떠한 보편적인 문제와 이 문제에 대한 해결책을 중심으로 이루어짐

디자인패턴의 종류

GoF의 디자인패턴 -> 생성 패턴, 구조 패턴, 행위 패턴

생성(Creational) 패턴	구조(Structural) 패턴	행동(Behavioral) 패턴
Singleton	Adapter	Command
Abstract Factory	Composite	Interpreter
Factory Method	Decorator	Iterator
Builder	Facade	Mediator
Prototype	Flyweight	Memento
	Proxy	Observer
		State
		Strategy
		Template Method

모든 디자인패턴의 이름은 해당 패턴의 솔루션을 담고 있음

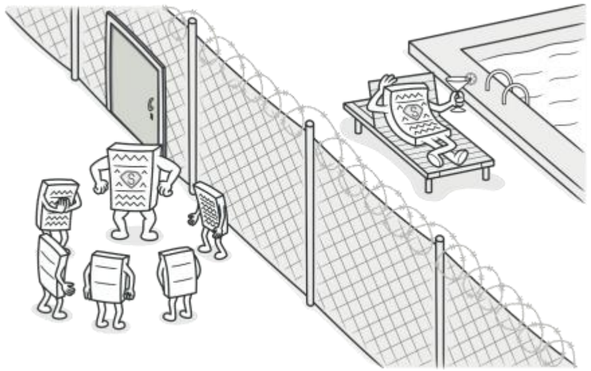
구조패턴: 클래스와 객체를 조합해 더 큰 구조를 만드는 디자인패턴

프록시패턴

*Proxy: 대리자, 대변인

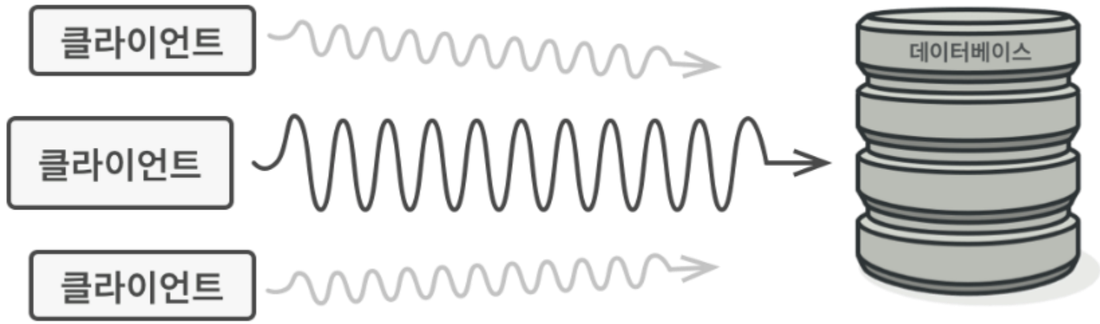
:특정 객체의 대리자나 대변인 역할을 하는 프록시 개체를 제공하는 디자인 패턴
객체를 직접 참조 X / 해당 객체의 대리인(Proxy)을 사용하여 실제 객체 사용을
실제 객체가 필요할 때까지 최대한 미루고, 그 전에는 대리인을 사용하는 패턴

-원래 객체에 대한 접근 제어
=> 요청이 원래 객체에 전달되기 전 또는 후에 무언가를 수행



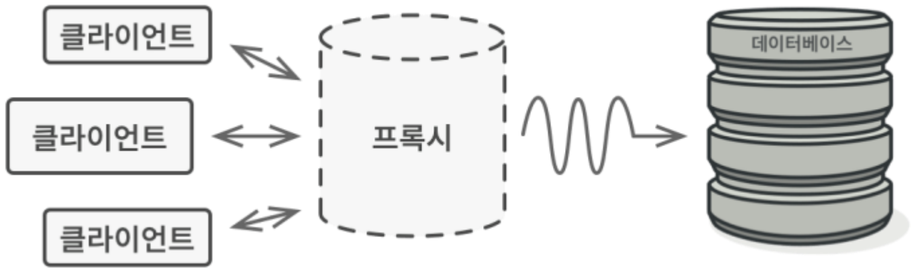
문제

객체에 대한 접근을 왜 제한할까?



실제로 필요할 때만 객체를 만들어 지연된 초기화 구현
-> 코드 중복 초래

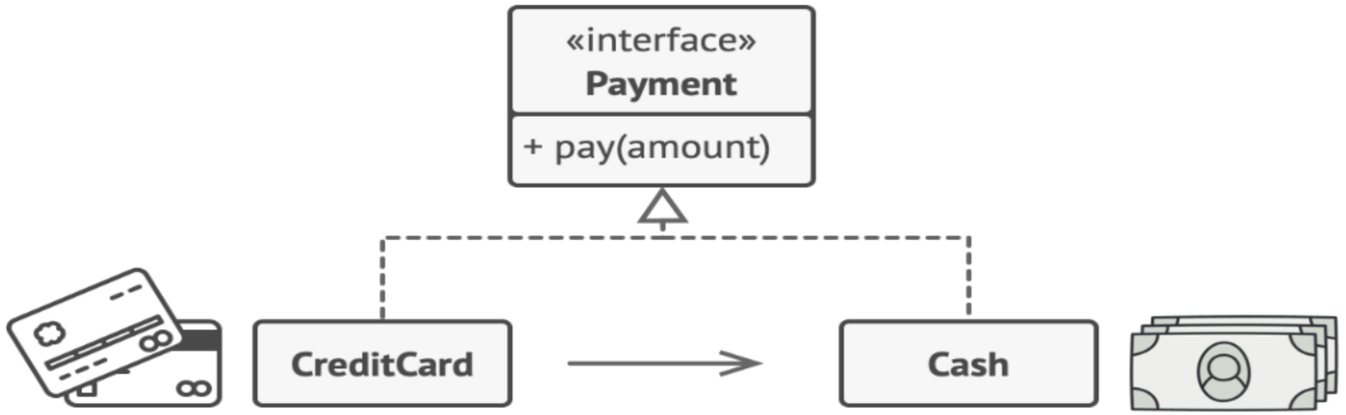
해결책



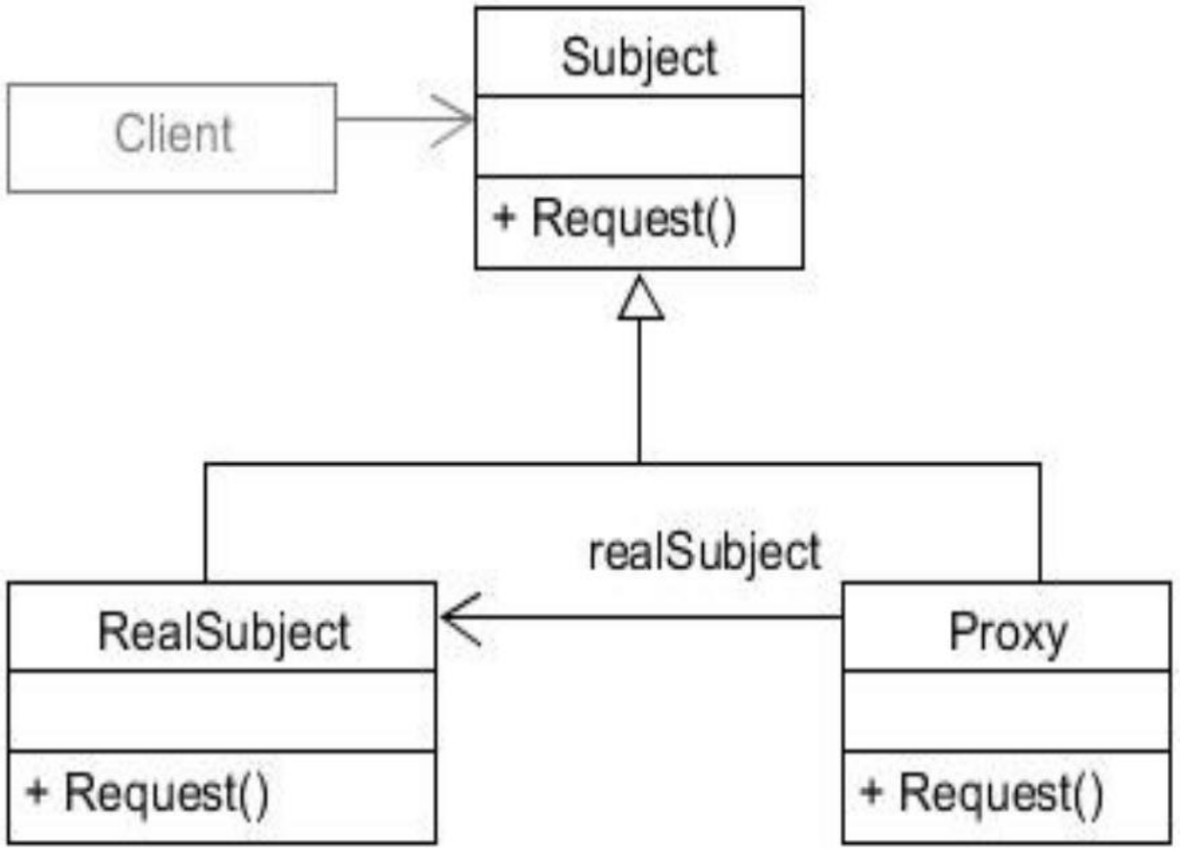
프록시 패턴은 원래 객체와 같은 인터페이스로 새 프록시 클래스의 생성 제안
-> 클래스의 메인 로직 이전이나 이후에 무언가를 실행해야 하는 경우
프록시는 해당 클래스를 변경하지 않고도 이 무언가를 수행할 수 있도록 함

if 실제상황?

신용 카드: 은행 계좌의 프록시 / 은행 계좌: 현금의 프록시
둘 다 같은 인터페이스를 구현하며 둘 다 결제에 사용될 수 있음

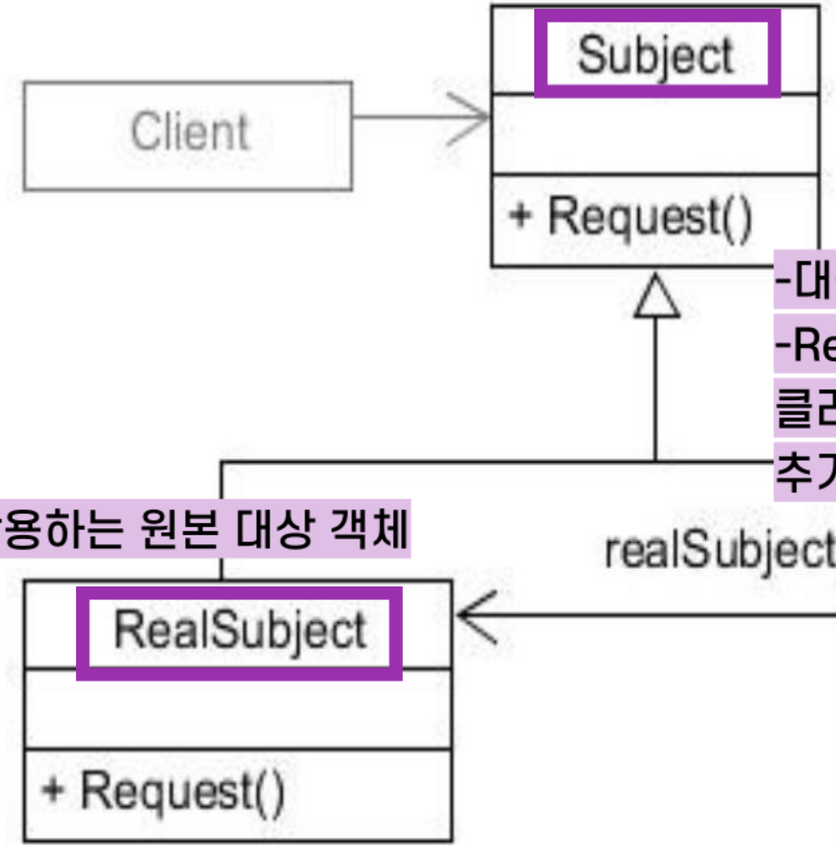


구조



구조

-Proxy와 RealSubject를 하나로 묶는 인터페이스
-클라이언트가 프록시와 대상 객체를 동일하게 다룰 수 있도록 정의



-대상 객체인 RealSubject를 중계하는 대리자 역할
-RealSubject와 같은 이름의 메서드를 호출하며,
클라이언트의 요청을 처리하기 전이나 후에
추가적인 작업을 수행

-클라이언트가 직접 상호작용하는 원본 대상 객체

프록시 패턴 예제

"이미지 뷰어 프로그램"을 만든다고 가정!

고해상도 이미지 경로를 인자로 받아 메모리에 적재

showImage() 메소드가 호출하면 화면에 렌더링 하는 HighREsolutionImage 클래스 구성

```java
class HighResolutionImage {
    String img;

    HighResolutionImage(String path) {
        loadImage(path);
    }

    private void loadImage(String path) {
        // 이미지를 디스크에서 불러와 메모리에 적재 (작업 자체가 무겁고 많은 자원을 필요로함)
        try {
            Thread.sleep(1000);
            img = path;
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.printf("%s에 있는 이미지 로딩 완료\n", path);
    }

    @Override
    public void showImage() {
        // 이미지를 화면에 렌더링
        System.out.printf("%s 이미지 출력\n", img);
    }
}
```

프록시 패턴을 적용하지 않은 코드

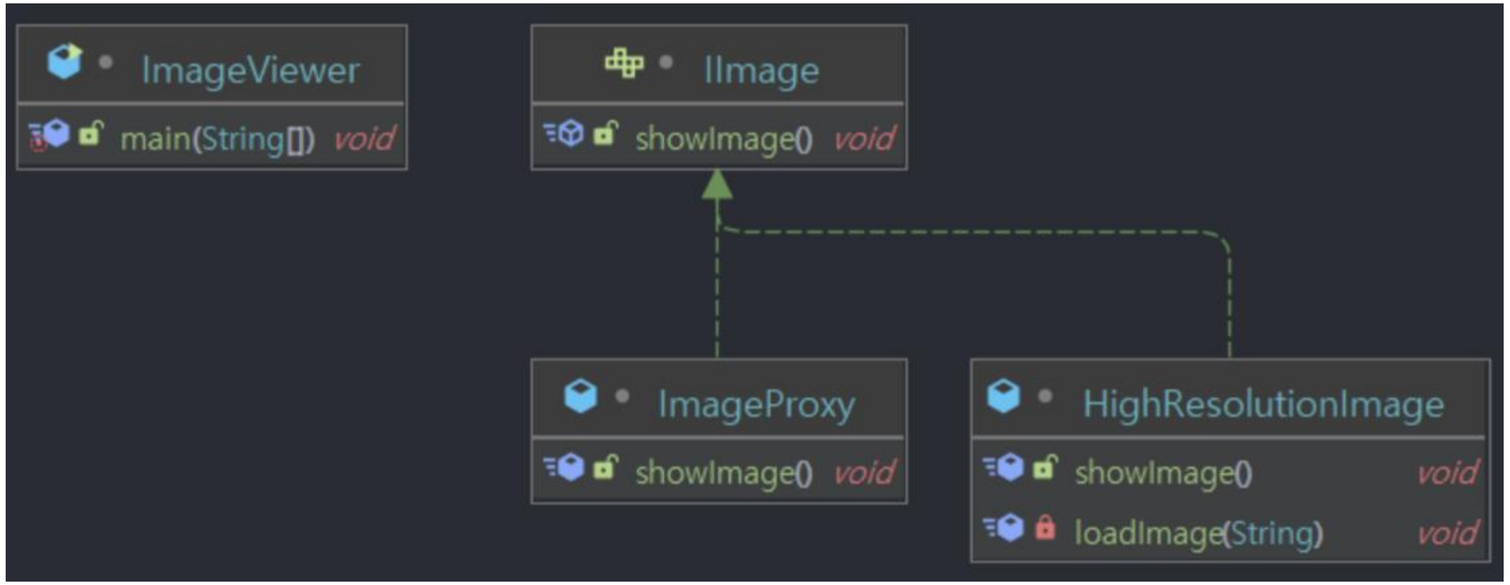
HighREsolutionImage 클래스를 이미지 뷰어인 ImageViewer 클래스에서
이미지 3개를 등록하고 로드

```
1  class ImageViewer {
2      public static void main(String[] args) {
3          HighResolutionImage highResolutionImage1 = new HighResolutionImage("./img/고해상도이미지_1");
4          HighResolutionImage highResolutionImage2 = new HighResolutionImage("./img/고해상도이미지_2");
5          HighResolutionImage highResolutionImage3 = new HighResolutionImage("./img/고해상도이미지_3");
6
7          highResolutionImage2.showImage();
8      }
9  }
```

이미지를 준비하는 과정에서 **시간을 다 뺏음**
-> 사용자가 목록에서 이미지를 선택하기 전까지 굳이 이미지를 메모리에 준비시킬
필요가 없을 것 => 사용자가 목록에서 선택한 이미지만 로딩시키면 되지 않을까?

프록시 패턴을 적용한 코드

프록시 클래스에서 사용자가 선택한 이미지만 로드해서 렌더링하도록 **대상 객체를 제어**를 하면 됨
=> 가상 프록시를 이용해 지연 초기화(Lazy Initialization)로 실제 객체의 사용 시점을 제어해보자!
(*HighResolutionImage 클래스가 대상 객체(RealSubject) 가 됨)



프록시 패턴을 적용한 코드

프록시를 구현할 때 가장 먼저 해야할 일
: 대상 객체와 프록시 객체를 하나로 묶어주는 인터페이스를 정의하는 것

```java
// 대상 객체와 프록시 객체를 묶는 인터페이스 (다형성)
interface IImage {
    void showImage(); // 이미지를 렌더링하기 위해 구현체가 구현해야 하는 추상메소드
}
```

```java
// 대상 객체 (RealSubject)
class HighResolutionImage implements IImage {
    String img;

    HighResolutionImage(String path) {
        loadImage(path);
    }

    private void loadImage(String path) {
        // 이미지를 디스크에서 불러와 메모리에 적재 (작업 자체가 무겁고 많은 자원을 필요로함)
        try {
            Thread.sleep(1000);
            img = path;
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.printf("%s에 있는 이미지 로딩 완료\n", path);
    }

    @Override
    public void showImage() {
        // 이미지를 화면에 렌더링
        System.out.printf("%s 이미지 출력\n", img);
    }
}
```

프록시 패턴을 적용한 코드

```java
// 프록시 객체 (Proxy)
class ImageProxy implements IImage {
    private IImage proxyImage;
    private String path;

    ImageProxy(String path) {
        this.path = path;
    }

    @Override
    public void showImage() {
        // 고해상도 이미지 로딩하기
        proxyImage = new HighResolutionImage(path);
        proxyImage.showImage();
    }
}
```

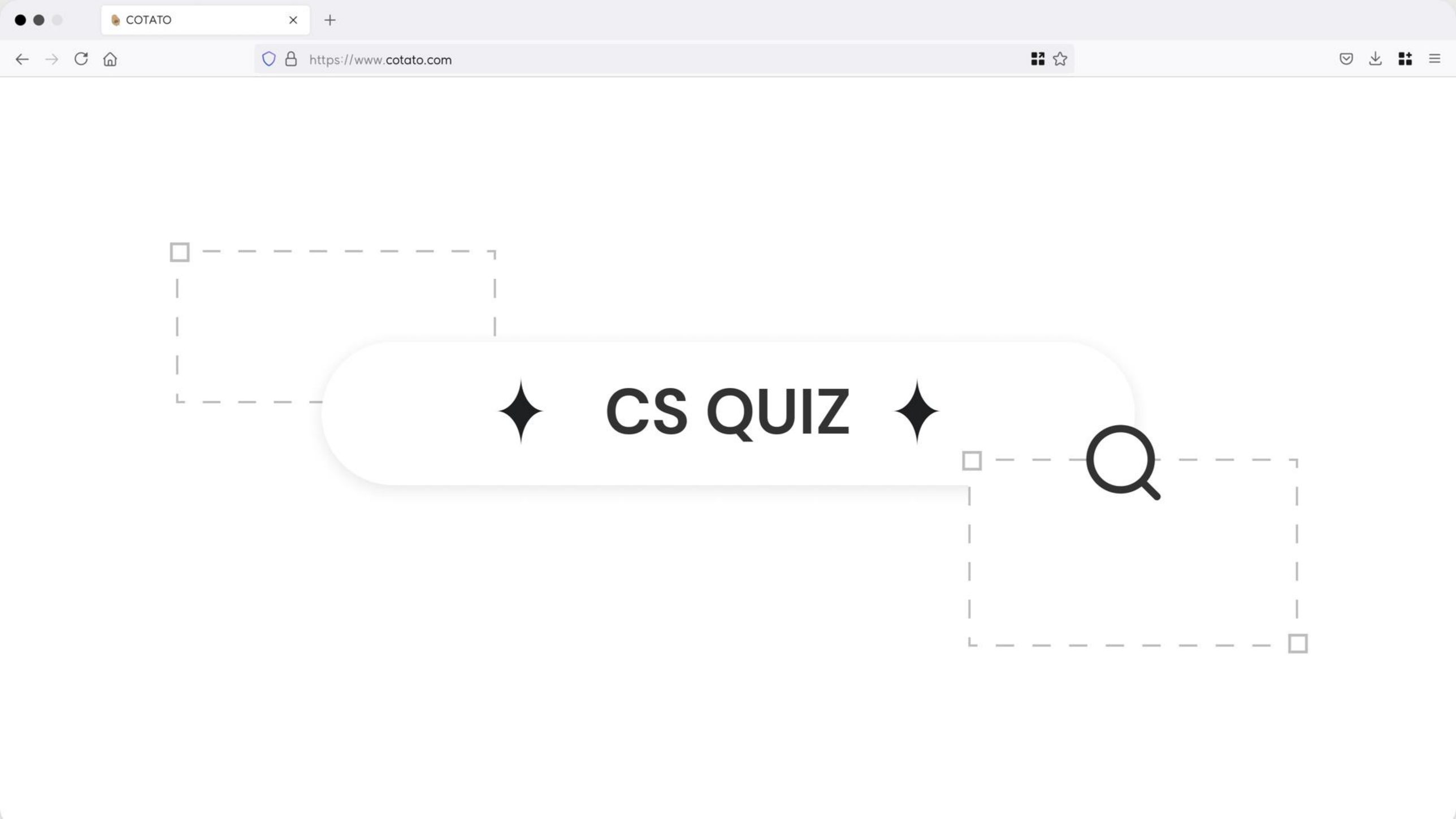
```java
class ImageViewer {
    public static void main(String[] args) {
        IImage highResolutionImage1 = new ImageProxy("./img/고해상도이미지_1");
        IImage highResolutionImage2 = new ImageProxy("./img/고해상도이미지_2");
        IImage highResolutionImage3 = new ImageProxy("./img/고해상도이미지_3");

        highResolutionImage2.showImage();
    }
}
```

프록시 객체 내에서 경로 데이터를 지니고 있다가 사용자가 showImage를 호출하면
대상 객체를 로드(lazyload) -> 이미지를 메모리에 적재하고 대상 객체의 showIMage() 메서드를
위임 호출함으로써, 실제 메소드를 호출하는 시점에 메모리 적재가 이루어짐
=> 불필요한 자원낭비가 발생하지 않게 됨

프록시패턴의 장/단점

장점	단점
개방 폐쇄 원칙(OCP) 준수 -> 기존 대상 객체의 코드를 변경하지 않고 새로운 기능을 추가할 수 있음	많은 프록시 클래스를 도입해야 하므로 코드의 복잡도가 증가함
단일 책임 원칙(SRP) 준수 -> 대상 객체는 자신의 기능에만 집중 하고, 그 이외 부가 기능을 제공하는 역할을 프록시 객체에 위임하여 다중 책임을 회피 할 수 있음	프록시 클래스 자체에 들어가는 자원이 많다면 서비스로부터의 응답이 늦어질 수 있음
사용자 입장에서는 프록시 객체나 실제 객체나 사용법은 유사하므로 사용성에 문제 되지 않음	성능이 저하될 수 있음 (객체를 생성할 때 한 단계를 거치게 되므로, 빈번한 객체 생성이 필요한 경우/ 객체 생성을 위해 스레드 생성 및 동기화가 구현되어야 하는 경우)


COTATO
https://www.cotato.com
CS QUIZ

Q.1

어떠한 보편적인 문제나 상황에 대해, 반복해서 일어나는 문제에 대한 해결방안을 정리한 것을 의미하는 이 용어를 적으시오.

COTATO

https://www.cotato.com

A.1

어떠한 보편적인 문제나 상황에 대해, 반복해서 일어나는 문제에 대한 해결방안을 정리한 것을 의미하는 이 용어를 적으시오.

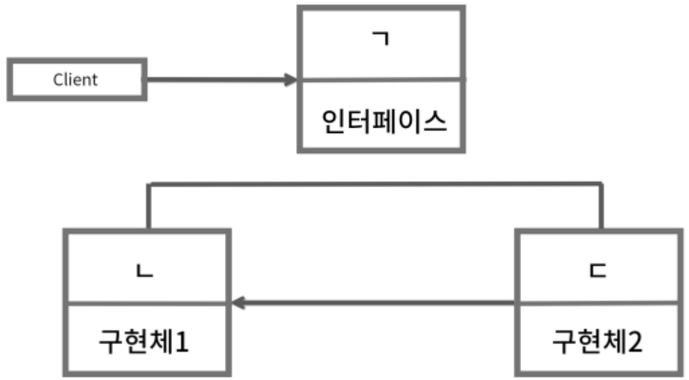
정답: 디자인 패턴

14 356

Q.2

다음은 프록시 패턴을 나타내는 구조도이다. 빈칸에 적절한 용어를 보기에서 골라 쓰시오.

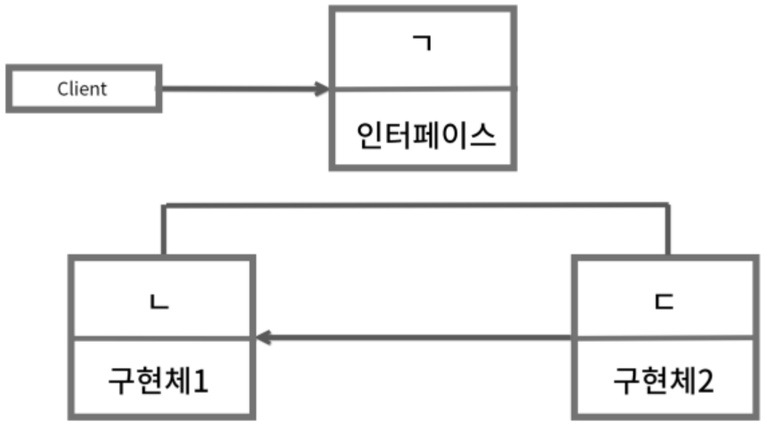
SRP RealSubject Proxy OCP Subject



A.2

다음은 프록시 패턴을 나타내는 구조도이다. 빈칸에 적절한 용어를 보기에서 골라 쓰시오.

SRP RealSubject Proxy OCP Subject



정답: ㄱ - Subject, ㄴ - RealSubject, ㄷ - Proxy

Q.3

프록시 패턴은 어떤 디자인 패턴인가요?

ㄱ. 객체의 생성을 제어하기 위한 디자인 패턴
ㄴ. 객체의 대리자 역할을 하는 디자인 패턴
ㄷ. 객체의 데이터를 숨기기 위한 디자인 패턴
ㄹ. 객체의 인터페이스를 정의하는 디자인 패턴

A.3

프록시 패턴은 어떤 디자인 패턴인가요?

ㄱ. 객체의 생성을 제어하기 위한 디자인 패턴
ㄴ. 객체의 대리자 역할을 하는 디자인 패턴
ㄷ. 객체의 데이터를 숨기기 위한 디자인 패턴
ㄹ. 객체의 인터페이스를 정의하는 디자인 패턴

정답: ㄴ

Q.4

프록시 패턴을 사용하여 객체에 대한 접근을 제한하는 이유는 무엇인가요?

ㄱ. 객체의 보안을 강화하기 위해
ㄴ. 성능 최적화를 위해
ㄷ. 중복 코드를 방지하기 위해
ㄹ. ㄱ,ㄴ, ㄷ 모두 맞다

A.4

프록시 패턴을 사용하여 객체에 대한 접근을 제한하는 이유는 무엇인가요?

ㄱ. 객체의 보안을 강화하기 위해
ㄴ. 성능 최적화를 위해
ㄷ. 중복 코드를 방지하기 위해
ㄹ. ㄱ,ㄴ, ㄷ 모두 맞다

정답: ㄹ

Q.5

프록시패턴의 장단점에 대한 내용을 바르게 이해하고 있는 친구들의 이름을 써주세요.

하정: 프록시패턴은 개방 폐쇄 원칙을 준수하기 때문에 기존 대상 객체의 코드를 변경하지 않고 새로운 기능을 추가할 수 있어.
관희: 프록시패턴을 사용하면 코드가 간단해진다는 장점이 있지.
혜선: 프록시패턴에서 클라이언트는 객체를 신경써야 한다는 점이 존재해.
민우: 프록시패턴에서 프록시 클래스 자체에 들어가는 자원이 많다면 서비스로부터 응답이 늦어질 수 있어.

A.5

프록시패턴의 장단점에 대한 내용을 바르게 이해하고 있는 친구들의 이름을 써주세요.

하정: 프록시패턴은 개방 폐쇄 원칙을 준수하기 때문에 기존 대상 객체의 코드를 변경하지 않고 새로운 기능을 추가할 수 있어.
관희: 프록시패턴을 사용하면 코드가 간단해진다는 장점이 있지.
혜선: 프록시패턴에서 클라이언트는 객체를 신경써야 한다는 점이 존재해.
민우: 프록시패턴에서 프록시 클래스 자체에 들어가는 자원이 많다면 서비스로부터 응답이 늦어질 수 있어.

정답: 하정, 민우

Q.6

프록시패턴의 구조에 대한 설명으로 옳지 않은 것을 고르시오.

1. Subject는 Proxy와 RealSubject를 하나로 묶는 인터페이스로 클라이언트가 프록시와 대상 객체를 동일하게 다룰 수 있도록 정의한다.
2. RealSubject는 클라이언트가 직접 상호작용하는 원본 대상 객체다.
3. Proxy는 대상 객체인 RealSubject를 중계하는 대리자 역할을 한다.
4. Proxy는 RealSubject와 같은 이름의 메서드를 호출하며, 클라이언트의 요청을 처리하기 전이나 후에 추가적인 작업을 수행할 수 없다.

A.6

프록시패턴의 구조에 대한 설명으로 옳지 않은 것을 고르시오.

1. Subject는 Proxy와 RealSubject를 하나로 묶는 인터페이스로 클라이언트가 프록시와 대상 객체를 동일하게 다룰 수 있도록 정의한다.
2. RealSubject는 클라이언트가 직접 상호작용하는 원본 대상 객체다.
3. Proxy는 대상 객체인 RealSubject를 중계하는 대리자 역할을 한다.
4. Proxy는 RealSubject와 같은 이름의 메서드를 호출하며, 클라이언트의 요청을 처리하기 전이나 후에 추가적인 작업을 수행할 수 없다.

정답: 4

Q.7

프록시 패턴에 대한 잘못된 설명을 고르시오.

1. 코드의 복잡도가 증가할 수 있다.
2. 사용자 입장에서는 프록시 객체와 실제 객체의 사용법 모두를 알고 있어야 한다.
3. 요청이 원래 객체에 전달되기 전 또는 후에 무언가를 수행할 수 있도록 한다.
4. 프록시는 해당 클래스를 변경하지 않고도 이 무언가를 수행할 수 있도록 한다.
5. 늦은 초기화(Lazy Initialization)로 실제 객체 사용 시점을 제어할 수 있다.

A.7

프록시 패턴에 대한 잘못된 설명을 고르시오.

1. 코드의 복잡도가 증가할 수 있다.
2. 사용자 입장에서는 프록시 객체와 실제 객체의 사용법 모두를 알고 있어야 한다.
3. 요청이 원래 객체에 전달되기 전 또는 후에 무언가를 수행할 수 있도록 한다.
4. 프록시는 해당 클래스를 변경하지 않고도 이 무언가를 수행할 수 있도록 한다.
5. 늦은 초기화(Lazy Initialization)로 실제 객체 사용 시점을 제어할 수 있다.

정답: 2

Q.8

프로그램 실행 중에 필요한 시점에 데이터를 로드하거나 객체를 초기화하는 기술을 "영어"로 작성하시오.

A.8

프로그램 실행 중에 필요한 시점에 데이터를 로드하거나 객체를 초기화하는 기술을 "영어"로 작성하시오.

정답: Lazy Loading, Lazy Initialization

Q.9

이미지 뷰어 프로그램을 만들고 있는 감자는 이미지를 불러오고 보여주는 객체(class HighResolutionImage)를 사용하고자 한다. 하지만 이 객체는 객체 생성과 동시에 이미지를 불러와 많은 시간이 소모되어, 감자는 프록시 패턴을 적용하려 한다. 이때 감자의 생각으로 잘못된 것을 고르시오.

1. 프록시 클래스를 정의해서 프록시 객체를 생성한다.
2. 프록시 클래스에는 이미지를 보여주는 함수가 정의되어 있다.
3. 프록시 객체가 생성될 때 기존 객체도 함께 생성된다.
4. 프록시 객체는 인터페이스를 정의한 구현체이다.

```java
// 프록시 객체 (Proxy)
class ImageProxy implements IImage {
    private IImage proxyImage;
    private String path;

    ImageProxy(String path) {
        this.path = path;
    }

    @Override
    public void showImage() {
        // 고해상도 이미지 로딩하기
        proxyImage = new HighResolutionImage(path);
        proxyImage.showImage();
    }
}
```

A.9

이미지 뷰어 프로그램을 만들고 있는 감자는 이미지를 불러오고 보여주는 객체(class HighResolutionImage)를 사용하고자 한다. 하지만 이 객체는 객체 생성과 동시에 이미지를 불러와 많은 시간이 소모되어, 감자는 프록시 패턴을 적용하려 한다. 이때 감자의 생각으로 잘못된 것을 고르시오.

1. 프록시 클래스를 정의해서 프록시 객체를 생성한다.
2. 프록시 클래스에는 이미지를 보여주는 함수가 정의되어 있다.
3. 프록시 객체가 생성될 때 기존 객체도 함께 생성된다.
4. 프록시 객체는 인터페이스를 정의한 구현체이다.

정답: 3

```
// 프록시 객체 (Proxy)
class ImageProxy implements IImage {
    private IImage proxyImage;
    private String path;

    ImageProxy(String path) {
        this.path = path;
    }

    @Override
    public void showImage() {
        // 고해상도 이미지 로딩하기
        proxyImage = new HighResolutionImage(path);
        proxyImage.showImage();
    }
}
```

다음 실제 생활 예시 중 프록시 패턴에 가장 적합하지 않은 상황을 고르시오.

1. 연예인은 직접 스케줄을 잡는 것이 아니라, 항상 매니저를 통해 스케줄을 잡는다.
2. 친구 10명이 같은 책이 필요한 경우엔 각자 구입하기보단, 공동 구매자를 선정하고 공동 구매를 해야겠다.
3. 많은 현금을 들고 다니기 부담스러우니까 신용카드를 사용하고, 현금은 나중에 통장에서 빠지게 처리해야겠다.
4. 여러 가게에서 음식을 주문 후 라이더에게 음식을 배달시키기보단, 내가 직접 가서 음식을 포장해와야겠어.
5. 이미지와 텍스트를 동시에 로딩하지 않고 대리 객체를 만들어 텍스트만 먼저 로딩하고 이미지는 추후에 불러온다.

다음 실제 생활 예시 중 프록시 패턴에 가장 적합하지 않은 상황을 고르시오.

1. 연예인은 직접 스케줄을 잡는 것이 아니라, 항상 매니저를 통해 스케줄을 잡는다.
2. 친구 10명이 같은 책이 필요한 경우엔 각자 구입하기보단, 공동 구매자를 선정하고 공동 구매를 해야겠다.
3. 많은 현금을 들고 다니기 부담스러우니까 신용카드를 사용하고, 현금은 나중에 통장에서 빠지게 처리해야겠다.
4. 여러 가게에서 음식을 주문 후 라이더에게 음식을 배달시키기보단, 내가 직접 가서 음식을 포장해와야겠어.
5. 이미지와 텍스트를 동시에 로딩하지 않고 대리 객체를 만들어 텍스트만 먼저 로딩하고 이미지는 추후에 불러온다.