프록시 패턴
Proxy pattern

작성자 : 박천호

프록시 패턴 vs 컴포지트 패턴

- 선정이유 : 더 많이 사용하는 더 중요한 패턴을 하나라도 더 깊게 이해해서 전우들에게 전달하자

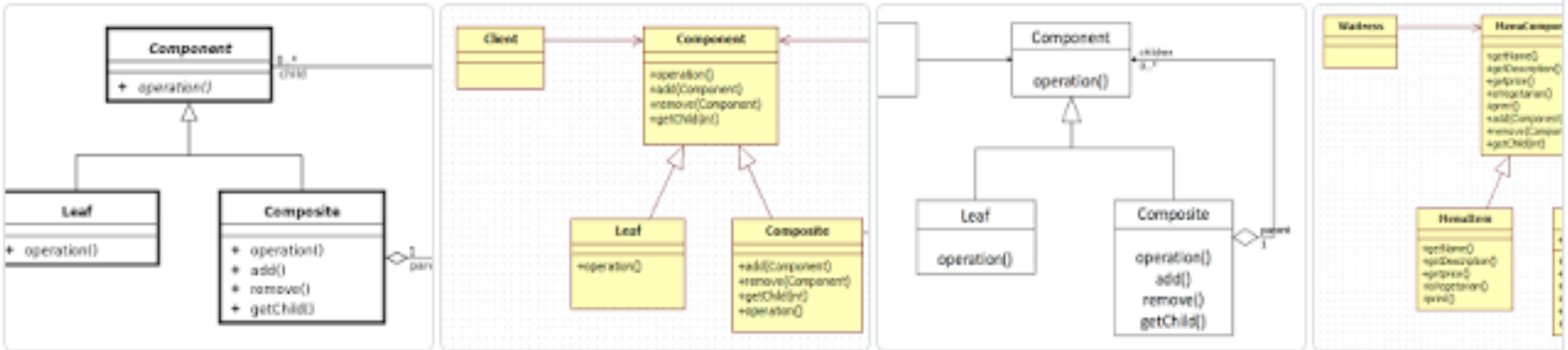
- 선정 기준 : 코딩은 곧 구글링이다 -> 구글에 많은 정보가 있는 것이 코딩에서 더 많이 사용한다 -> 구글에 검색해서 나오는 데이터의 양이 많으면 더 중요하다

Google

컴포지트 패턴

전체 이미지 동영상 뉴스 지도 더보기 설정 도구

검색결과 약 30,400개 (0.31초)

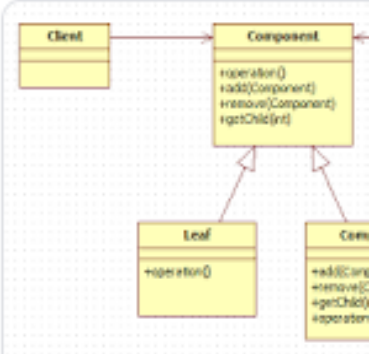


컴포지트 패턴(Composite pattern)이란 객체들의 관계를 트리 구조로 구성하여 부분-전체 계층을 표현하는 **패턴**으로, 사용자가 단일 객체와 복합 객체 모두 동일하게 다루도록 한다.

ko.wikipedia.org › wiki › 컴포지트_패턴

컴포지트 패턴 - 위키백과, 우리 모두의 백과사전

추천 스니펫 정보 사용자 의견



컴포지트

컴포지트 패턴이란 객
계층을 표현하는 패턴
일하게 다루도록 한다

Google

프록시 패턴

전체 이미지 뉴스 동영상 쇼핑 더보기 설정 도구

검색결과 약 118,000개 (0.33초)

jdm.kr › blog

프록시 패턴(Proxy Pattern) :: JDM's Blog

2019. 4. 7. - 실제 기능을 수행하는 객체Real Object 대신 가상의 객체Proxy Object를 사용해 로직의 흐름을 제어하는 디자인 패턴입니다. 2. **프록시 패턴** 특징.

ko.wikipedia.org › wiki › 프록시_패턴

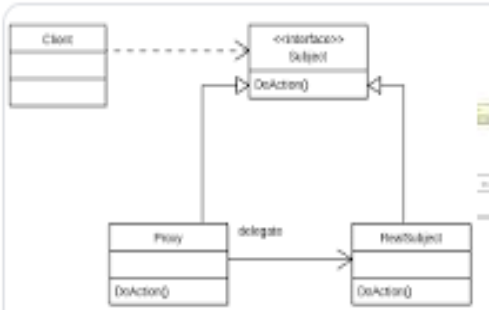
프록시 패턴 - 위키백과, 우리 모두의 백과사전

프록시 패턴(**proxy pattern**)은 컴퓨터 프로그래밍에서 소프트웨어 디자인 패턴의 하나이다. 일반적으로 프록시는 다른 무언가와 이어지는 인터페이스의 역할을 하는 ...

limkydev.tistory.com › ...

[Design_Pattern] 프록시 패턴(Proxy Pattern) - Limky 삽질블로그

2017. 8. 25. - 안녕하세요. Limky 입니다. 이번 시간은 **프록시 패턴**(**Proxy Pattern**)에 대해서 알아보겠습니다. Proxy는 우리말로 대리자, 대변인 이라는 뜻입니다.



프록시 패턴

프록시 패턴은 컴퓨터 프로그라
나이다. 일반적으로 프록시는 다
역할을 하는 클래스이다. 프록시
을 수행할 수 있다. 프록시 패턴
트 포인터 객체이다. 위키백과

Google

컴포짓 패턴

전체 이미지 동영상 뉴스 지도 더보기 설정 도구

검색결과 약 56,300개 (0.38초)

다음 검색어에 대한 결과 포함: **컴포지트** 패턴
다음 검색어에 대한 결과만 표시: 컴포짓 패턴

mygumi.tistory.com › ...

컴포지트 패턴(Composite Pattern) :: 마이구미 :: 마이구미의 ...

2019. 3. 10. - 이 글은 디자인 **패턴** 중 **컴포지트 패턴**(Composite **Pattern**) 을 다룬다. 위키피디아의 내용을 기반으로 정리할 예정이다.

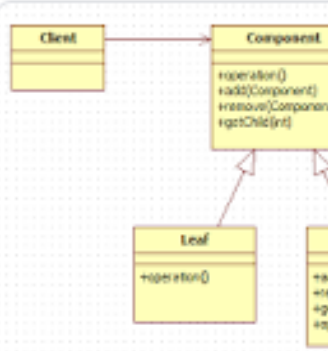
jdm.kr › blog

컴포지트 패턴(Composite Pattern) :: JDM's Blog

2016. 3. 30. - 여기서 **컴포지트**Composite의 의미는 일부 또는 그룹을 표현하는 객체들을 트리 구조Tree Structures로 구성한다는 겁니다. 2. **컴포지트 패턴**의 활용.

ko.wikipedia.org › wiki › 컴포지트_패턴

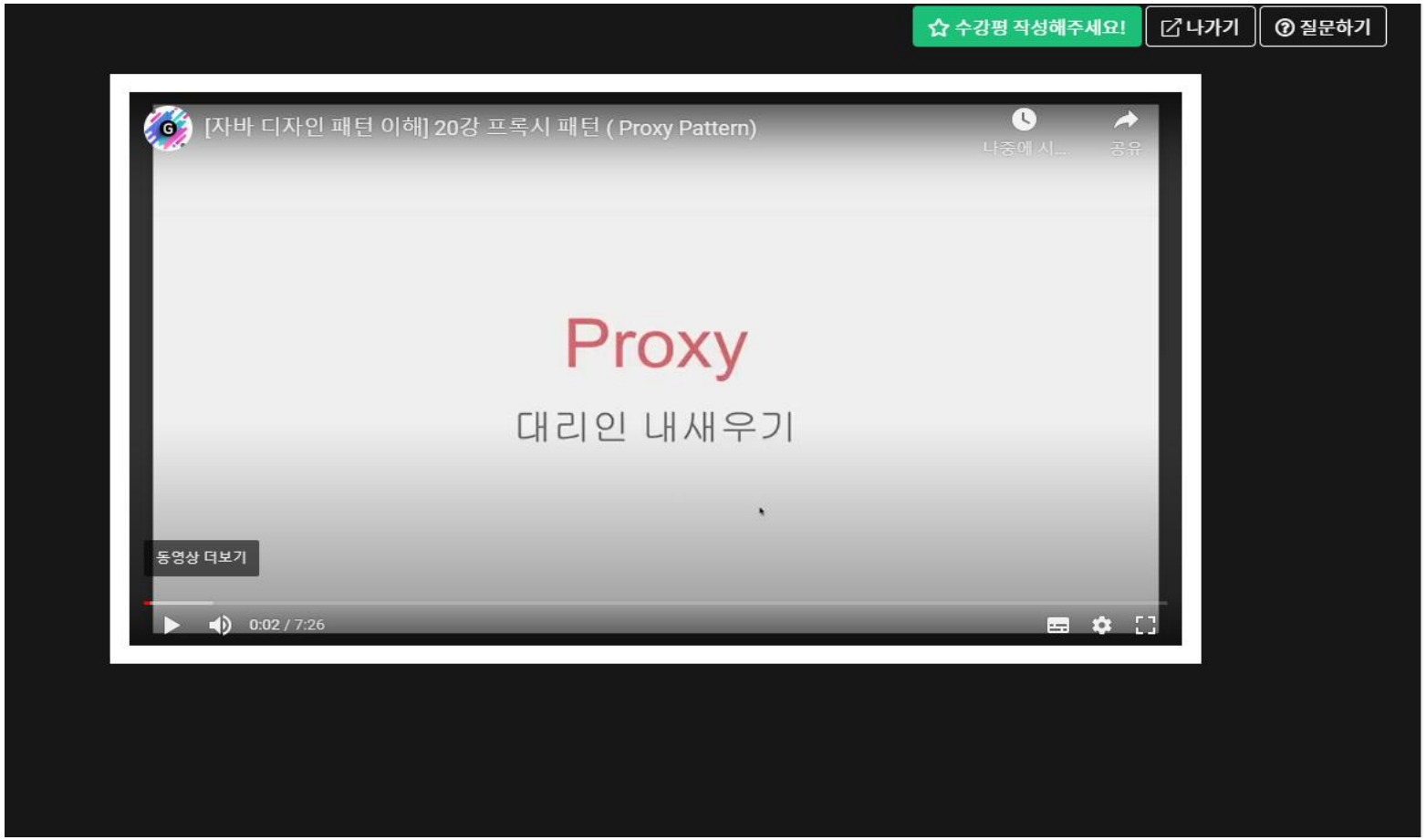
컴포지트 패턴 - 위키백과, 우리 모두의 백과사전



컴포지트

컴포지트 패턴이란
계층을 표현하는 패
일하게 다루도록 한

가람센세의 만행



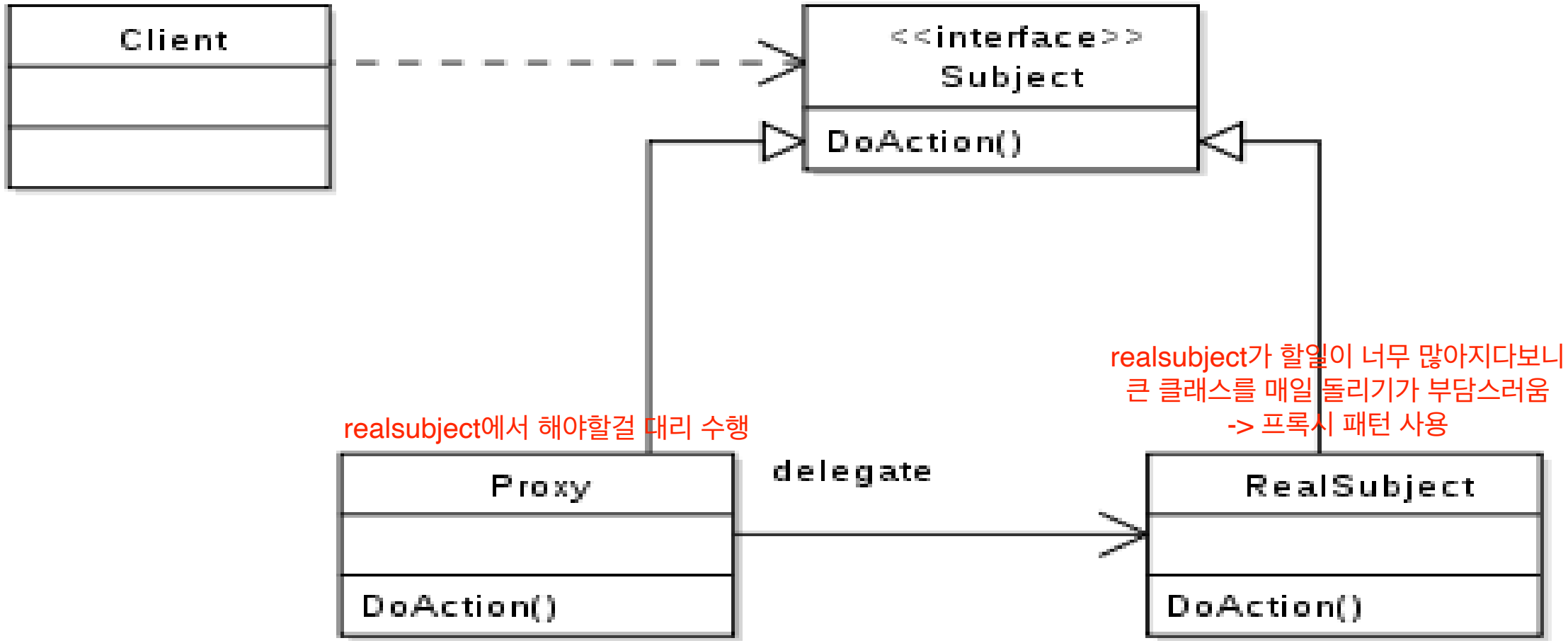
7분 26초 만에 끝내 버리는 가람신 당신은 대체....

=> 구글링 해서 다른 동영상과 블로그를 위조로 설명하겠습니다.

정의 대리(행위)나 대리권, 대리 투표, 대리인 등을 뜻한다.

- 프록시 패턴(proxy pattern)은 컴퓨터 프로그래밍에서 소프트웨어 디자인 패턴의 하나이다.
- 일반적으로 프록시는 다른 무언가와 이어지는 인터페이스의 역할을 하는 클래스이다. 프록시는 어떠한 것(이를테면 네트워크 연결, 메모리 안의 커다란 객체, 파일, 또 복제할 수 없거나 수요가 많은 리소스)과도 인터페이스의 역할을 수행할 수 있다.
- 프록시 패턴의 잘 알려진 예로는 참조 횟수 스마트 포인터 객체이다.
- 복합적인 오브젝트들의 다수의 복사본이 존재해야만 하는 상황에서 프록시 패턴은 애플리케이션의 메모리 사용량을 줄이기 위해서 플라이웨이트 패턴과 결합된 형태로 나올 수도 있다.

UML


Client
<<interface>>
Subject
DoAction()
realsubject에서 해야할걸 대리 수행
realsubject가 할일이 너무 많아지다보니
큰 클래스를 매일 돌리기가 부담스러움
-> 프록시 패턴 사용
Proxy
DoAction()
delegate
RealSubject
DoAction()

프록시 패턴

- 작업을 나눠서 구현 한다
- A객체에 대한 접근을 제어하기 위해 a를 대신할 b를 제공하여 주는 것이 proxy패턴
- 프록시의 종류 : 원격 / 보호 / 스마트
- 원격 : 원격에 있는 객체를 대신해 주는 프록시
- 가상 : 생성하는데 리소스가 많이 드는 객체가 준비 될 때 까지 대신해주는 프록시
- 스마트 : 원래 객체에 대한 접근하기 전에 사전에 처리할 것이 있는 경우 (동시 접근제한, 접근 수 통계처리) 에 사용

프록시를 만들어 봅시다

- 프록시와 리얼 서브젝트가 공유 할 인터페이스 서브젝트 생성
- 인터페이스 서브젝트에서 프록시와 리얼 서브젝트에서 오버라이드 할 메소드 만들어 두기
- 프록시 만들기 메소드 재정의
- 리얼 서브젝트에서 메소드 재정의
- 메인에서 객체 생성해서 메소드 실행

1. 공유할 인터페이스 생성

```
1 package ProxyPattern;
2
3 public interface Subject_1 {
4     public void Show_Name();
5     public void set_Name(String name);
6     public void Complicated_Work();
7 }
8
```

2. 프록시가 해결 못하는 거 해결할 리얼 서브젝트 생성

RealSubject 클래스

```
1  package ProxyPattern;
2
3  public class real_subject_1 implements Subject_1{
4
5      private String name;
6
7          public void set_Name(String name)
8          {
9              this.name = name;
10         }
11
12         public void Show_Name()
13         {
14             System.out.println("my name is "+ name);
15         }
16
17         public void Complicated_Work()
18         {
19             System.out.println("proxy가 처리 못하는 작업을 수행합니다.");
20         }
21 }
```

인터페이스를 상속받기때문에 어쨌든 subject_1의 모든 추상메소드를 재정의하긴해야하지만,
set_name과 show_name은 프록시에서 한다.

리소스가 많이 들어가는 작업이라 real subject가 해결해야함

3. 리얼 서브젝트가 못하는 프록시 생성

```
3  public class proxy_1 implements Subject_1{
4
5      public real_subject_1 subject;
6      public String name;
7
8        public void set_Name(String name)
9          {
10             System.out.println("proxy가 대신해서 처리 할 수 있어요.");
11             if(subject!= null)
12             {
13                 subject.set_Name(name);
14             }
15             this.name =name;
16         }
17
18         public void Show_Name()
19         {
20             System.out.println("proxy가 대신해서 처리 할 수 있어요.");
21             System.out.println("my name is"+ name);
22         }
23         public void Complicated_Work() 정의를 아예 안하고 상속만 받아놓음
```

complicated work()가 먼저 수행되어야 proxy가 작동하기 시작할 수 있다.
complicated work()가 수행될때 subject 객체가 선언되어 그때부터 간단한 일을
proxy가 수행한다.

3. 리얼 서브젝트가 못하는 프록시 생성

```
22          }
23          public void Complicated_Work()
24          {
25              subject = new real_subject_1();
26              subject.Complicated_Work();
27          }
28  }
29
```

복잡한 일은 Real Subject가 하도록 던짐

4. 메인 메소드에서 실행

```
1  package ProxyPattern;
2
3  public class main_1 {
4
5      public static void main(String argsp[])
6      {
7          proxy_1 proxy1 = new proxy_1();
8
9          proxy1.set_Name("홍길동");
10         System.out.println("========");
11
12         proxy1.Show_Name();
13         System.out.println("================");
14
15         proxy1.Complicated_Work();
16     }
17 }
18
```

5. 결과물

```
proxy가 대신해서 처리 할 수 있어요.
==================
proxy가 대신해서 처리 할 수 있어요.
my name is홍길동
==================
proxy가 처리 못하는 작업을 수행합니다.
```

참고한 사이트

- https://www.inflearn.com/course/%EC%9E%90%EB%B0%94-%EB%94%94%EC%9E%90%EC%9D%B8-%ED%8C%A8%ED%84%B4/lecture/3211
- https://lktprogrammer.tistory.com/34