Adapter Pattern

결론 및 정리

정의

> *23.08.02 : 정리*
> *target 과 adaptee 는 그냥 둘 다 interface 임.*
> *근데, target 이 많이 쓰이거나 해서, adaptee 를 target 으로 바꿔주고자 하는 경우가 있을 수 있음. (**Context / Problem**)*
> *이때, **Adapter 를 사용함. Adapter 는 Concrete Target class 의 일종으로, Adaptee를 변수로 가지고 이를 활용하여 구현해주는 친구**라고 생각하면 됨.*
> *이 까지만 생각하면 사실 끝임.*
>
> *조금 더 깊게 생각하면, Adaptee 가 Target 으로 변한 것도 아님. 콘센트에 비유할 필요도 없음. 그냥 있는 그대로 생각하자.*
> *Adaptee 라는 다른 Interface 를 Target 에 대응하기 위해 만든 Concrete Target 이 Adapter 이며, Adapter는 Adaptee를 변수로서 가진다.*

서로 호환되지 않는 인터페이스를 가진 두 개의 클래스를 함께 작동할 수 있도록 해주는 패턴. 보통, 한 쪽을 한 쪽에 맞춰 구현한다.

Target은 바꿔서 나오는 결과물로 내고자 하는 Interface다.
Client는 Target Class 를 변수로 지녀서 사용하는 Class 이다.
Adaptee는 바뀌게 될 Interface이다.
Adapter는 Adaptee를 Clinet 로 바꿔주는 역할을 수행하는 Class 이다.

내부 과정은 어떻게 동작하는가?

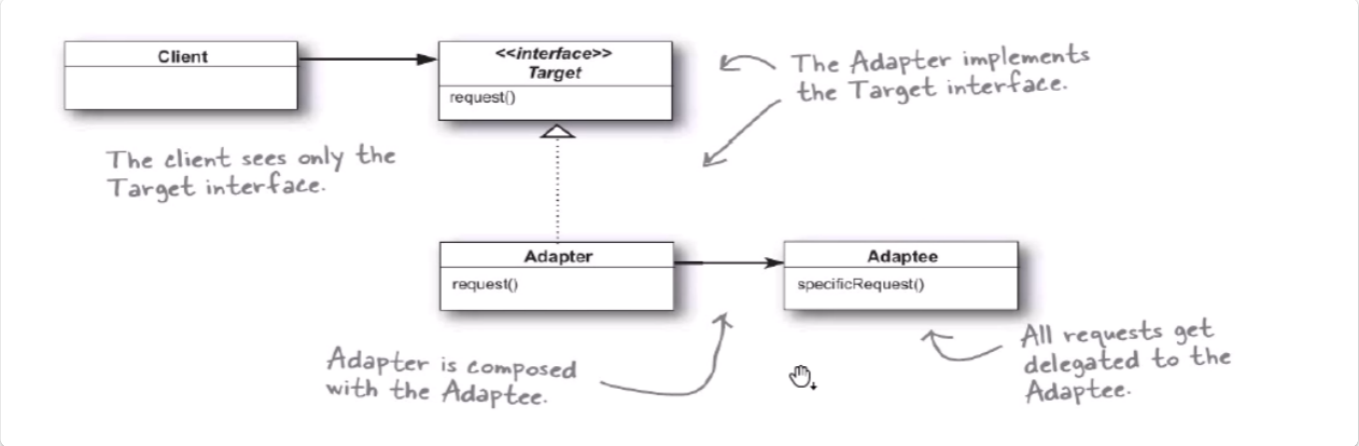
1. Client 가, 가지고 있는 Target Interface 내의 변수 - Adapter 에게 Request 를 준다. Request 는 Client 가 스스로의 method를 실행하는 것을 의미한다.
2. Adapter에는 해당 Request 를 어떻게 처리할지에 대한 내용이 들어있으며, 이는 보통 Adaptee Class에서 대응되는 Method를 Target Interface 의 method 에서 어떻게 실행할지로 구현되어 있다.
3. Client 는 Target Interface를 기반으로 결과물을 받으며, 거기에 Adapter 가 있는지 없는지는 알지 못한다.

이를 통해 얻을 수 있는 Decoupling 이, Adapter Class 의 장점이다.

> *23.07.12 다시 살펴봤을 때 해당 내용이 무슨 말 하는 지 잘 모르겠음.*
> *지금 봤을 때 이해가 되는 내용은 다음과 같음.*
>
> - *Adaptee를 Target으로 바꿔주는 게 Adapter 다.*
>   - *이 바꿔준다는 것은, 다음과 같은 상황을 의미한다.*
>   - *Target Interface 는 결국 Interface다. 이 친구를 기반으로 해서 implements 되는 많은 class들이 존재할 수 있다. 그 중에 Adapter 가 하나 섞여 있는 거다. Adapter 는 Adaptee 를 field 로서 가지고 있으며, Adaptee 의 method 를 사용하여 Target Interface 의 method 들을 implements 한다.*
>   - *위의 경우, 이는 Adaptee가 Target 처럼 사용되는 거나 마찬가지 아닌가? 이를 중간에 경유하는 친구인 Adapter 는 말 그대로 Adapter 가 아닌가?*
> - *Request 는 Client 가 Adapter 에 전달하는 호출 혹은 요청 그 자체를 의미한다. Target Interface 내에 들어있는 Method 가 Adapter Class 에서 구현될 때, Adapter Class는 보통 Adaptee Class의 method 를 기반으로 하여 호출한다. 이는 두 가지 사항을 의미한다.*
>   - *Adaptee Class 가 Target Interface 처럼 동작할 수 있다는 것.*
>   - *Clinet 는 그 안에 Adaptee Class가 들어있는 지 모르는데도, Adaptee 의 동작을 실행할 수 있다는 것.*

구현

구현 자체는 굉장히 단순하다. 다음과 같은 구조를 바탕으로, 제작한다.




```
public class TurkeyAdapter implements Duck {
        Turkey turkey;

        public TurkeyAdapter (Turkey turkey) {
                this.turkey = turkey
        }

        // ... Other Converting method turkey to duck
}
```

Target Interface 와 Adaptee Interface 가 존재한다 하자.
Target Interface를 구현하는 Adapter Class를 만들어, Adaptee Interface 를 변수로서 가지도록 한다.
이후, Target Interface 의 method 를 구현할 때, Adaptee Class 의 Method를 대응하여 사용한다.
이를 통해, Adaptee Class 가 Target Class 처럼 동작할 수 있으며, Client Class 는 이를 모르고 사용한다.

두 Class 가 하나 처럼 취급되어 사용될 수 있는 것이다.
이를 만들어 두면, 다음에 Adapter Class 를 이용하는 것도 쉽다.

Client 는 Target Interface 로 변수 만들어서 가지고.
해당 변수에 Adapter Class를 생성하며, 넣어주면 되는데, 이 때 Adapter 가 대응으로서 가지고 있는 Adaptee Class 만 함께 넣어주면 되는 것이기 때문이다.

Context & Problem & Merit

- Context / Problem : 하나의 Interface 가 다른 Interface 에 대응(호환)되도록 만들고자 할 때 사용한다.
- Merit : 적절한 Decoupling, Target Interface 를 사용하는 Clinet Class 는 Adapter 의 존재 유무 등을 신경쓰지 않고 사용할 수 있다.

강의 들으며

다른 두 Interface 가 존재하고, 이 둘이 호환될 수 있도록 도와주는 거라는 건 알겠는데. 뭐가 어느 쪽으로 맞춰지게 하며, 무슨 동작을 정확히 하는지는 이해하지 못했음.

하나를 Interface 로 얻고, 다른 하나를 Instance Variable 로 얻어둠.
영상의 예시에선, Turkey 와 Duck 이란 Interface 가 있었는데.

```
public class TurkeyAdapter implements Duck {
        Turkey turkey;

        public TurkeyAdapter (Turkey turkey) {
                this.turkey = turkey
        }

        // ... Other Converting method turkey to duck
}
```

과 같이 전환함.

전선으로서 서술되어 있는 거에서, 뭘 꼳고, 뭐는 들어가고, 뭐는 바꿔주고, 이걸 생각하니까 더 복잡해지는 듯 하다.
전선 없이 빼고 생각해보자.

Target은 바꿔서 나오는 결과물로 내고자 하는 Interface다.
Client는 Target Class 를 변수로 지녀서 사용하는 Class 이다.
Adaptee는 바뀌게 될 Interface이다.
Adapter는 Adaptee를 Clinet 로 바꿔주는 역할을 수행하는 Class 이다.

내부 과정은 어떻게 동작하는가?

1. Client 가, 가지고 있는 Target Interface 내의 변수 - Adapter 에게 Request 를 준다. Request 는 Client 가 스스로의 method를 실행하는 것을 의미한다.
2. Adapter에는 해당 Request 를 어떻게 처리할지에 대한 내용이 들어있으며, 이는 보통 Adaptee Class에서 대응되는 Method를 Target Interface 의 method 에서 어떻게 실행할지로 구현되어 있다.
3. Client 는 Target Interface를 기반으로 결과물을 받으며, 거기에 Adapter 가 있는지 없는지는 알지 못한다.

즉, Duck 이 Target Interface 이고, Turkey가 Adaptee, TurkeyAdapter 가 Adapter Class, Duck을 변수로서 가지고 사용하는 Class 가 Client 이다!

이는 Decoupled 되어 있다!

Multiple Inheritence가 가능한 Language 라면, 이걸 이용해서 구현할 수도 있다!
(물론 Composition 이 더 권장되므로, 기존 구조를 사용하는 것이 좋다.)

이후 Phone Adapter 의 실행예시를 보여준다.