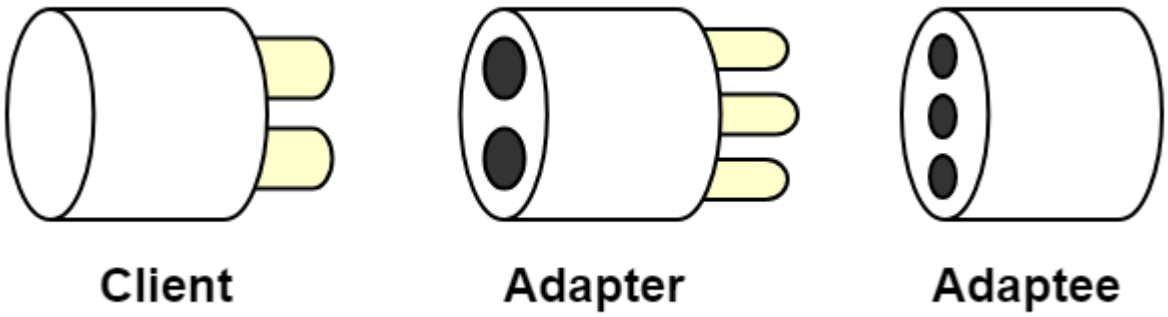
Adapter Pattern

Adapter Pattern(적응자 패턴)

기본 개념



의도

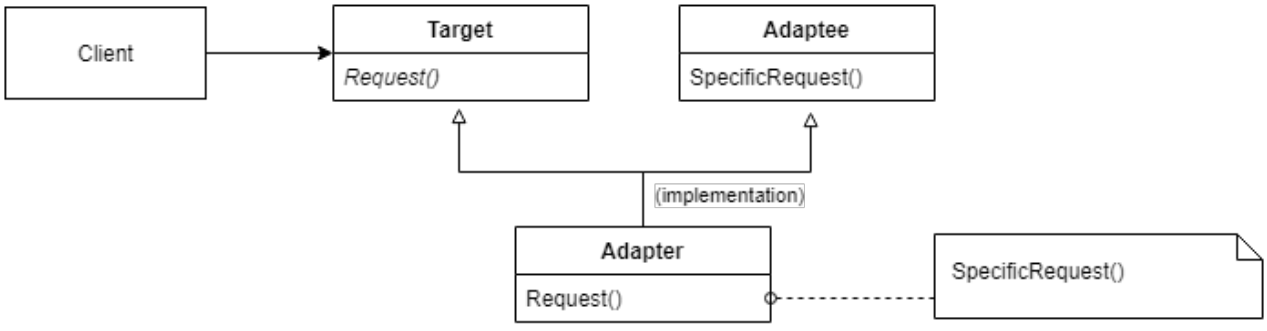
서로 일치 하지 않는 인터페이스를 갖는 클래스들을 함께 묶어 사용하고 싶을 때

사용자(Client)가 원하는 형태로 인터페이스를 변환 시켜 사용

적응자 패턴 작성법

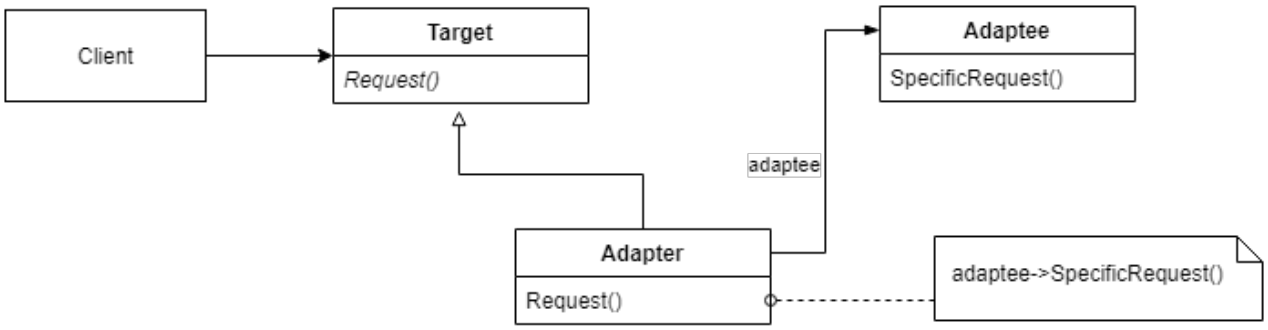
(1) 클래스 적응자 활용

다중 상속을 이용하여 인터페이스를 적응시키는 방식



(2) 객체 적응자 활용

단일 상속 + 객체 합성을 사용하여 인터페이스를 적응 시키는 방식



Example (GoF의 디자인패턴 책 내용)

그래픽 편집기 케이스

(1) 목적:

그림 편집기에서 그래픽요소를 생성 관리 하기 위해 Shape 클래스와 그의 서브 클래스들

(ex. LineShape, PolygonShape, TextShape 등등)로 나누어 개발이 필요하다.

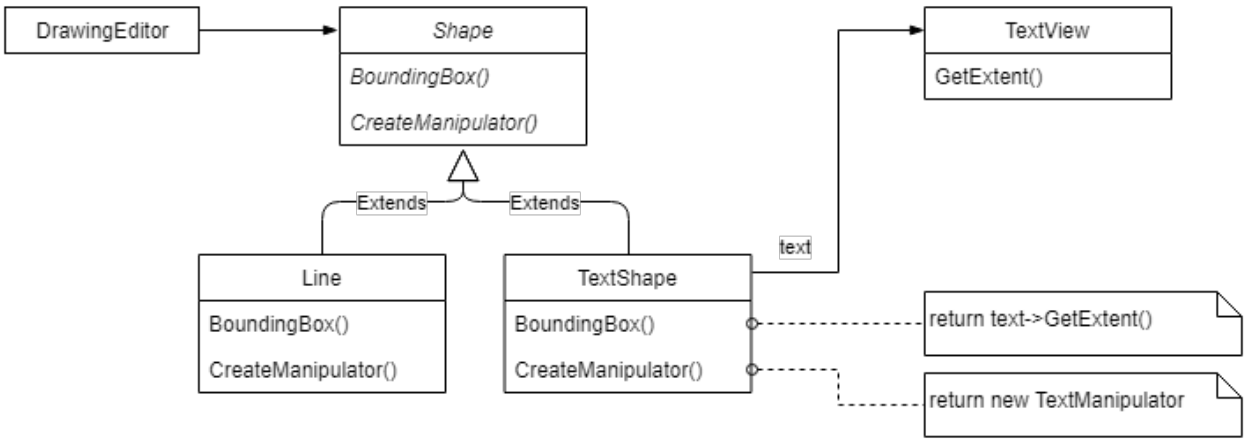
텍스트를 담당하는 TextShape은 이미 TextView에 의한 유사하고 복잡한 기능이 존재하므로 재활용하면 좋을 듯 하다.

(2) 문제:

Shape 클래스와 TextView는 서로 다른 동기에 의해 개발되어 수정하지 않는한 사용이 불가하다.

(3) solution:

그러기 위해서 중간에 TextShape클래스를 adapter로 TextView를 adaptee로 패턴화하여 개발하면 된다.



Example 2

https://github.com/iluwatar/java-design-patterns/tree/master/adapter

```
public interface RowingBoat {
    void row();
}
```

```
public class FishingBoat {
    public void sail() {
        System.out.println("The fishing boat is sailing.");
    }
}
```

```
public class Captain implements RowingBoat {

    private RowingBoat rowingBoat;

    public Captain(RowingBoat rowingBoat) {
        this.rowingBoat = rowingBoat;
    }

    @Override
    public void row() {
        rowingBoat.row();
    }
}
```

```
public class FishingBoatAdapter implements RowingBoat {

    private FishingBoat boat;

    public FishingBoatAdapter() {
        boat = new FishingBoat();
    }

    @Override
    public void row() {
        boat.sail();
    }
}
```

```
class AdapterApp {
    public static void main(String[] args) {
        Captain captain = new Captain(new FishingBoatAdapter());
        captain.row();
    }
}
```

JDK에서의 Adapter Pattern 사례

java.io.InputStreamReader

https://yaboong.github.io/design-pattern/2018/10/15/adapter-pattern/

관련 패턴과의 관계

vs. Bridge Pattern

vs. Decorator Pattern

vs. Proxy Pattern

http://www.jidum.com/jidums/view.do?jidumId=991