고급객체지향 프로그래밍
강의노트 #09

Adapter, Facade Pattern

조용주
ycho@smu.ac.kr

어댑터 패턴 (Adapter Pattern)

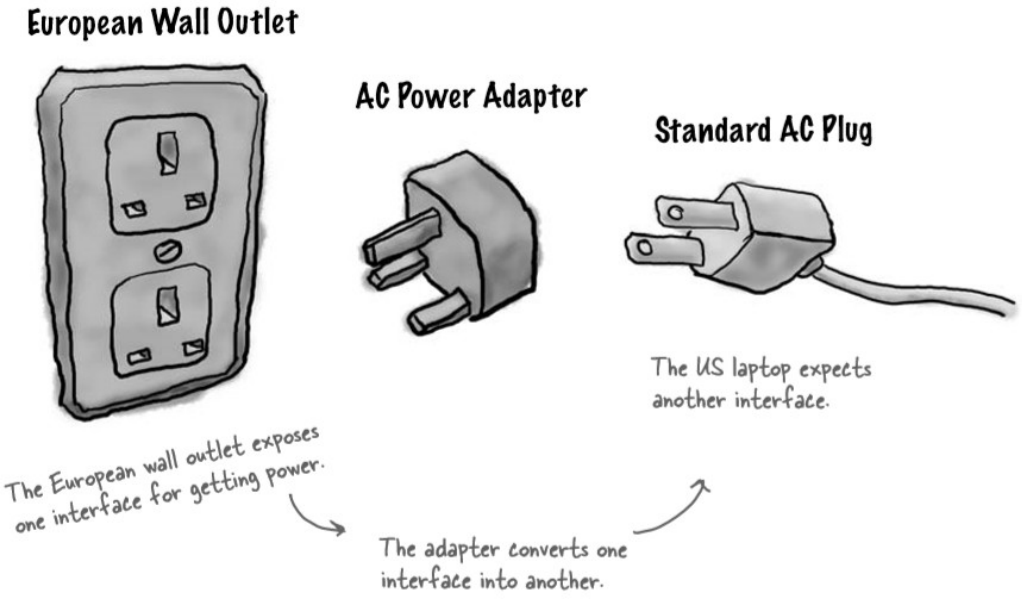
- 목적
  - Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.
  - Also known as Wrapper
  - 클래스의 인터페이스를 클라이언트가 원하는 형태의 또 다른 인터페이스로 변환. 어댑터는 호환되지 않는 인터페이스 때문에 동작하지 않는 클래스들을 함께 동작할 수 있도록 만들어줌

퍼사드 패턴 (Façade Pattern)

- 목적
  - Provide a unified interface to a set of interfaces in a sub system. Façade defines a higher-level interface that mak es the subsystem easier to use.
  - 서브시스템에 있는 여러 개의 인터페이스를 통합하는 한 개의 인터페이스를 제공. 퍼사드는 서브 시스템을 쉽게 사용할 수 있도록 해주는 고급 수준의 인터페이스를 정의

어댑터 패턴

- 객체를 감싸는 역할을 함 (Object wrapping)
  - 서로 호환되지 않는 두 개 인터페이스를 연결하는 작업
  - 서로 다른 인터페이스를 동일하게 변환
- 예: 전기 플러그
  - 서로 다른 플러그(유럽, 미국)의 경우 어댑터를 사용해서 변환시킬 수 있음



어댑터 패턴

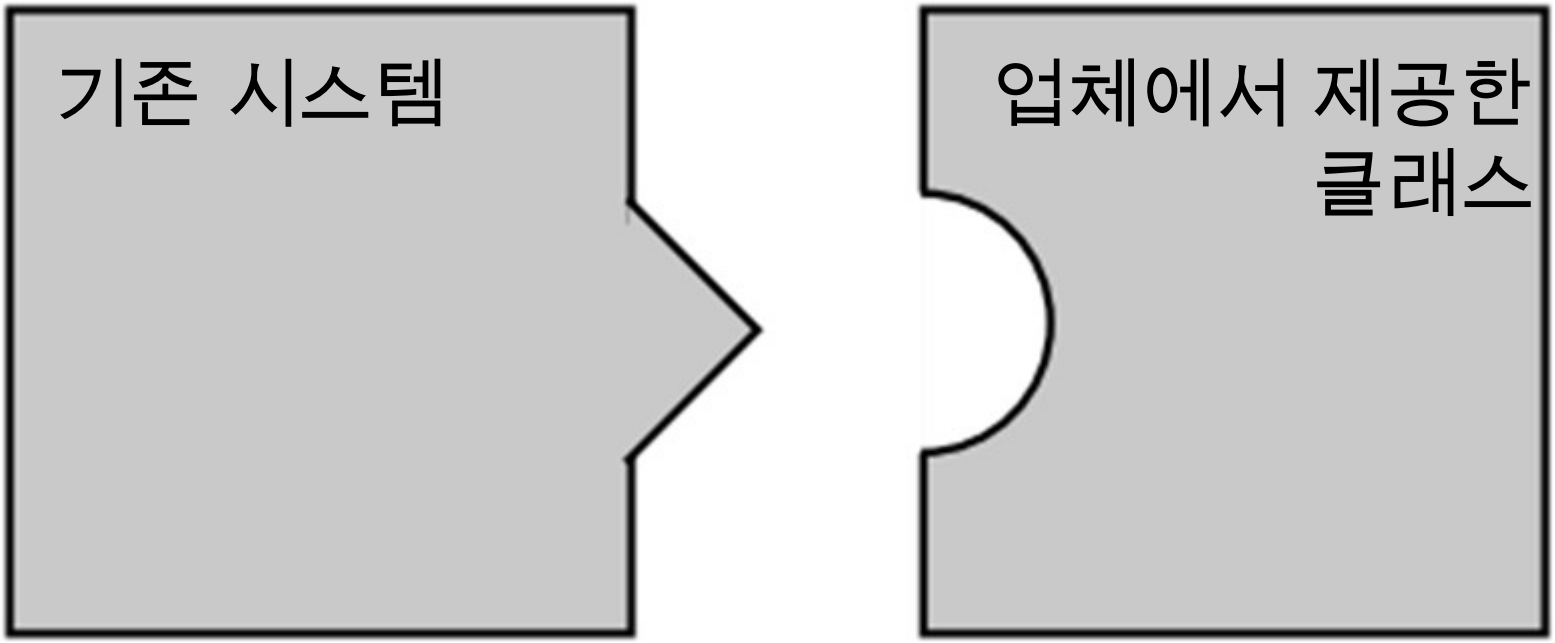
- 객체지향 어댑터
  - 일상 생활에서 쓰이는 어댑터하고 똑같은 역할을 함
  - 어떤 인터페이스를 클라이언트에서 요구하는 형태의 인터페이스에 적응시켜주는 역할을 함

디자인 패턴 요소

요소	설명
이름	어댑터 (Adapter)
문제	사용 객체의 API 가 서로 다름
해결방안	함수를 변환하는 객체를 중간에 넣음
결과	변경 최소화

객체지향 어댑터

- 기존 소프트웨어 시스템에서 새로운 업체에서 제공한 클래스 라이브러리를 사용해야 한다고 가정
- 새로 채택한 업체에서 사용하는 인터페이스가 기존 업체에서 사용하던 인터페이스와 다르다면 ?

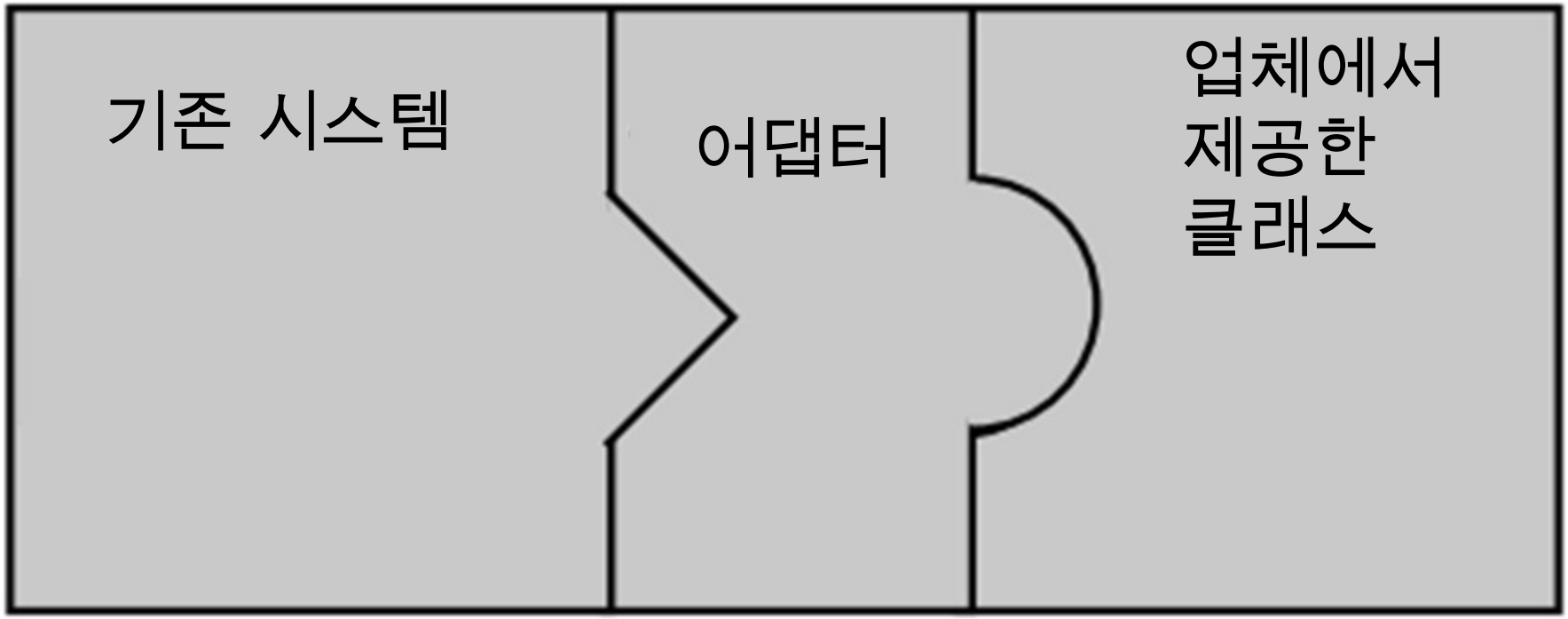


객체지향 어댑터

- 기존 코드를 바꿀 수 없다면, 새로 사용하기로 한 업체에서 사용하는 인터페이스를 기존에 사용하던 인터페이스에 적응시켜주는 클래스를 만들어서 사용 가능

객체지향 어댑터

- 어댑터는 클라이언트로부터 요청을 받아서 새로운 업체에서 제공하는 클래스에서 받아들일 수 있는 형태의 요청으로 변환시켜주는 중개인 역할을 함



사례 1 오리 탈을 쓴 칠면조

- 다음 코드는 예전에 봤던 내용

```
public interface Duck {
    public void quack();
    public void fly();
}

public class MallardDuck implements Duck {
    public void quack() {
        System.out.println("Quack");
    }
    public void fly() {
        System.out.println("I'm flying");
    }
}
```

사례 1 오리 탈을 쓴 칠면조

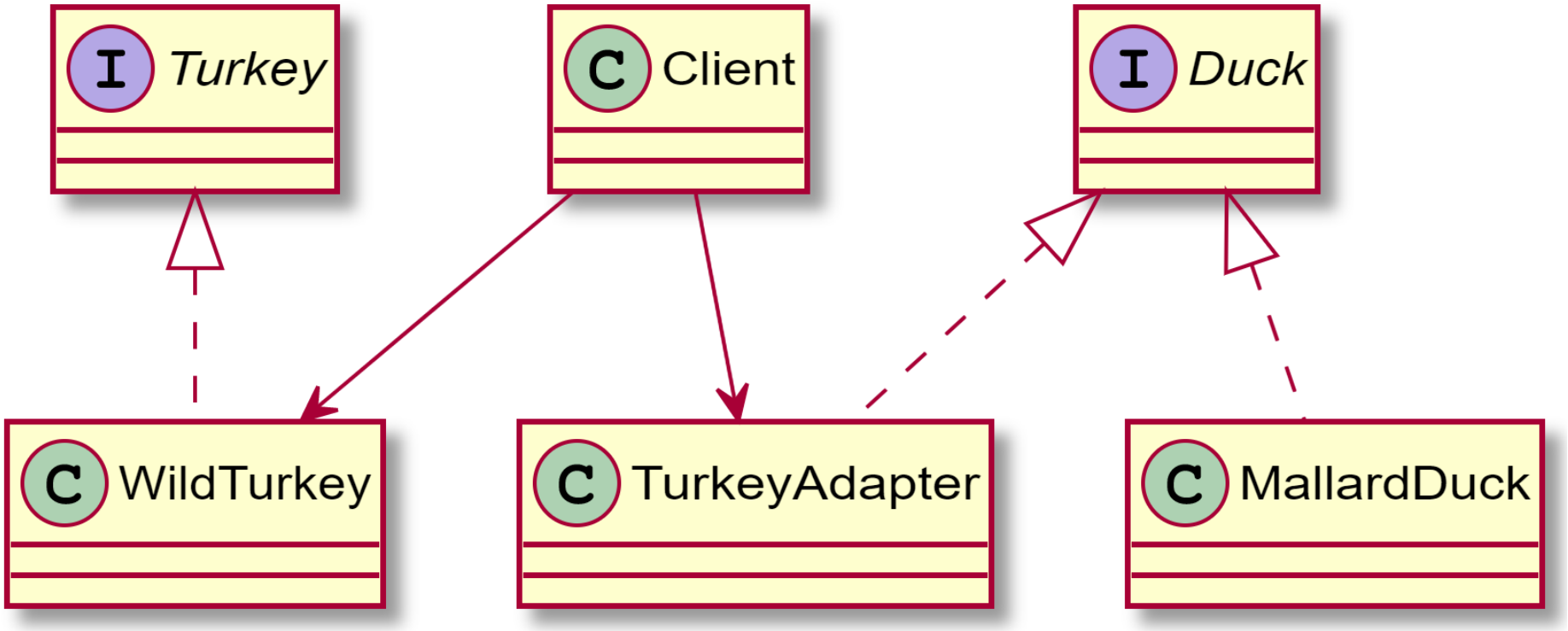
```
public interface Turkey {
    public void gobble();
    public void fly();
}

public class WildTurkey implements Turkey {
    public void gobble() {
        System.out.println("Gobble gobble");
    }
    public void fly() {
        System.out.println("I'm flying a
short distance");
    }
}
```

사례 1 오리 탈을 쓴 칠면조

- Duck 객체가 모자라서 Turkey 객체를 대신 사용해야 하는 상황이라고 가정
- 어댑터 구현

```
public class TurkeyAdapter implements Duck {
    Turkey turkey;
    public TurkeyAdapter(Turkey turkey) {
        this.turkey = turkey;
    }
    public void quack() {
        turkey.gobble();
    }
    public void fly() {
        for (int i = 0; i < 5; i++) {
            turkey.fly();
        }
    }
}
```

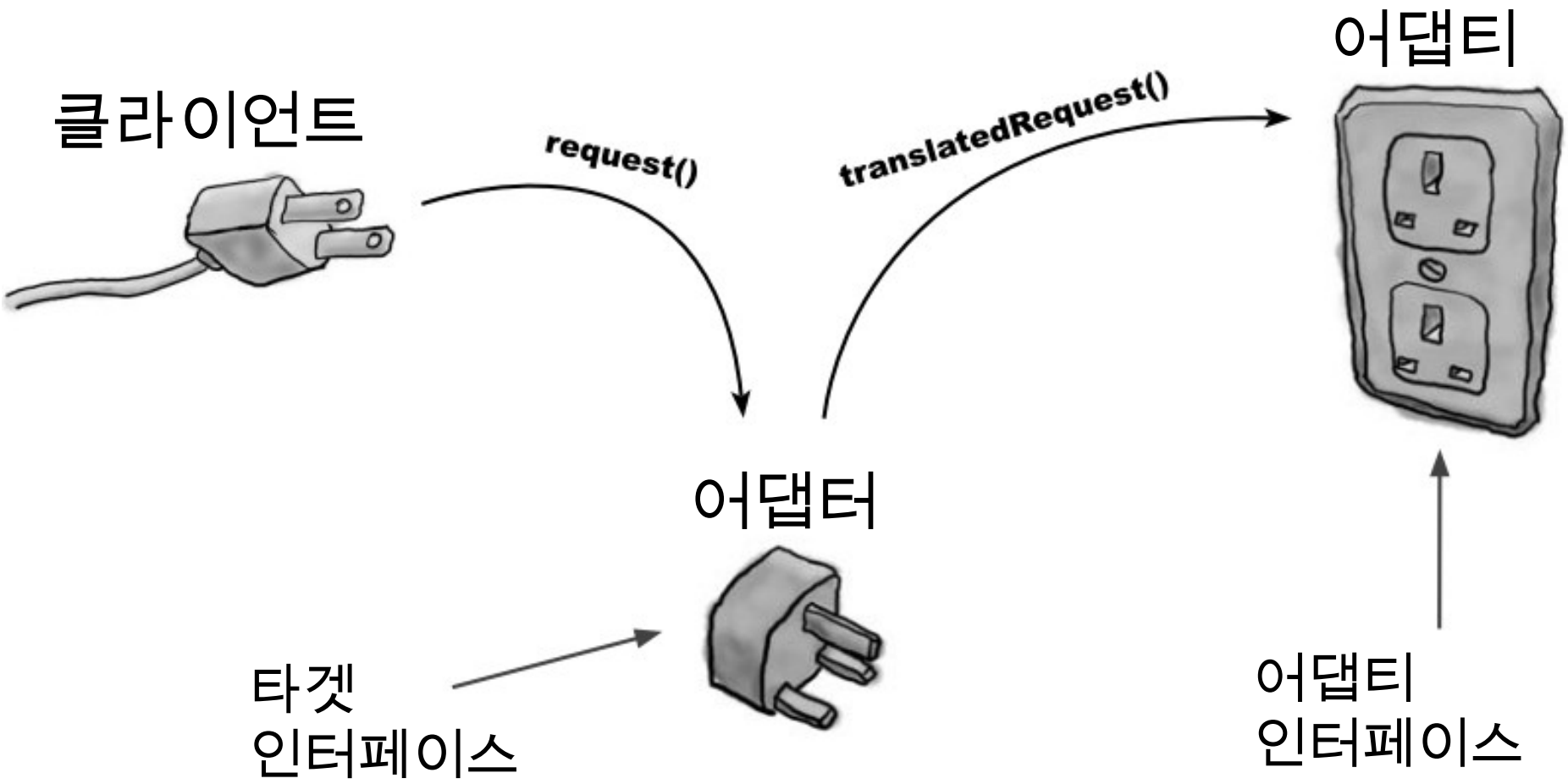
사례 1 오리 탈을 쓴 칠면조



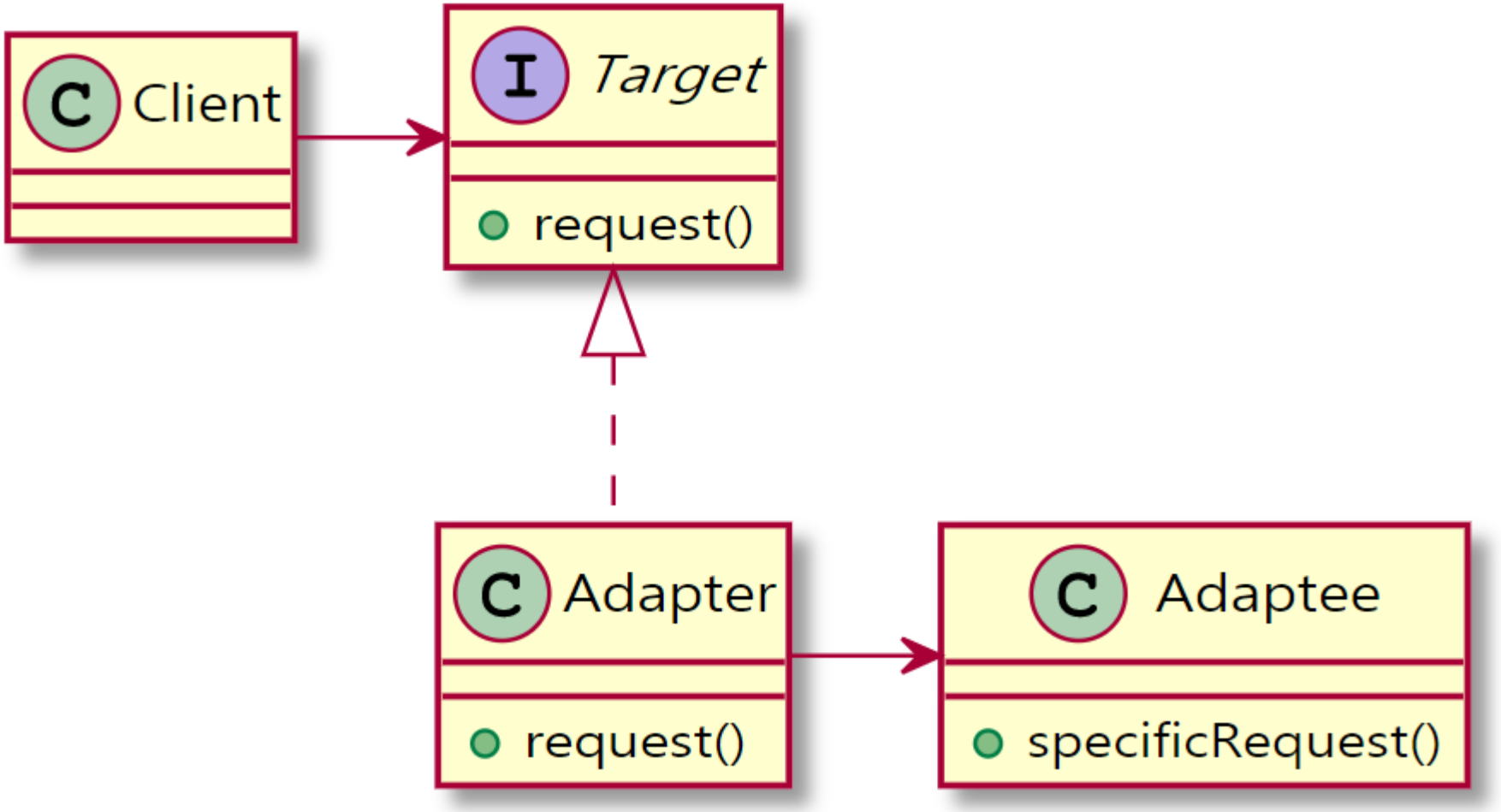
사례 1 오리 탈을 쓴 칠면조

```
public class DuckTestDrive {
  public static void main(String[] args) {
    MallardDuck duck = new MallardDuck();
    WildTurkey turkey = new WildTurkey();
    Duck turkeyAdapter = new TurkeyAdapter(turkey);

    System.out.println("The Turkey says…");
    turkey.gobble();
    turkey.fly();
    System.out.println("\nThe Duck says…");
    testDuck(duck);
    System.out.println("\nThe TurkeyAdapter says…");
    testDuck(turkeyAdapter);
  }
  static void testDuck(Duck duck) {
    duck.quack();
    duck.fly();
  }
}
```

어댑터 패턴


클라이언트
request()
translatedRequest()
어댑티
어댑터
타겟
인터페이스
어댑티
인터페이스

어댑터 패턴 정의


Client
Target
request()
Adapter
request()
Adaptee
specificRequest()

사례 2 Enumeration 을 Iterator 에 적응시키기

- Enumeration 예제 코드

```
import java.util.*;

public class Enumeration1 {
  public static void printEnumeration(Enumeration e) {
    while (e.hasMoreElements()) {
      System.out.println("" + e.nextElement());
    }
  }
  public static void main(String[] args) {
    Vector v = new Vector();
    for (int i = 0; i < 10; i++) {
      v.add(i);
    }
    Enumeration e = v.elements();
    Enumeration1.printEnumeration(e);
  }
}
```

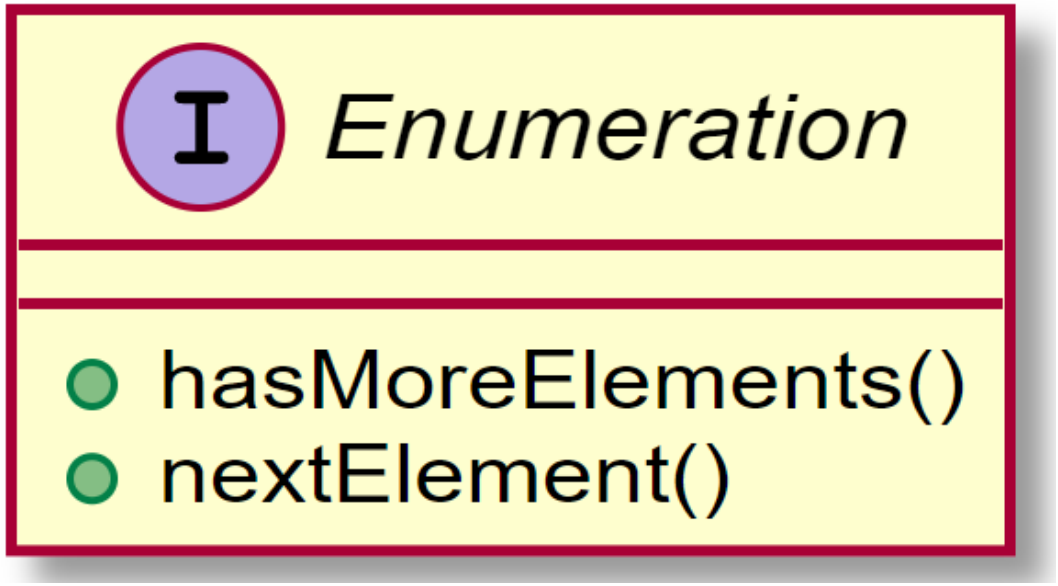
사례 2 Enumeration 을 Iterator 에 적응시키기

- Iterator 예제 코드

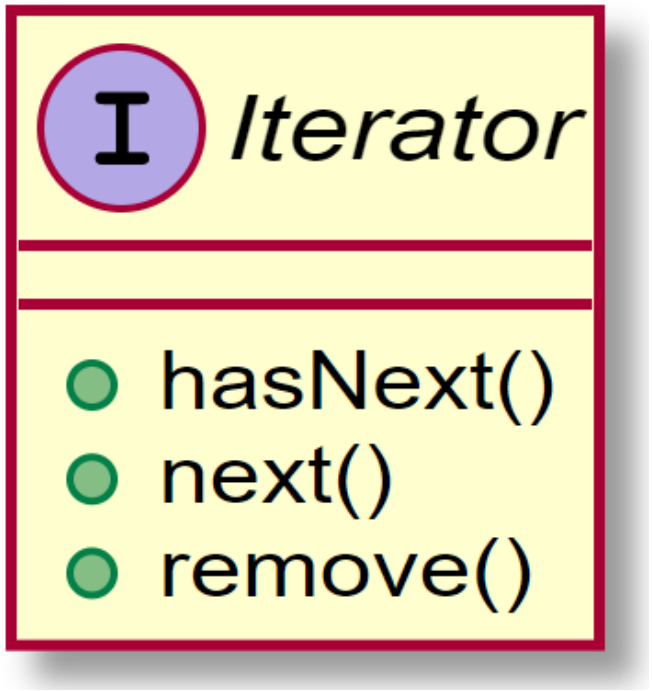
```
import java.util.*;

public class Iterator1 {
  public static void printIterator(Iterator it) {
    while (it.hasNext()) {
      System.out.println("" + it.next());
    }
  }
  public static void main(String[] args) {
    Vector v = new Vector();
    for (int i = 0; i < 10; i++) {
      v.add(i);
    }
    Iterator it = v.iterator();
    Iterator1.printIterator(it);
  }
}
```

사례 2 Enumeration 을 Iterator 에 적응시키기

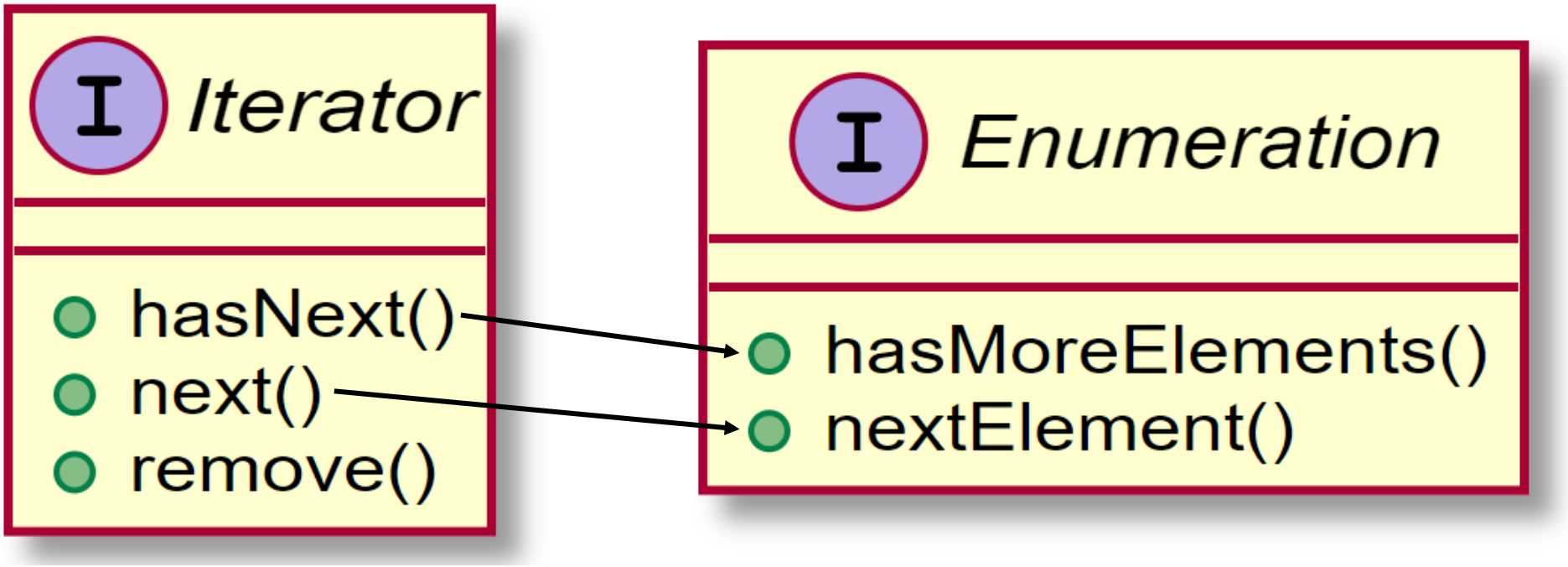
- Enumeration



- Iterator

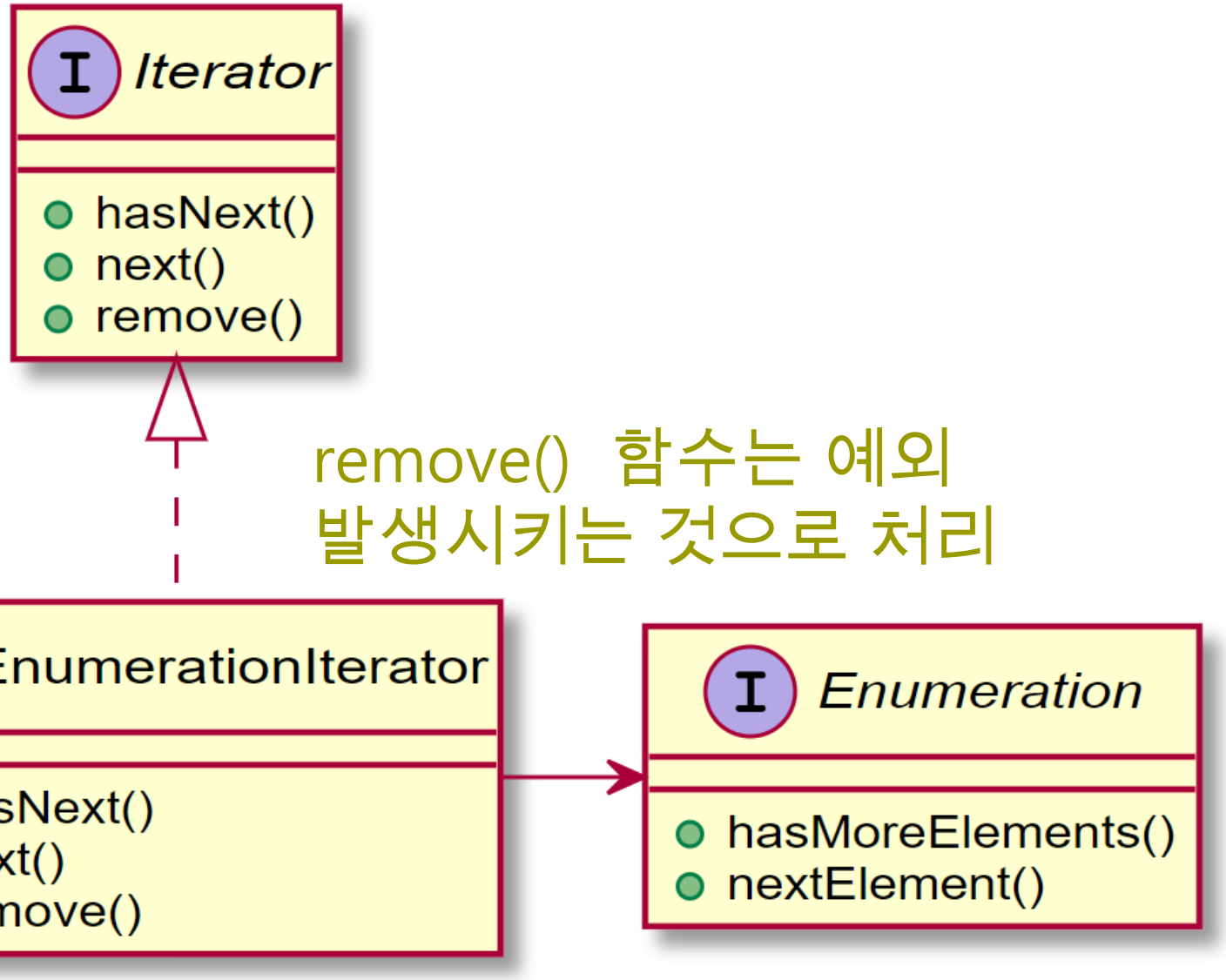


사례 2 Enumeration 을 Iterator 에 적응시키기



- Iterator – 타겟 인터페이스
- Enumeration – 어댑티 (Adaptee) 인터페이스
- remove() 메소드는 어떻게 ?

사례 2 Enumeration 을 Iterator 에 적응시키기



사례 2 Enumeration 을 Iterator 에 적응시키기

```
public class EnumerationIterator implements Iterator {
  Enumeration enumeration;
  public EnumerationIterator(Enumeration enmt) {
    this.enumeration = enmt;
  }
  public boolean hasNext() {
    return enumeration.hasMoreElements();
  }
  public Object next() {
    return enumeration.nextElement();
  }
  public void remove() {
    throw new UnsupportedOperationException();
  }
}
```

```
public class Iterator2 {
  public static void printIterator(Iterator it) {
    while (it.hasNext()) {
      System.out.println("" + it.next());
    }
  }

  public static void main(String[] args) {
    Vector v = new Vector();
    for (int i = 0; i < 10; i++) {
      v.add(i);
    }
    Enumeration e = v.elements();
    EnumerationIterator it = new EnumerationIterator(e);
    Iterator2.printIterator(it);
  }
}
```

어댑터 사용 예

- 자바에서 배열을 고정 크기의 리스트로 변환
  - Arrays.asList() 함수 사용
  - 변환된 리스트는 ArraysAdapter 로서 Arrays 의 특징을 가지게 됨
    - 리스트로 변환하더라도 고정 크기이므로 add(), remove() 사용 불가
    - List 의 set() 함수를 이용해서 요소 내용 변경 가능 (단 원래 Arrays 의 내용도 변경됨)

어댑터 사용 예

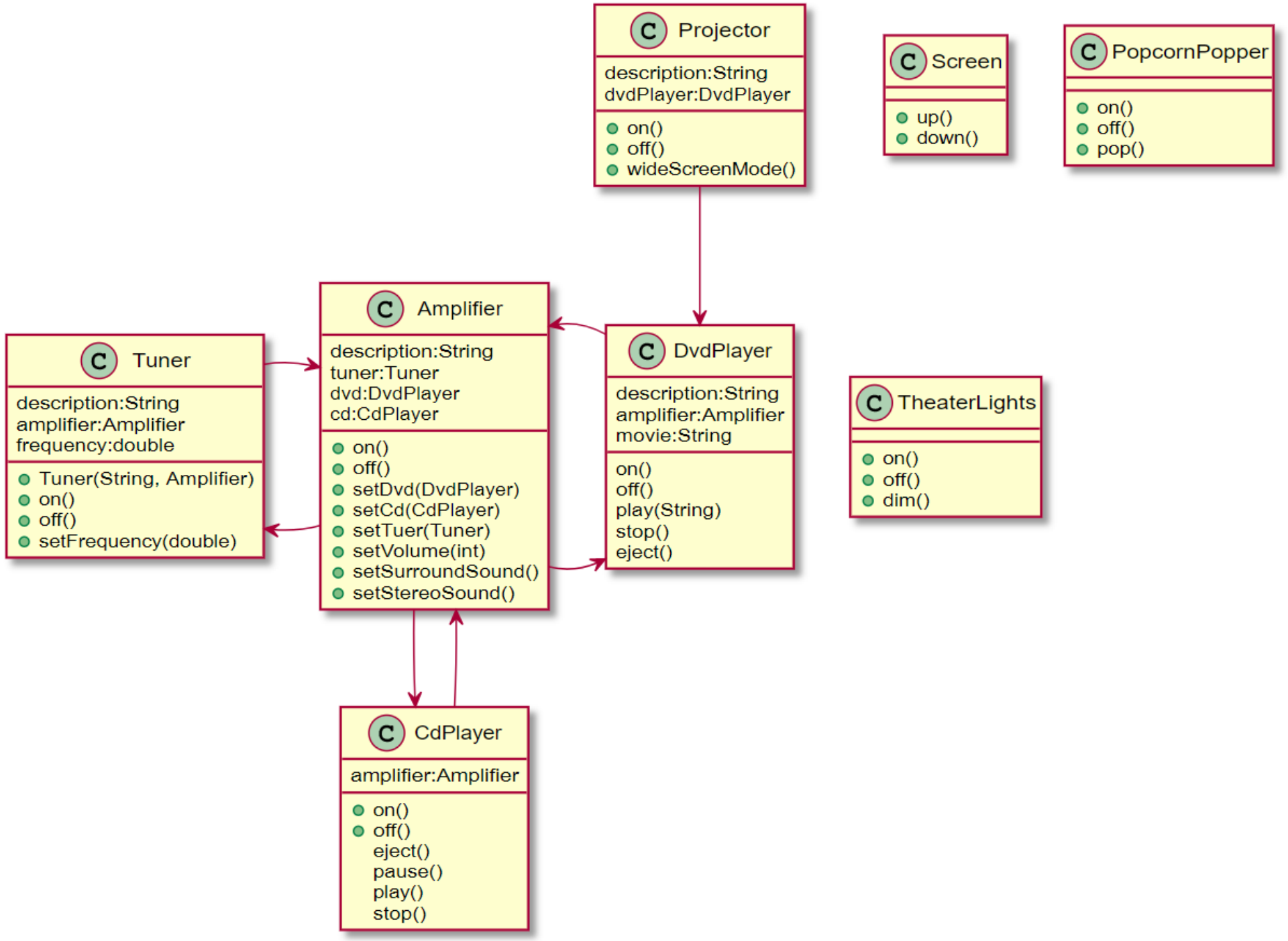
```
import java.util.Arrays;
import java.util.List;

public class ArraysAdapter {
  public static void main(String[] args) {
    String[] cities = { "Seoul", "Incheon", "Busan",
"Sejong" };
    List<String> cityList = Arrays.asList(cities);
    System.out.printf("cities.length = %d\n",
cities.length);
    System.out.printf("cityList.size = %d\n",
cityList.size());
```

어댑터 사용 예

```
    cityList.set(0, "Suwon");
    System.out.println("\mPrint out cities");
    for (String s : cities) {
      System.out.println(s);
    }
    System.out.println("\mPrint out cityList");
    for (String s : cityList) {
      System.out.println(s);
    }
  }
}
```

홈 씨어터

- 홈 씨어터 (Home Theater) 구축 가정
  - DVD 플레이어, 프로젝터, 자동 스크린, 서라운드 음향 시스템 및 팝콘 기계를 갖춘 시스템을 구성
- 영화 보기 (복잡한 방법)
  - 팝콘 기계를 켜고 튀기기 시작
  - 전등을 어둡게 조절, 스크린을 내림
  - 프로젝터를 켜고 프로젝터로 DVD 신호 입력
  - 프로젝터를 와이드 스크린 모드로 전환
  - 앰프를 켜고 DVD 로 전환
  - 앰프를 서라운드 음향 모드로 전환
  - 앰프 볼륨을 중간 (5) 로 설정
  - DVD 플레이어를 켜고 재생 시작


Projector
description:String
dvdPlayer:DvdPlayer
on()
off()
wideScreenMode()
Screen
up()
down()
PopcornPopper
on()
off()
pop()
Amplifier
description:String
tuner:Tuner
dvd:DvdPlayer
cd:CdPlayer
on()
off()
setDvd(DvdPlayer)
setCd(CdPlayer)
setTuer(Tuner)
setVolume(int)
setSurroundSound()
setStereoSound()
Tuner
description:String
amplifier:Amplifier
frequency:double
Tuner(String, Amplifier)
on()
off()
setFrequency(double)
DvdPlayer
description:String
amplifier:Amplifier
movie:String
on()
off()
play(String)
stop()
eject()
TheaterLights
on()
off()
dim()
CdPlayer
amplifier:Amplifier
on()
off()
eject()
pause()
play()
stop()

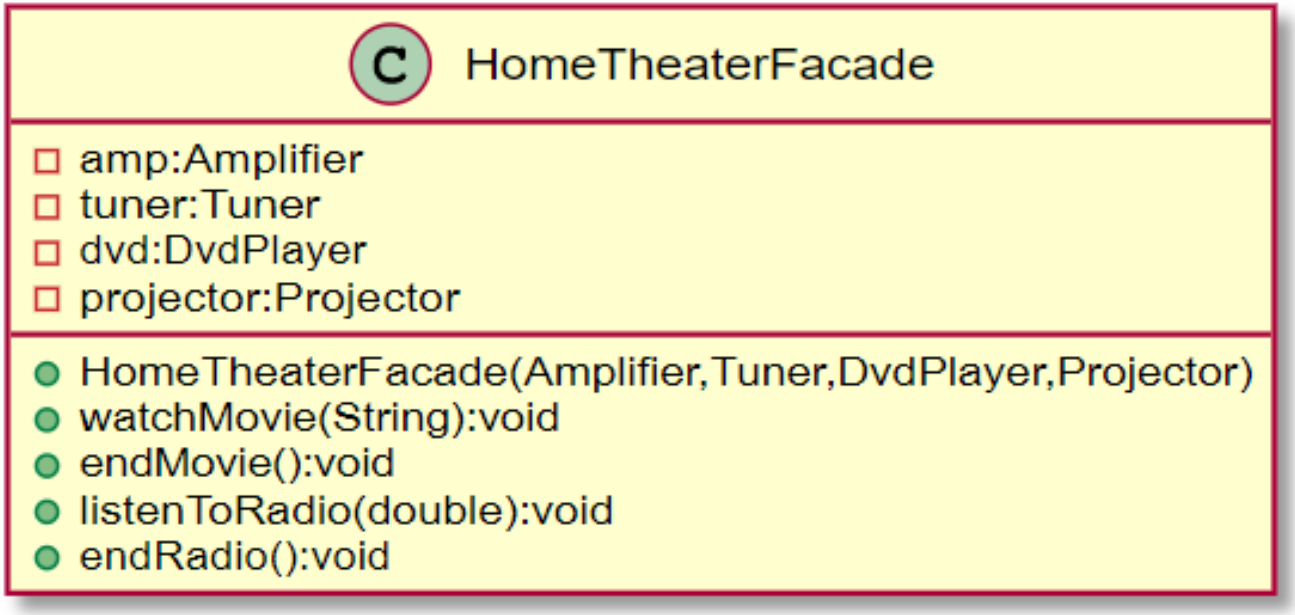
홈 씨어터

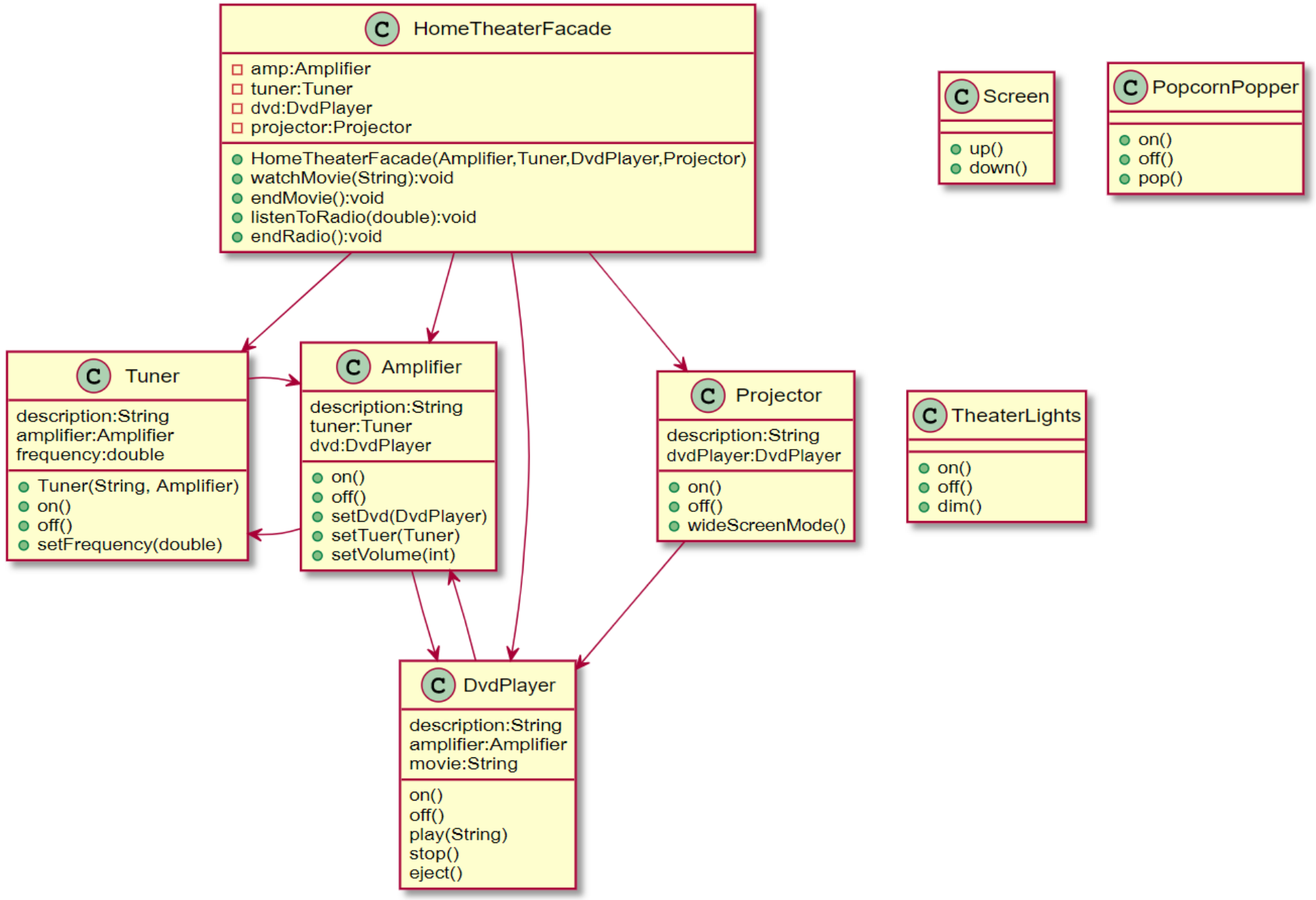
- 영화 보기를 코드로 구현

```
popper.on();
popper.pop();
lights.dim(10);
screen.down();
projector.on();
projector.setInput(dvd);
projector.wideScreenMode();
amp.on();
amp.setDvd(dvd);
amp.setSurroundSound();
amp.setVolume(5);
dvd.on();
dvd.play(movie);
```

퍼사드 패턴을 이용하는 홈 씨어터

- 퍼사드 패턴을 이용해서 쓰기 쉬운 인터페이스를 제공
- 복잡한 시스템을 직접 건드리고 싶다면 기존 인터페이스 사용 가능
- 퍼사드 클래스 제공




HomeTheaterFacade
amp:Amplifier
tuner:Tuner
dvd:DvdPlayer
projector:Projector
HomeTheaterFacade(Amplifier,Tuner,DvdPlayer,Projector)
watchMovie(String):void
endMovie():void
listenToRadio(double):void
endRadio():void
Screen
up()
down()
PopcornPopper
on()
off()
pop()
Tuner
description:String
amplifier:Amplifier
frequency:double
Tuner(String, Amplifier)
on()
off()
setFrequency(double)
Amplifier
description:String
tuner:Tuner
dvd:DvdPlayer
on()
off()
setDvd(DvdPlayer)
setTuer(Tuner)
setVolume(int)
Projector
description:String
dvdPlayer:DvdPlayer
on()
off()
wideScreenMode()
TheaterLights
on()
off()
dim()
DvdPlayer
description:String
amplifier:Amplifier
movie:String
on()
off()
play(String)
stop()
eject()

퍼사드 패턴을 이용하는 홈 씨어터

- 홈 씨어터 시스템용 퍼사드 클래스 제공
  - watchMovie() 같이 몇 가지 간단한 메소드를 제공
- 퍼사드 클래스에서는 홈 씨어터 구성요소들을 하나의 서브시스템으로 간주하고 , watchMovie() 메소드에서는 서브시스템의 메소드들을 호출하여 필요한 작업을 처리
- 클라이언트는 서브시스템이 아닌 홈 씨어터 퍼사드에 있는 메소드를 호출
- 퍼사드를 쓰더라도 서브시스템에는 여전히 직접 접근 가능
  - 서브시스템 클래스의 고급 기능이 필요하다면 사용 가능

```
public class HomeTheaterFacade {
  private Amplifier amp;
  private Tuner tuner;
  private DvdPlayer dvd;
  private CdPlayer cd;
  private Projector projector;
  private TheaterLights lights;
  private Screen screen;
  private PopcornPopper popper;

  public HomeTheaterFacade(Amplifier a, Tuner t,
      DvdPlayer d, CdPlayer c, Projector p,
      Screen s, TheaterLights l, PopcornPopper pp) {
    this.amp = a; this.tuner = t; this.dvd = d;
    this.cd = c; this.projector = p; this.lights = l;
    this.screen = s; this.popper = pp;
  }
```

```
public void watchMovie(String movie) {
  System.out.println("Get ready to watch a movie...");
  popper.on();
  popper.pop();
  lights.dim(10);
  screen.down();
  projector.on();
  projector.wideScreenMode();
  amp.on();
  amp.setDvd(dvd);
  amp.setVolume(5);
  dvd.on();
  dvd.play(movie);
}
```

```
  public void endMovie() {
    System.out.println("Shutting movie theater down..");
    popper.off();
    lights.on();
    screen.up();
    projector.off();
    amp.off();
    dvd.stop();
    dvd.eject();
    dvd.off();
  }
  … //  기타 메소드
}
```

퍼사드 패턴 사용

```
public class HomeTheaterTestDrive {
  public static void main(String[] args) {
    //  컴포넌트 객체 생성 코드
    HomeTheaterFacade homeTheater
     = new HomeTheaterFacade(amp, tuner, dvd, cd,
              projector, screen, lights, popper);
    homeTheater.watchMovie("Raiders of the Lost Ark");
    homeTheater.endMovie();
  }
}
```

디자인 패턴 요소

요소	설명
이름	퍼사드 (Façade)
문제	서브시스템이 너무 많고 사용하기가 복잡함
해결방안	단순한 인터페이스를 제공하는 객체를 중간에 넣음
결과	최소 지식 원칙에 입각해 의존성 최소화

- 퍼사드 패턴
  - 어떤 서브시스템의 일련의 인터페이스에 대한 통합된 인터페이스 제공
  - 퍼사드에서 고수준 인터페이스를 정의하기 때문에 서브시스템을 더 쉽게 사용할 수 있음

퍼사드 패턴의 정의