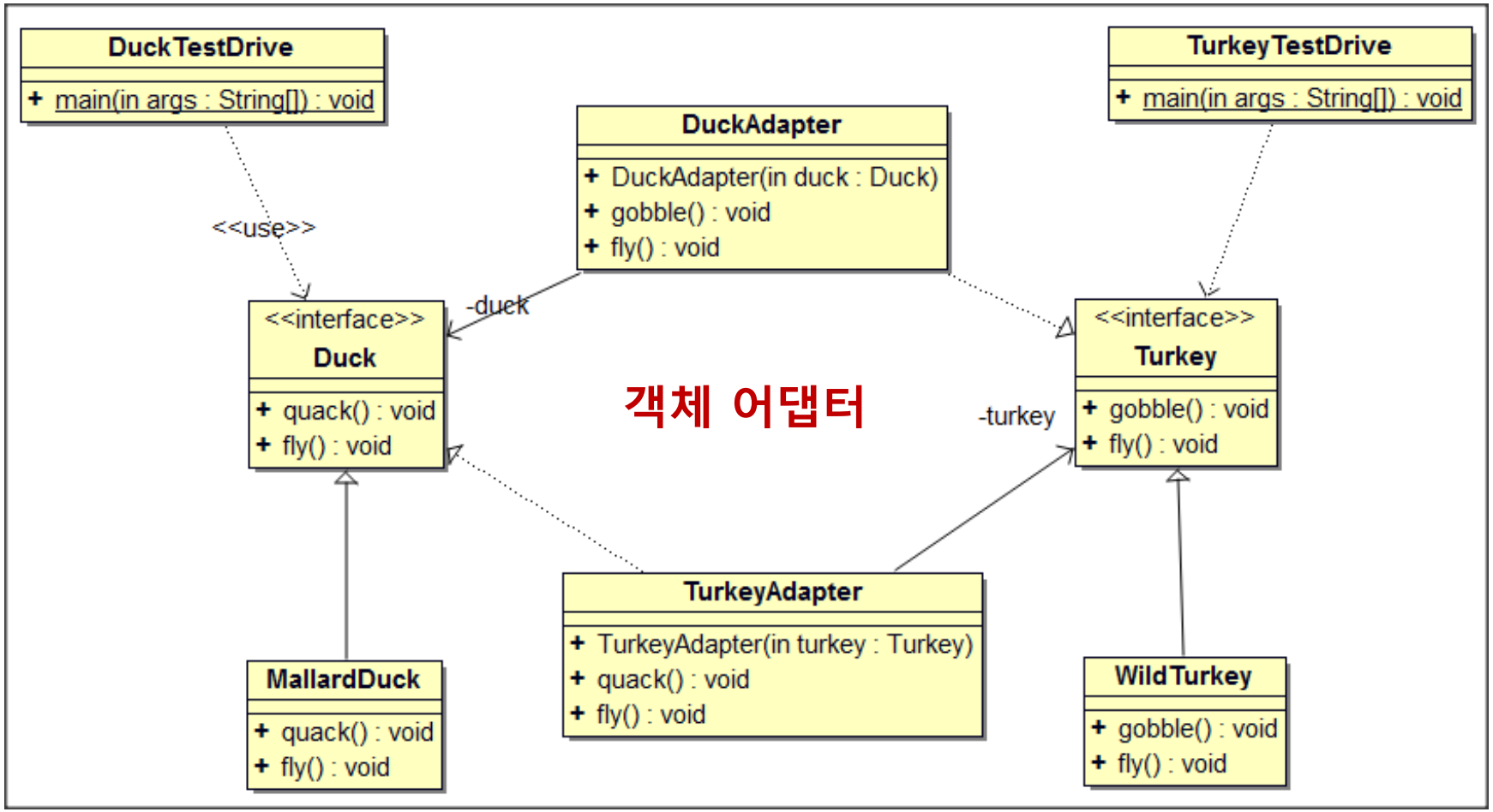
7장. 어댑터 패턴과 퍼사드 패턴

동의대학교 컴퓨터소프트웨어공학과

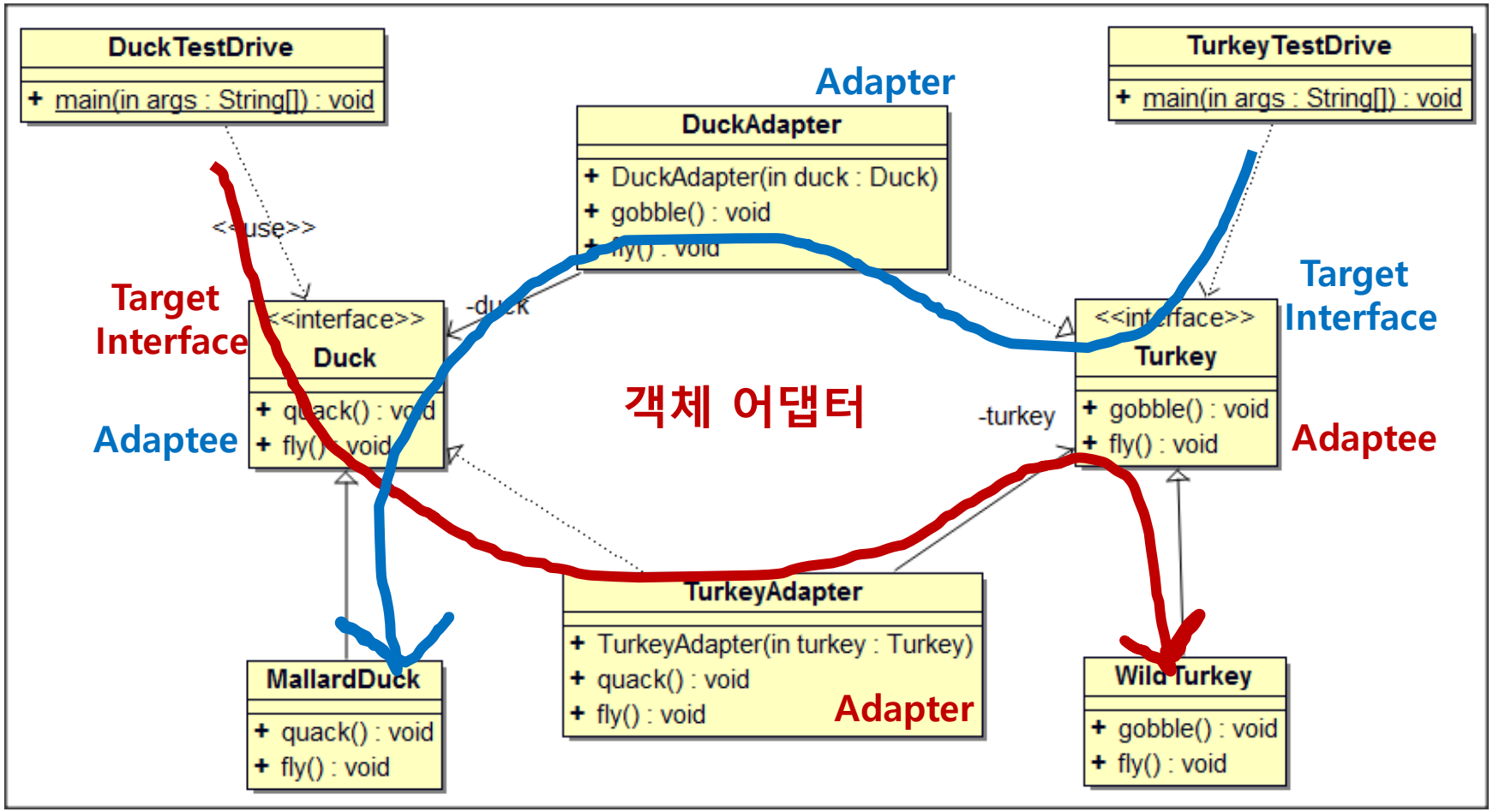
학습 목표

- **어댑터 패턴**의 개념을 이해하고 활용할 수 있다.
- **퍼사드 패턴**의 개념을 이해하고 활용할 수 있다.
- **최소 지식 원칙**의 개념을 이해하고 이를 준수하기 위한 가이드라인을 활용할 수 있다.

클래스 다이어그램: adapter



Target Interface-Adapter-Adaptee

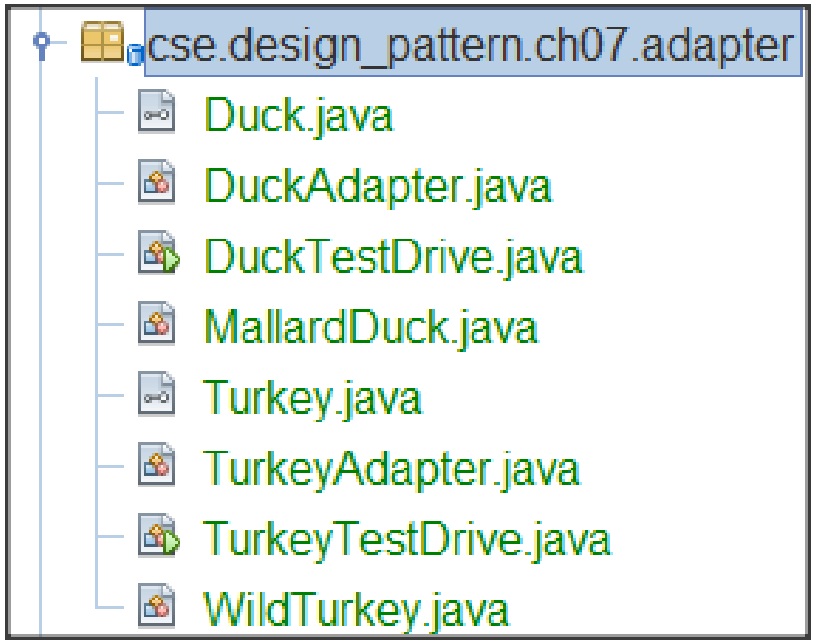


현재 상황

- DuckTestDrive에서 Duck 객체가 필요한데 부족한 상황
  → Turkey 객체를 마치 Duck 객체처럼 보이도록 TurkeyAdapter를 사용

- TurkeyTestDrive에서 Turkey 객체가 필요한데 부족한 상황
  → Duck 객체를 마치 Turkey 객체처럼 보이도록 DuckAdapter를 사용

실습: adapter


cse.design_pattern.ch07.adapter
Duck.java
DuckAdapter.java
DuckTestDrive.java
MallardDuck.java
Turkey.java
TurkeyAdapter.java
TurkeyTestDrive.java
WildTurkey.java

Duck.java

```java
package cse.design_pattern.ch07.adapter;

public interface Duck {

    void quack();

    void fly();

}
```

Turkey.java

```java
package cse.design_pattern.ch07.adapter;

public interface Turkey {

    void gobble();

    void fly();

}
```

MallardDuck.java WildTurkey.java

```java
package cse.design_pattern.ch07.adapter;

public class MallardDuck implements Duck {

    public void quack() {
        System.out.println("꽥");
    }

    public void fly() {
        System.out.println("날고 있어요!");
    }

}
```

```java
package cse.design_pattern.ch07.adapter;

public class WildTurkey implements Turkey {

    public void gobble() {
        System.out.println("꼬르륵");
    }

    public void fly() {
        System.out.println("조금만 날 수 있어요!");
    }

}
```

TurkeyAdapter.java

```
1   package cse.design_pattern.ch07.adapter;
2
3   public class TurkeyAdapter implements Duck {
4
6       private Turkey turkey;
7
7       public TurkeyAdapter(Turkey turkey) {
8           this.turkey = turkey;
9       }
10
12      public void quack() {
13          turkey.gobble();
14      }
16      public void fly() {
17          for (int i = 0; i < 5; i++) {
18              turkey.fly();
19          }
20      }
21  }
```

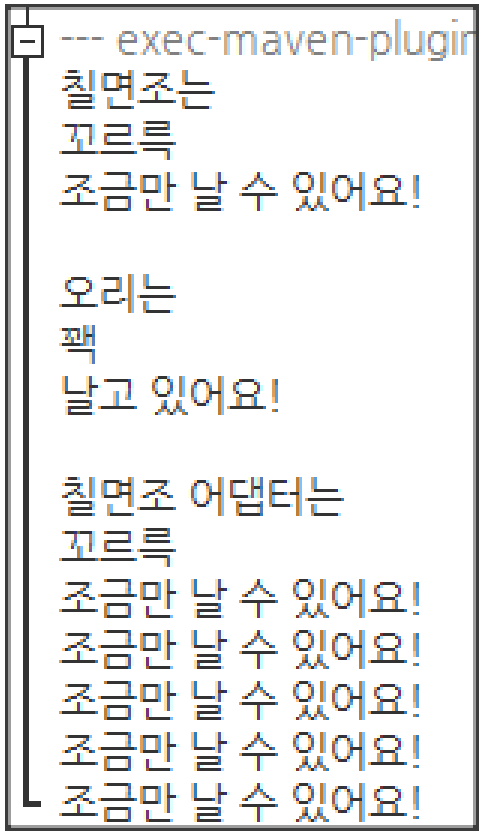
DuckAdapter.java

```
1   package cse.design_pattern.ch07.adapter;
2
3   import java.util.Random;
4
5   public class DuckAdapter implements Turkey {
6
7       private Duck duck;
8       Random random;
9
10      public DuckAdapter(Duck duck) {
11          this.duck = duck;
12          random = new Random();
13      }
14
15      public void gobble() {
16          duck.quack();
17      }
18
19      public void fly() {
20          if (random.nextInt(5) == 0) {
21              duck.fly();
22          }
23      }
24
25  }
```

DuckTestDrive.java

```
6   package cse.design_pattern.ch07.adapter;
7
8   /**...4 lines */
12  public class DuckTestDrive {
13
14      /**...3 lines */
17      public static void main(String[] args) {
18          MallardDuck duck = new MallardDuck();
19          WildTurkey turkey = new WildTurkey();
20          Duck turkeyAdapter = new TurkeyAdapter(turkey);
21
22          System.out.println("칠면조는 ");
23          turkey.gobble();
24          turkey.fly();
25
26          System.out.println("₩n오리는 ");
27          testDuck(duck);
28
29          System.out.println("₩n칠면조 어댑터는 ");
30          testDuck(turkeyAdapter);
31      }
32
33      public static void testDuck(Duck duck) {
34          duck.quack();
35          duck.fly();
36      }
37
38  }
```

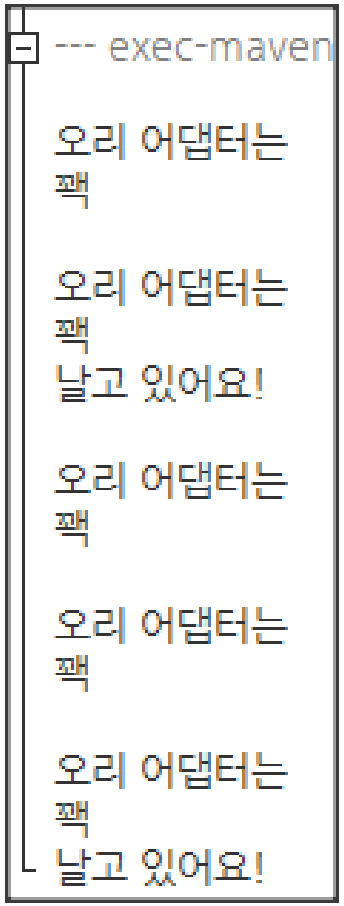
실행 결과


--- exec-maven-plugin
칠면조는
꼬르륵
조금만 날 수 있어요!
오리는
꽥
날고 있어요!
칠면조 어댑터는
꼬르륵
조금만 날 수 있어요!
조금만 날 수 있어요!
조금만 날 수 있어요!
조금만 날 수 있어요!
조금만 날 수 있어요!

TurkeyTestDrive.java

```
6   package cse.design_pattern.ch07.adapter;
7
8   /**...4 lines */
12  public class TurkeyTestDrive {
13
14      /**...3 lines */
17      public static void main(String[] args) {
18          MallardDuck duck = new MallardDuck();
19          Turkey duckAdapter = new DuckAdapter(duck);
20
21          for (int i = 0; i < 5; i++) {
22              System.out.println("₩n오리 어댑터는 ");
23              duckAdapter.gobble();
24              duckAdapter.fly();
25          }
26      }
27
28  }
```

실행 결과


--- exec-maven
오리 어댑터는
꽥
오리 어댑터는
꽥
날고 있어요!
오리 어댑터는
꽥
오리 어댑터는
꽥
오리 어댑터는
꽥
날고 있어요!

어댑터 패턴[Larman01]

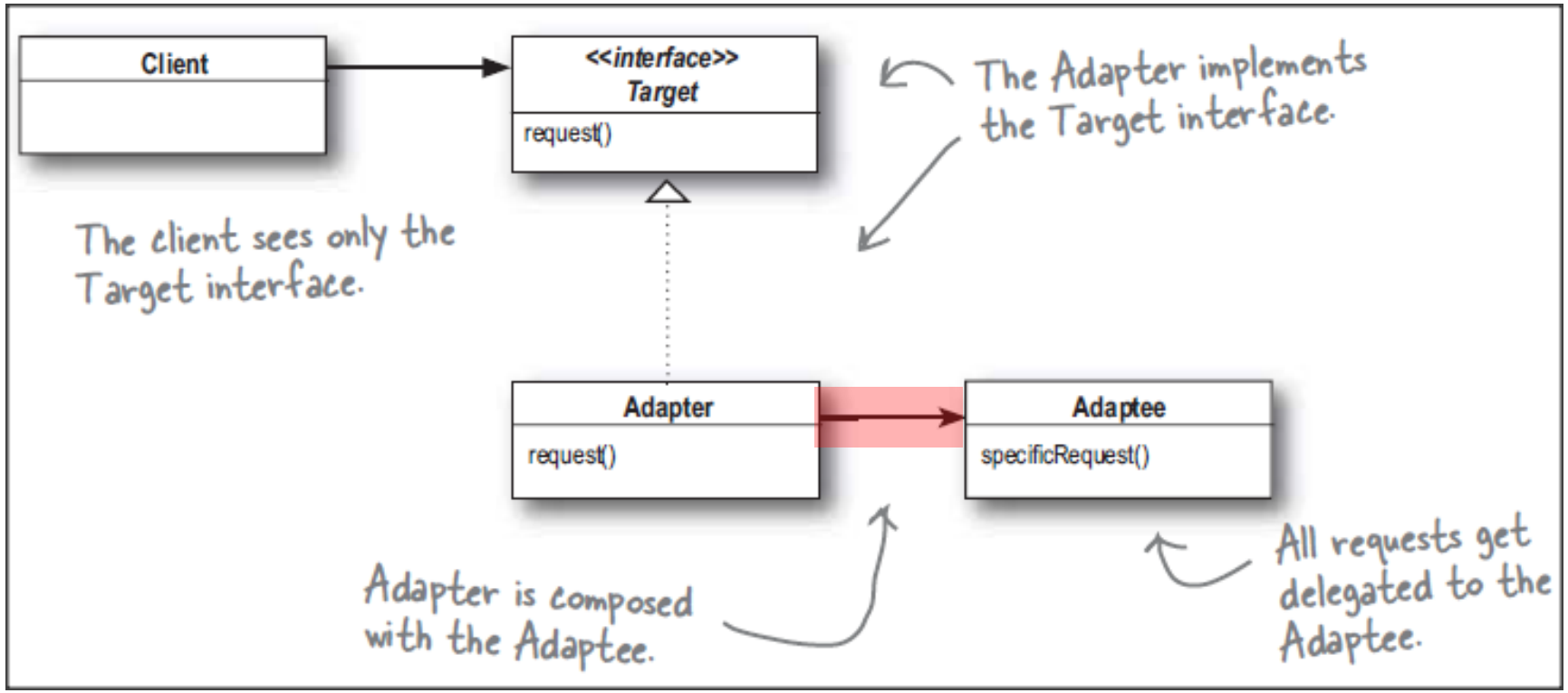
- 문제
  - **호환되지 않는 인터페이스**를 어떻게 해결할 수 있는가? 또는 다른 인터페이스를 가지는 유사한 컴포넌트에 대한 안정적인 인터페이스를 어떻게 제공할 수 있는가?

- 해결 방안
  - 중간에 **어댑터 객체를 사용**하여 컴포넌트의 원래 인터페이스를 다른 인터페이스로 변환한다.

- 어댑터 패턴 구현 방법
  - **객체 어댑터** 이용: 인터페이스 구현
  - **클래스 어댑터** 이용: 클래스 상속 (다중 상속)

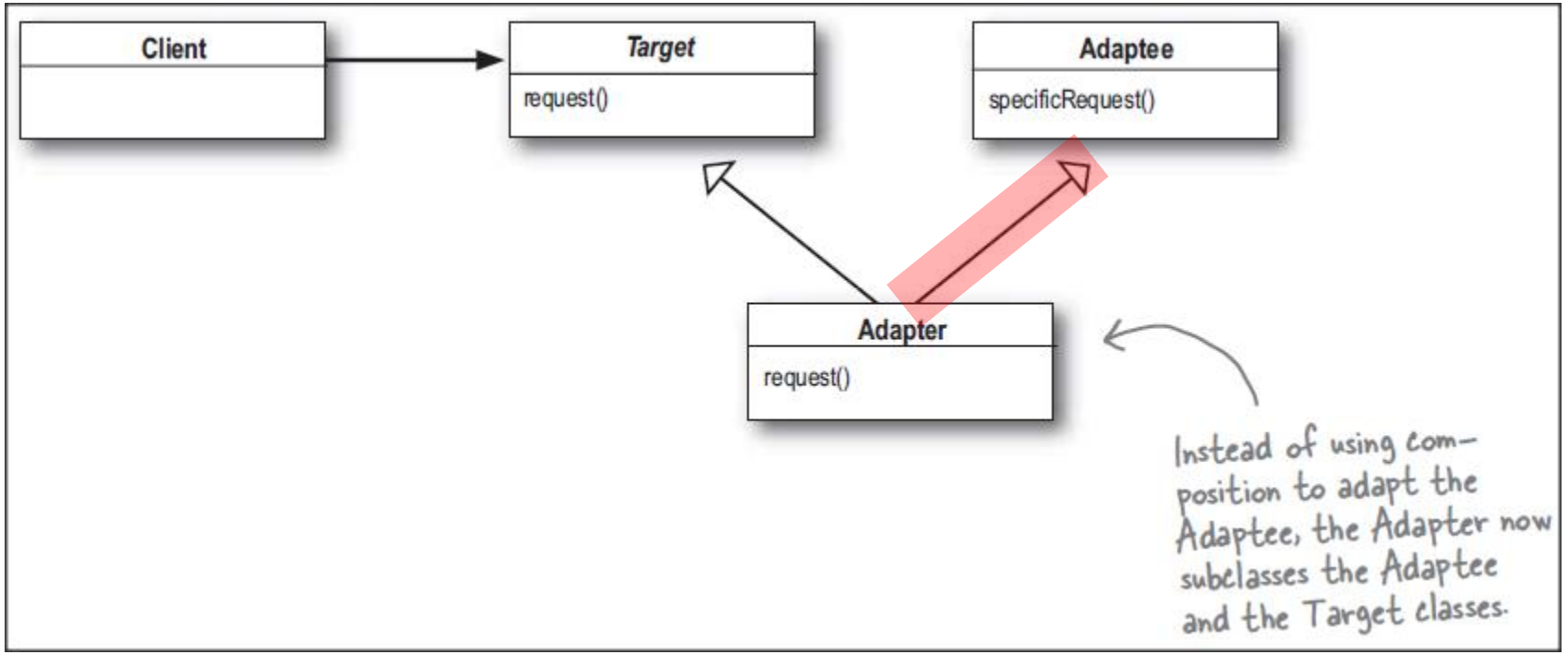
객체 어댑터 사용

- Adapter에서 Adaptee에 대하여 구성(composition)을 사용
  → 어댑티의 서브 클래스에 대해서도 어댑터 사용 가능!



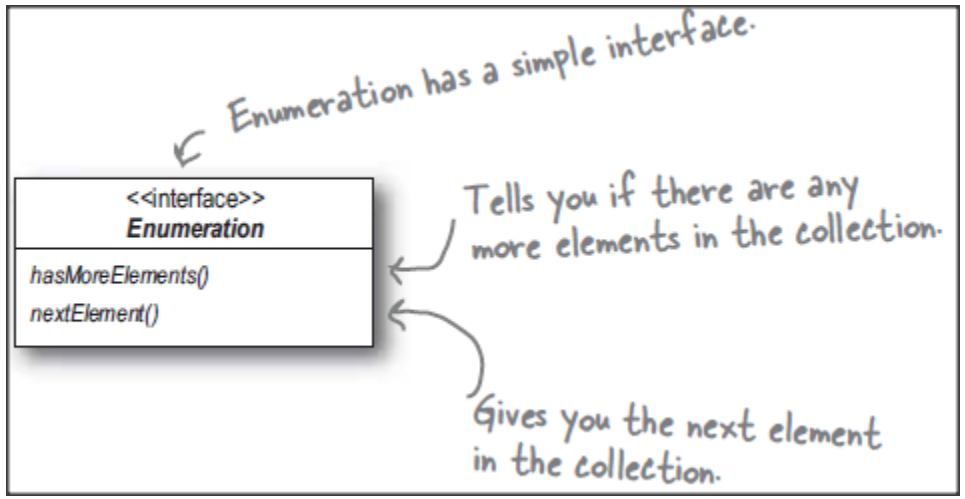
클래스 어댑터 사용

- Adapter에서 Adaptee에 대하여 상속을 사용
- 자바에서는 다중 상속이 지원되지 않으므로 사용 불가

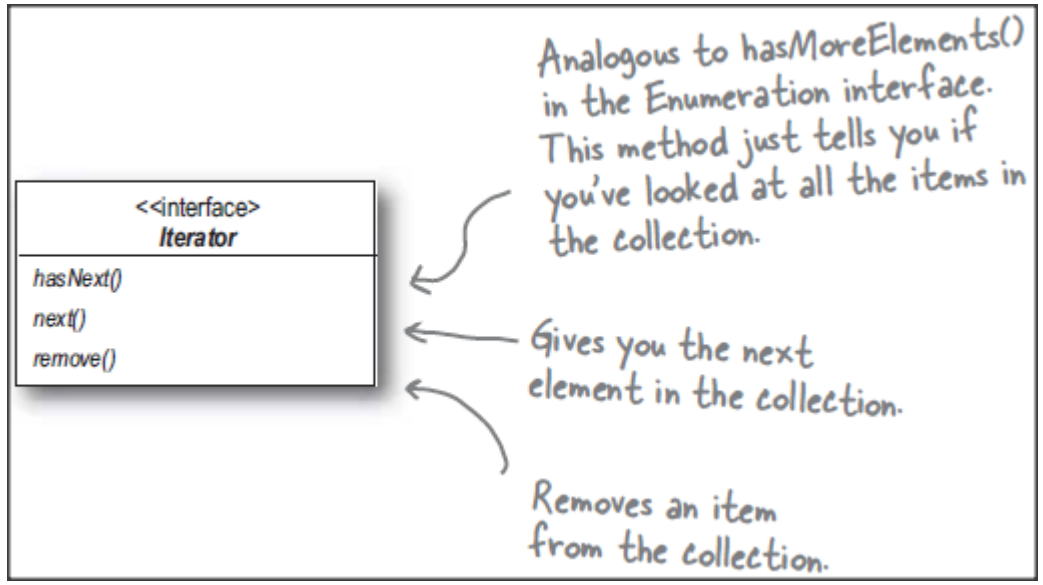


Enumeration vs. Iterator

- java.util.Enumeration: since 1.0



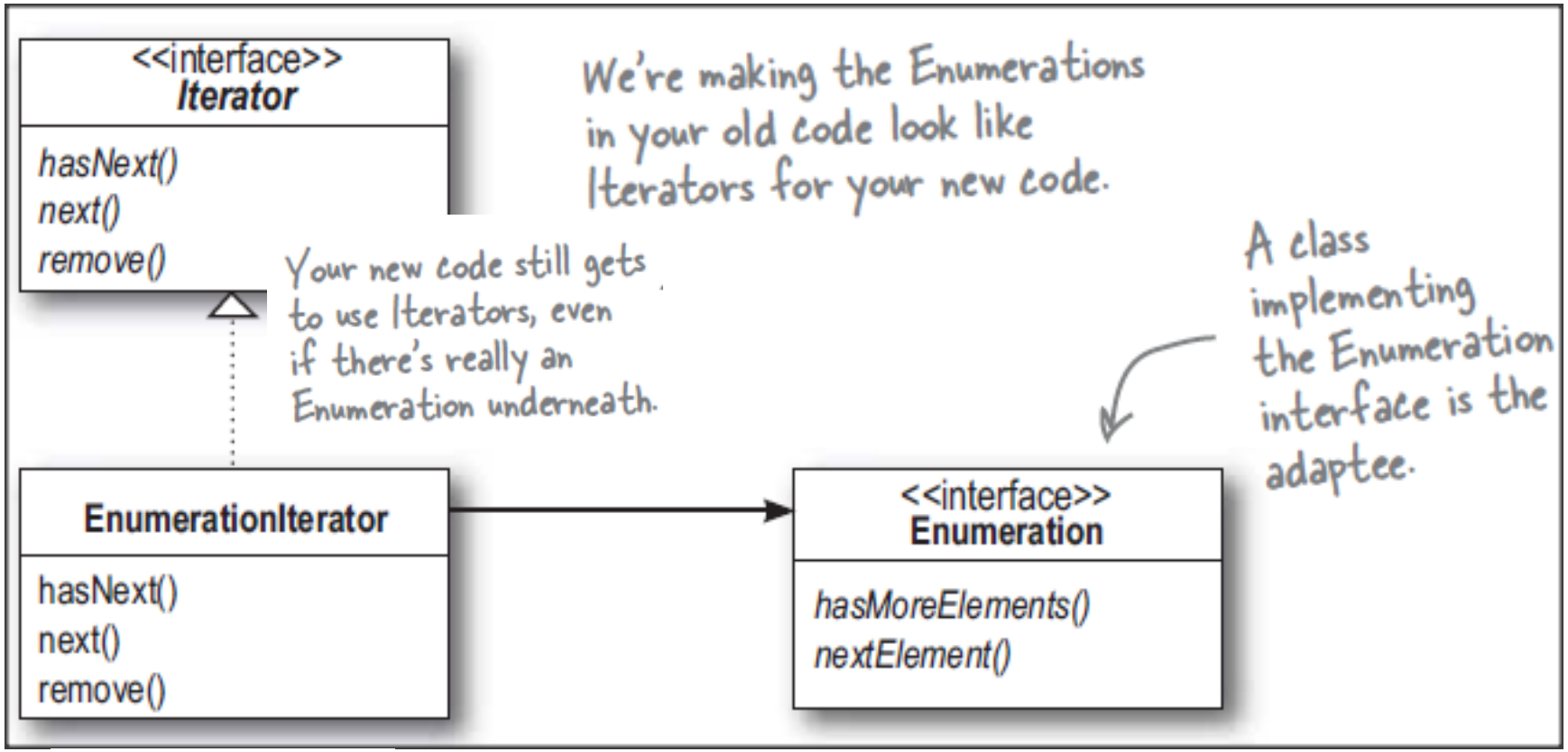
java.util.Iterator: since 1.2



Enumeration: The functionality of this interface is duplicated by the Iterator interface. In addition, Iterator adds an optional remove operation, and has shorter method names. **New implementations should consider using Iterator in preference to Enumeration**. It is possible to adapt an Enumeration to an Iterator by using the asIterator() method (since 9).

Enumeration 어댑터 사용

- 객체 어댑터 이용



EnumerationIterator.java

```
public class EnumerationIterator implements Iterator
{
    Enumeration enum;

    public EnumerationIterator(Enumeration enum) {
        this.enum = enum;
    }

    public boolean hasNext() {
        return enum.hasMoreElements();
    }

    public Object next() {
        return enum.nextElement();
    }

    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```

Since we're adapting Enumeration to Iterator, our Adapter implements the Iterator interface. it has to look like an Iterator.

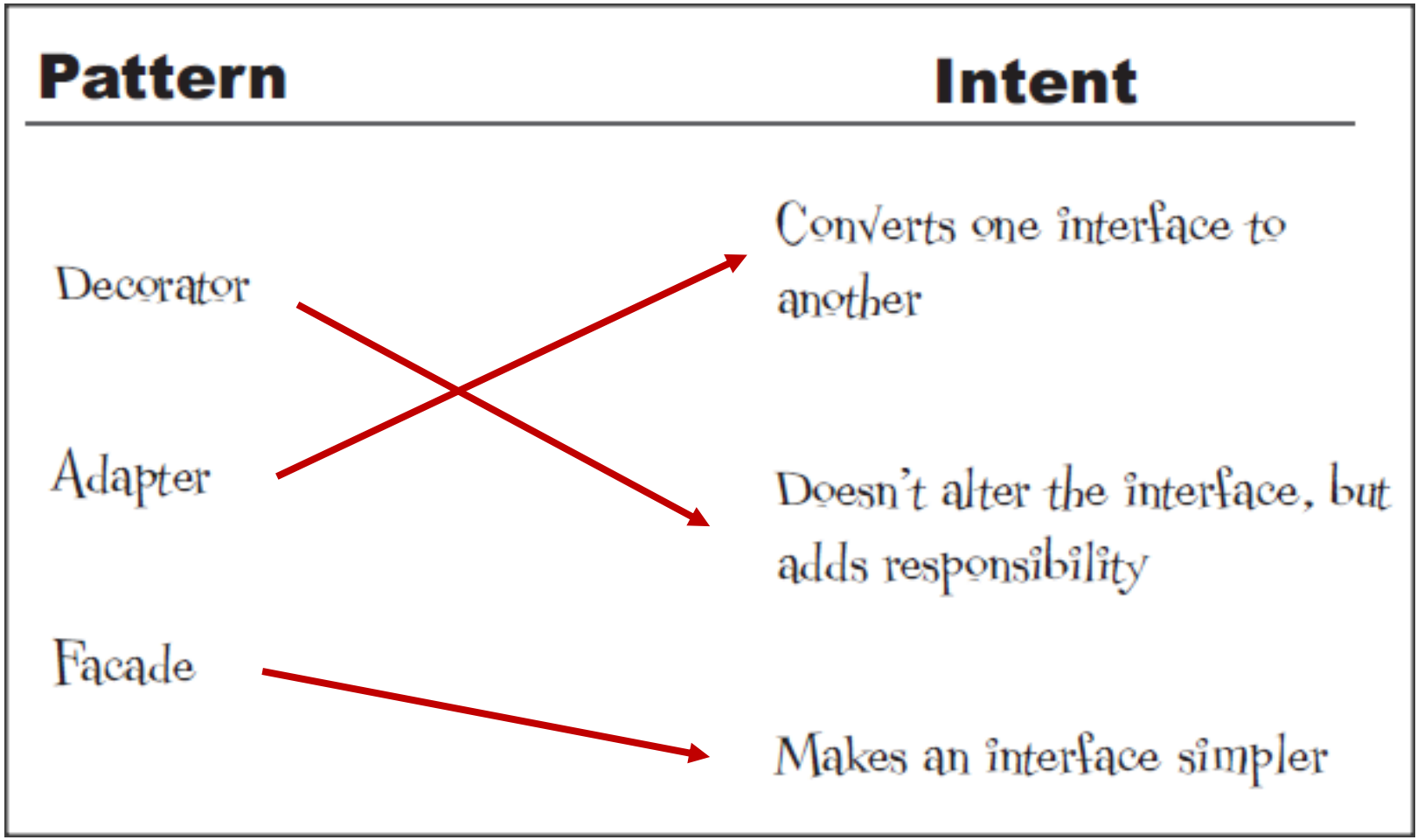
The Enumeration we're adapting. We're using composition so we stash it in an instance variable.

The Iterator's hasNext() method is delegated to the Enumeration's hasMoreElements() method...

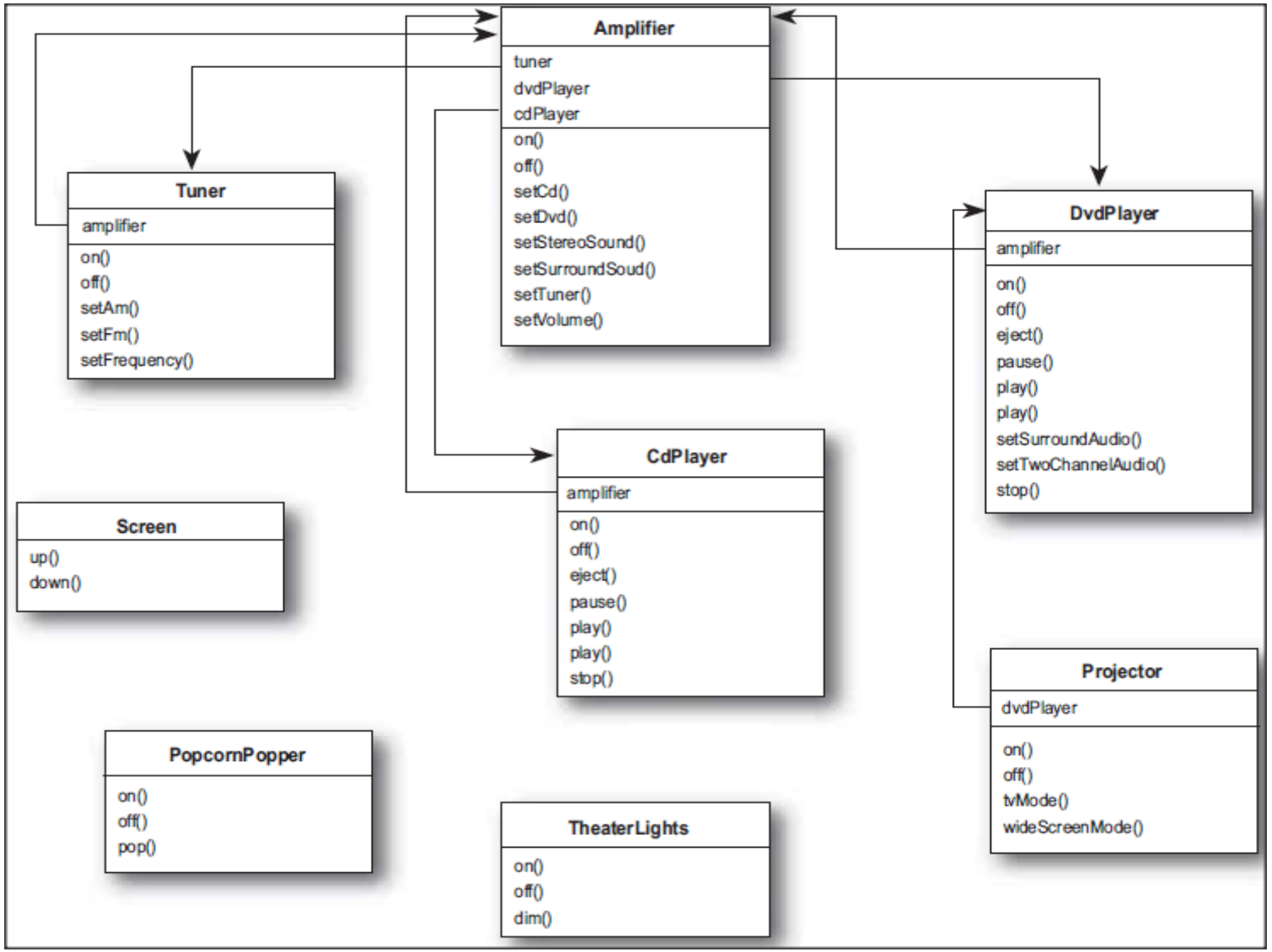
... and the Iterator's next() method is delegated to the Enumerations's nextElement() method.

Unfortunately, we can't support Iterator's remove() method, so we have to punt (in other words, we give up!). Here we just throw an exception.

패턴의 의미 찾기

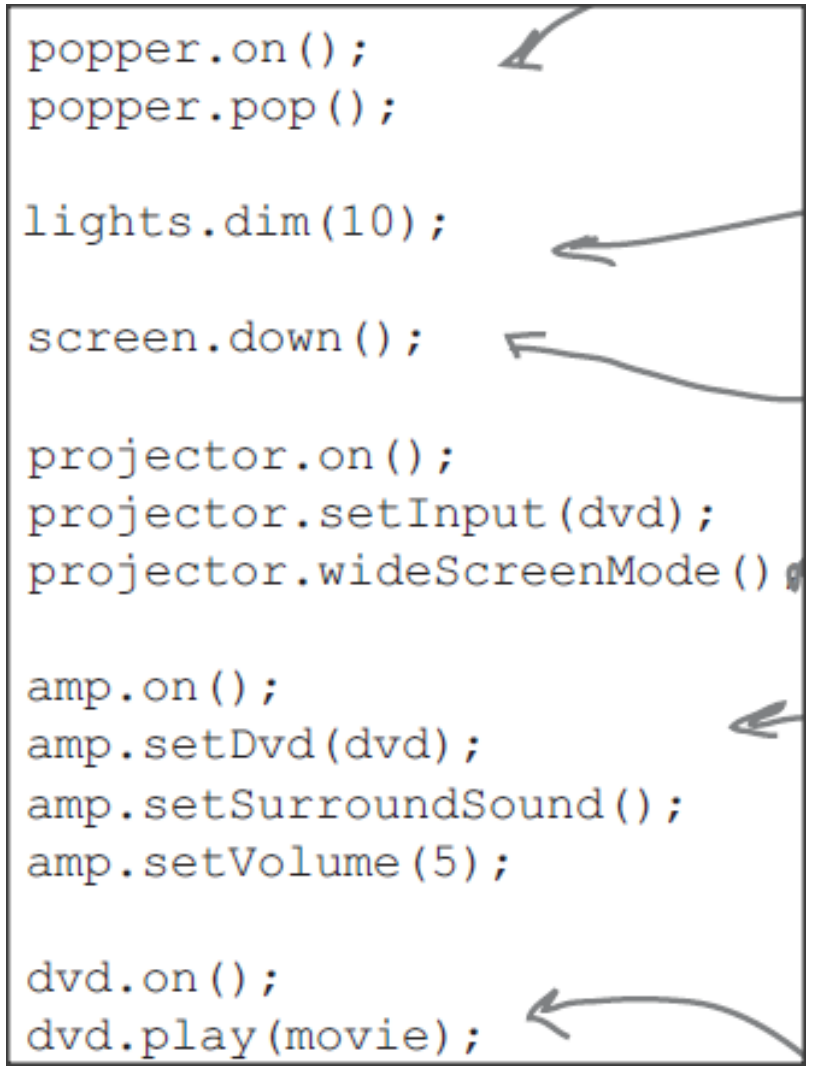

Pattern
Intent
Decorator
Adapter
Facade
Converts one interface to another
Doesn't alter the interface, but adds responsibility
Makes an interface simpler

Home Theater

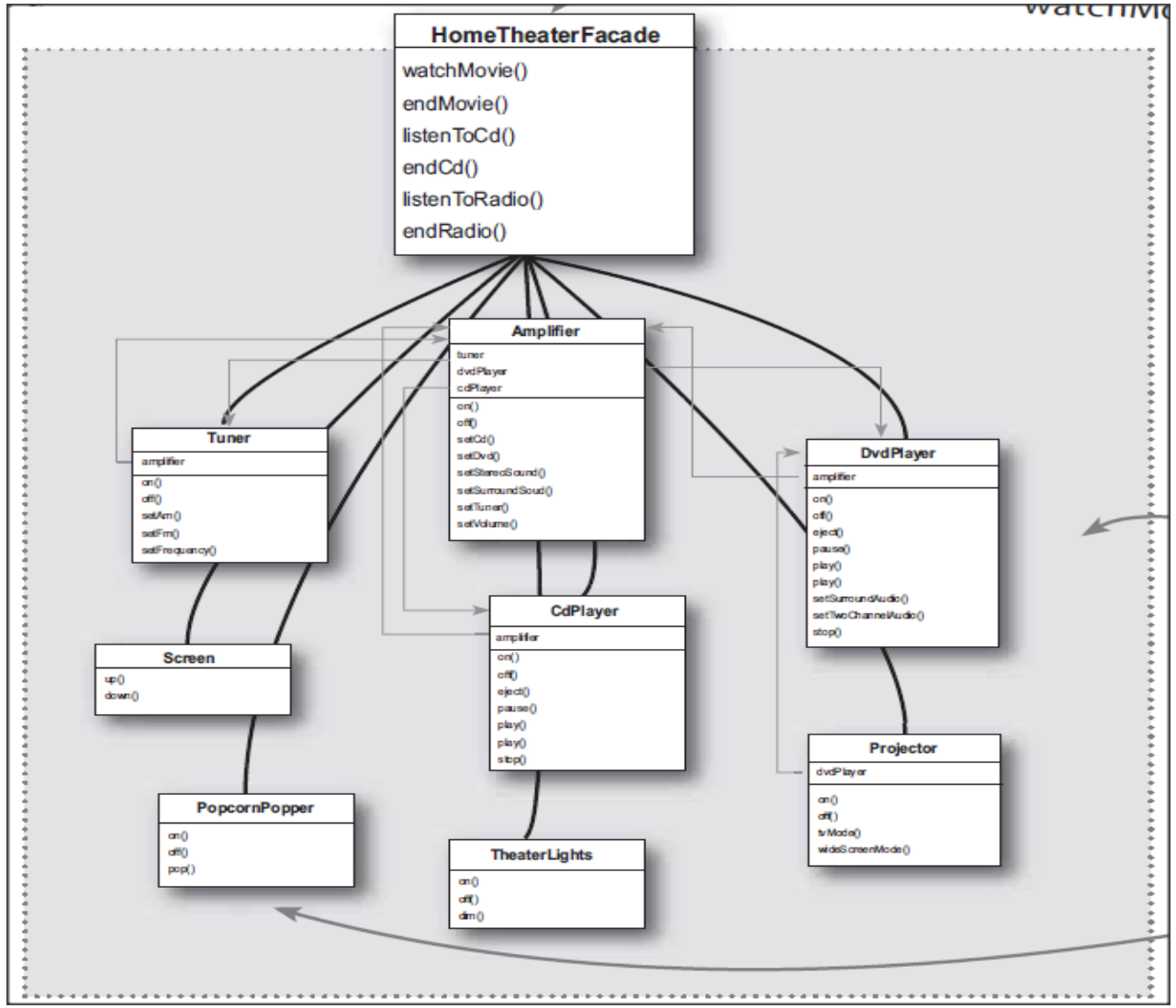


영화를 보려면...

1. Turn on the popcorn popper
2. Start the popper popping
3. Dim the lights
4. Put the screen down
5. Turn the projector on
6. Set the projector input to DVD
7. Put the projector on wide-screen mode
8. Turn the sound amplifier on
9. Set the amplifier to DVD input
10. Set the amplifier to surround sound
11. Set the amplifier volume to medium (5)
12. Turn the DVD Player on
13. Start the DVD Player playing



퍼사드 클래스 도입!


HomeTheaterFacade
watchMovie()
endMovie()
listenToCd()
endCd()
listenToRadio()
endRadio()
Amplifier
tuner
dvdPlayer
cdPlayer
on()
off()
setCd()
setDvd()
setStereoSound()
setSurroundSoud()
setTuner()
setVolume()
Tuner
amplifier
on()
off()
setAm()
setFm()
setFrequency()
DvdPlayer
amplifier
on()
off()
eject()
pause()
play()
play()
setSurroundAudio()
setTwoChannelAudio()
stop()
CdPlayer
amplifier
on()
off()
eject()
pause()
play()
play()
stop()
Screen
up()
down()
Projector
dvdPlayer
on()
off()
tvMode()
wideScreenMode()
PopcornPopper
on()
off()
pop()
TheaterLights
on()
off()
dim()

HomeTheaterFacade.java

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

    public HomeTheaterFacade(Amplifier amp,
                             Tuner tuner,
                             DvdPlayer dvd,
                             CdPlayer cd,
                             Projector projector,
                             Screen screen,
                             TheaterLights lights,
                             PopcornPopper popper) {

        this.amp = amp;
        this.tuner = tuner;
        this.dvd = dvd;
        this.cd = cd;
        this.projector = projector;
        this.screen = screen;
        this.lights = lights;
        this.popper = popper;
    }

        // other methods here
}
```

Here's the composition; these are all the components of the subsystem we are going to use.

The facade is passed a reference to each component of the subsystem in its constructor. The facade then assigns each to the corresponding instance variable.

We're just about to fill these in...

```
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.setSurroundSound();
    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();
}
```

watchMovie() follows the same sequence we had to do by hand before, but wraps it up in a handy method that does all the work. Notice that for each task we are delegating the responsibility to the corresponding component in the subsystem.

And endMovie() takes care of shutting everything down for us. Again, each task is delegated to the appropriate component in the subsystem.

HomeTheaterTestDrive.java

```
public class HomeTheaterTestDrive {
    public static void main(String[] args) {
        // instantiate components here

        HomeTheaterFacade homeTheater =
                new HomeTheaterFacade(amp, tuner, dvd, cd,
                        projector, screen, lights, popper);

        homeTheater.watchMovie("Raiders of the Lost Ark");
        homeTheater.endMovie();
    }
}
```

Here we're creating the components right in the test drive. Normally the client is given a facade, it doesn't have to construct one itself.

First you instantiate the Facade with all the components in the subsystem

Use the simplified interface to first start the movie up, and then shut it down.

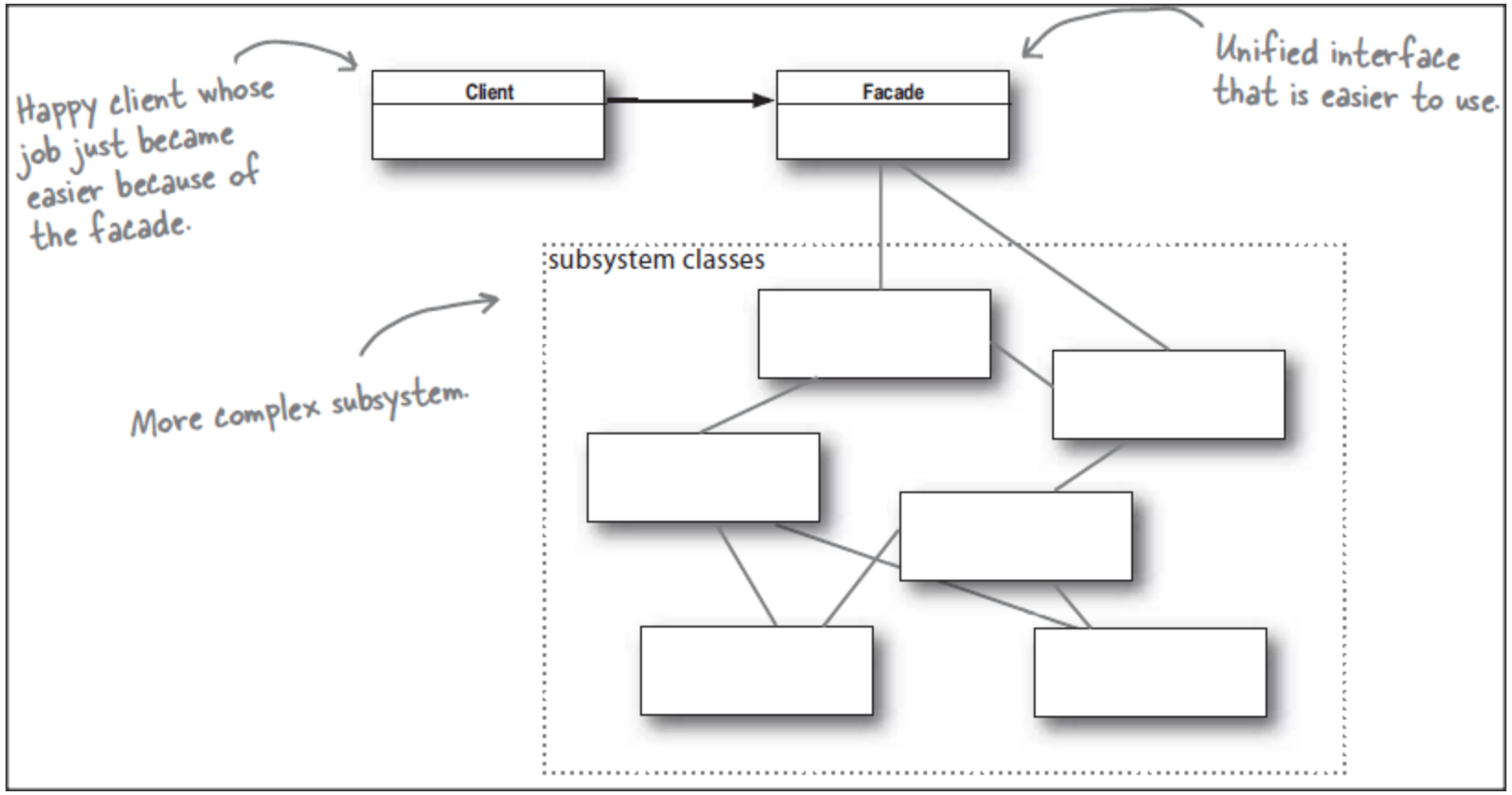
실행 결과

```
File Edit Window Help SnakesWhy'dItHaveToBeSnakes?
%java HomeTheaterTestDrive

Get ready to watch a movie...
Popcorn Popper on
Popcorn Popper popping popcorn!
Theater Ceiling Lights dimming to 10%
Theater Screen going down
Top-O-Line Projector on
Top-O-Line Projector in widescreen mode (16x9 aspect ratio)
Top-O-Line Amplifier on
Top-O-Line Amplifier setting DVD player to Top-O-Line DVD Player
Top-O-Line Amplifier surround sound on (5 speakers, 1 subwoofer)
Top-O-Line Amplifier setting volume to 5
Top-O-Line DVD Player on
Top-O-Line DVD Player playing "Raiders of the Lost Ark"
Shutting movie theater down...
Popcorn Popper off
Theater Ceiling Lights on
Theater Screen going up
Top-O-Line Projector off
Top-O-Line Amplifier off
Top-O-Line DVD Player stopped "Raiders of the Lost Ark"
Top-O-Line DVD Player eject
Top-O-Line DVD Player off
%
```

퍼사드 패턴 정의

- 어떤 서브 시스템의 여러 인터페이스에 대한 통합 인터페이스 제공
- 퍼사드: 서브 시스템을 보다 사용하기 쉽게 고수준 인터페이스 정의



최소 지식 원칙***

- The Principle of Least Knowledge – talk only to your immediate friends.
- 아주 가까운(또는 친밀한) 친구하고만 이야기하라.

- 친구를 만들지 않고 객체와 이야기하기 위한 **가이드 라인**:
  다음 객체에게만 메소드 호출을 한다.
  - **객체 자신**
  - 메소드에게 매개변수로 전달된 객체: **매개변수 의존성**
  - 메소드가 생성하거나 인스턴스를 만든 객체: **생성 의존성**
  - 그 객체에 속하는 구성요소(즉, 인스턴스 변수): **연관**

가이드 라인 준수 여부

```
public float getTemp() {
    Thermometer thermometer = station.getThermometer();
    return thermometer.getTemperature();
}
```

thermometer: 인스턴스 변수 X

this → station → thermometer

Here we get the thermometer object from the station and then call the getTemperature() method ourselves.

```
public float getTemp() {
    return station.getTemperature();
}
```

station: 인스턴스 변수 O

this → station

When we apply the principle, we add a method to the Station class that makes the request to the thermometer for us. This reduces the number of classes we're dependent on.

실습

Enumeration 사용 예

```
import java.util.List;     // ListEnumerationExample.java
import java.util.ArrayList;
import java.util.Collections;
import java.util.Enumeration;

public class ListEnumerationExample {
    public static void main(String[] args) {
        List<String> fruits = new ArrayList<>();
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Cherry");

        Enumeration<String> e = Collections.enumeration(fruits);

        while (e.hasMoreElements()) {
            String fruit = e.nextElement();
            System.out.println(fruit);
        }
    }
}
```

EnumerationIterator 사용 예

```
import java.util.*;  // AdapterPatternExample.java

public class AdapterPatternExample {
    public static void main(String[] args) {
        List<String> fruits = new ArrayList<>();
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Cherry");

        Enumeration<String> enumeration = Collections.enumeration(fruits);
        Iterator<String> iterator = new EnumerationIterator<>(enumeration);

        while (iterator.hasNext()) {
            String fruit = iterator.next();
            System.out.println(fruit);
        }
    }
}
```

어댑터 패턴 사용 예

```
class EnumerationIterator<E> implements Iterator<E> {
    private final Enumeration<E> enumeration;

    public EnumerationIterator(Enumeration<E> enumeration) {
        this.enumeration = enumeration;
    }

    @Override
    public boolean hasNext() {
        return enumeration.hasMoreElements();
    }

    @Override
    public E next() {
        return enumeration.nextElement();
    }

    @Override
    public void remove() {
        throw new UnsupportedOperationException("remove() not supported by Enumeration");
    }
}
```

Iterator 사용 예

```
import java.util.*;

public class IteratorExample {
    public static void main(String[] args) {
        List<String> fruits = new ArrayList<>();
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Cherry");

        Iterator<String> iterator = fruits.iterator();

        while (iterator.hasNext()) {
            String fruit = iterator.next();
            System.out.println(fruit);
        }
    }
}
```

ListIterator 사용 예

```
import java.util.*;

public class ListIteratorExample {
    public static void main(String[] args) {
        List<String> fruits = new ArrayList<>();
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Cherry");

        ListIterator<String> iterator = fruits.listIterator();

        while (iterator.hasNext()) {
            String fruit = iterator.next();
            System.out.println(fruit);
        }
    }
}
```