面向对象设计模式　适配器与外观

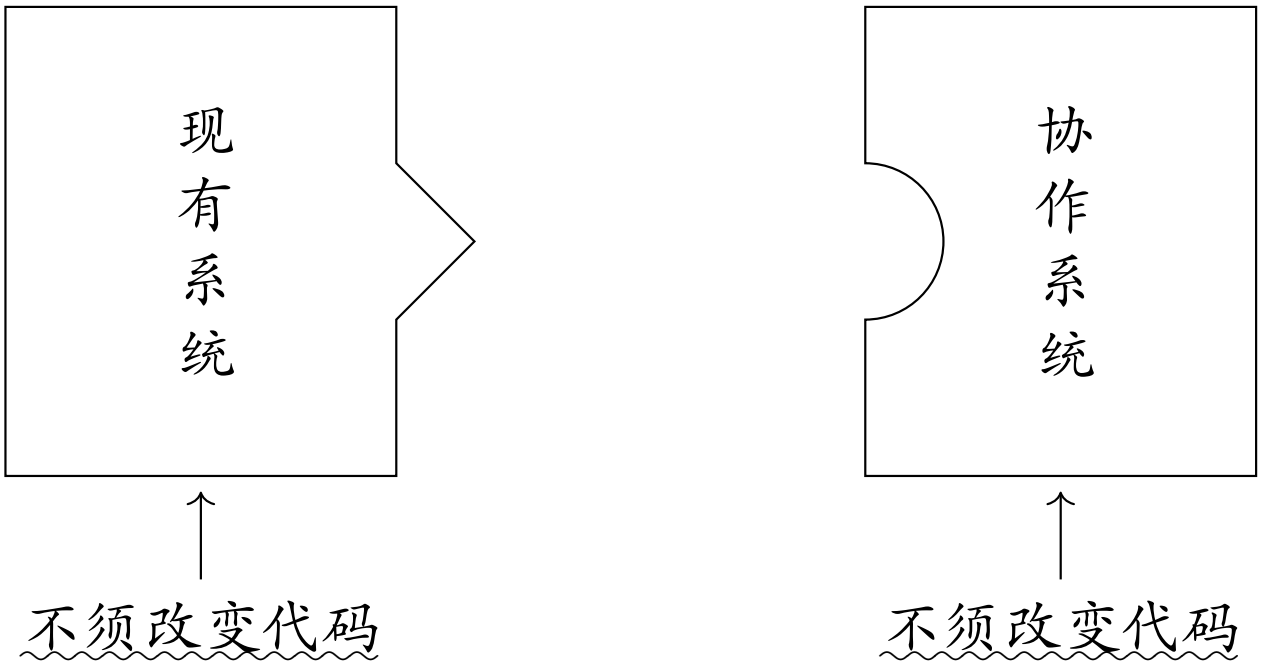
曹东刚
caodg@pku.edu.cn

北京大学信息学院研究生课程 - 面向对象的分析与设计
http://sei.pku.edu.cn/~caodg/course/oo

内容提要

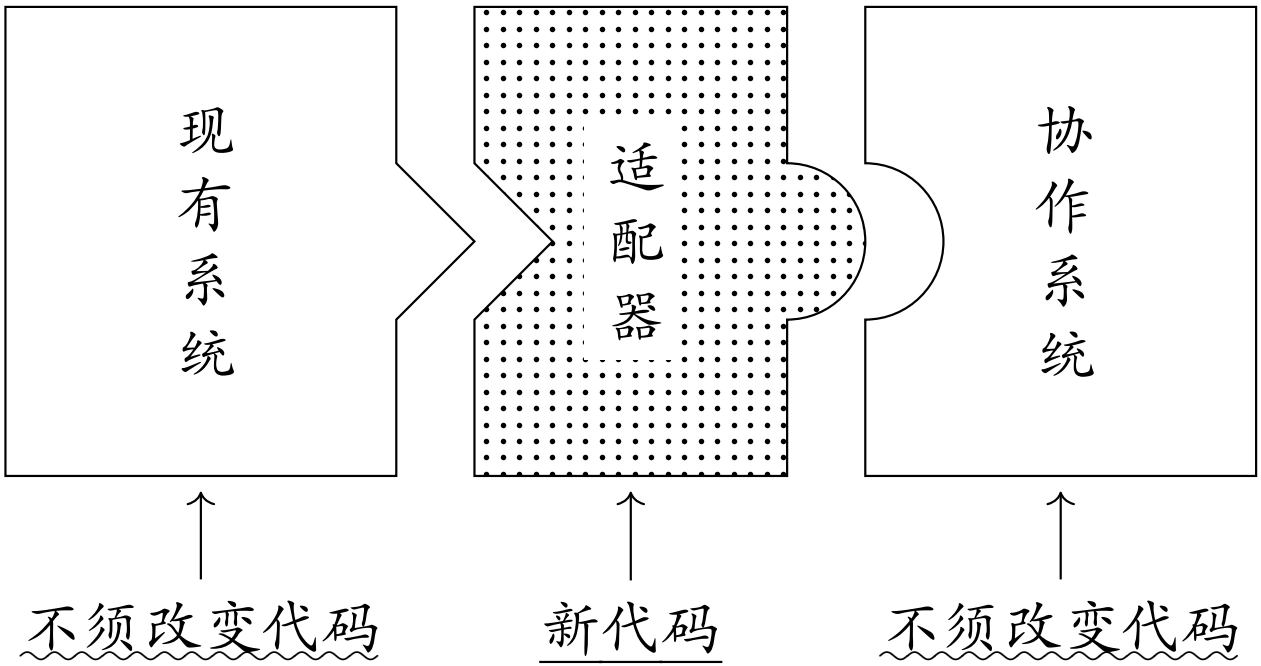
面向对象适配器

作用：将对象接口转换成另一种所期望的接口



面向对象适配器

作用：将对象接口转换成另一种所期望的接口



鸭子适配器

如果它走路像鸭子，叫起来像鸭子，那么它:

一定

是一只鸭子

鸭子适配器

如果它走路像鸭子，叫起来像鸭子，那么它:

~~一定~~可能

是一只~~鸭子~~包装了鸭子适配器的火鸡...

一个简易鸭子适配器的实现

定义鸭子接口并实现一只鸭子子类

```
1  public interface Duck {
2    public void quack();
3    public void fly();
4  }
5
6  public class MallardDuck implements Duck {
7    public void quack() {
8      System.out.println("Quack");
9    }
10   public void fly() {
11     System.out.println("I'm flying");
12   }
13 }
```

一个简易鸭子适配器的实现

定义火鸡接口并实现一只火鸡子类

```
1  public interface Turkey {
2    public void gobble();
3    public void fly();
4  }
5
6  public class WildTurkey implements Turkey {
7    public void gobble() {
8      System.out.println("Gobble gobble");
9    }
10   public void fly() {
11     System.out.println("I'm flying a short distance");
12   }
13 }
```

一个简易鸭子适配器的实现

鸭子适配器

```
1  public class TurkeyAdapter implements Duck {
2    Turkey turkey;
3    public TurkeyAdapter(Turkey turkey) {
4      this.turkey = turkey;
5    }
6    public void quack() {
7      turkey.gobble();
8    }
9    public void fly() {
10     for(int i=0; i < 5; i++)
11       turkey.fly();
12   }
13 }
```

一个简易鸭子适配器的实现

测试代码

```
1  public class DuckTestDrive {
2    public static void main(String[] args) {
3      WildTurkey turkey = new WildTurkey();
4      Duck turkeyAdapter = new TurkeyAdapter(turkey);
5      System.out.println("The TurkeyAdapter says...");
6      testDuck(turkeyAdapter);
7    }
8
9    static void testDuck(Duck duck) {
10     duck.quack();
11     duck.fly();
12   }
13 }
```

一个简易鸭子适配器的实现

测试结果

```
%java DuckTestDrive
The TurkeyAdapter says...
Gobble gobble
I'm flying a short distance
I'm flying a short distance
I'm flying a short distance
I'm flying a short distance
I'm flying a short distance
```

鸭子适配器解析

客户使用适配器的过程

1. 客户通过目标接口 (Duck) 调用适配器 (TurkeyAdapter) 的方法对适配器发出请求 (quack(), fly())
2. 适配器使用被适配者接口 (Turkey) 把请求转换成被适配者的一个或多个调用请求 (gobble(), fly())
3. 客户收到调用结果，但并未察觉这一切是适配器在起转换作用

思考

Q: 一个适配器需要做多少工作？和目标接口的大小有何关系？

思考

Q: 一个适配器需要做多少工作？和目标接口的大小有何关系？

Q: 一个适配器能封装几个类？

思考

Q: 一个适配器需要做多少工作？和目标接口的大小有何关系？

Q: 一个适配器能封装几个类？

Q: 如果系统中新旧代码并存，旧的部分使用旧的目标接口，新的部分使用新的目标接口，会怎样？

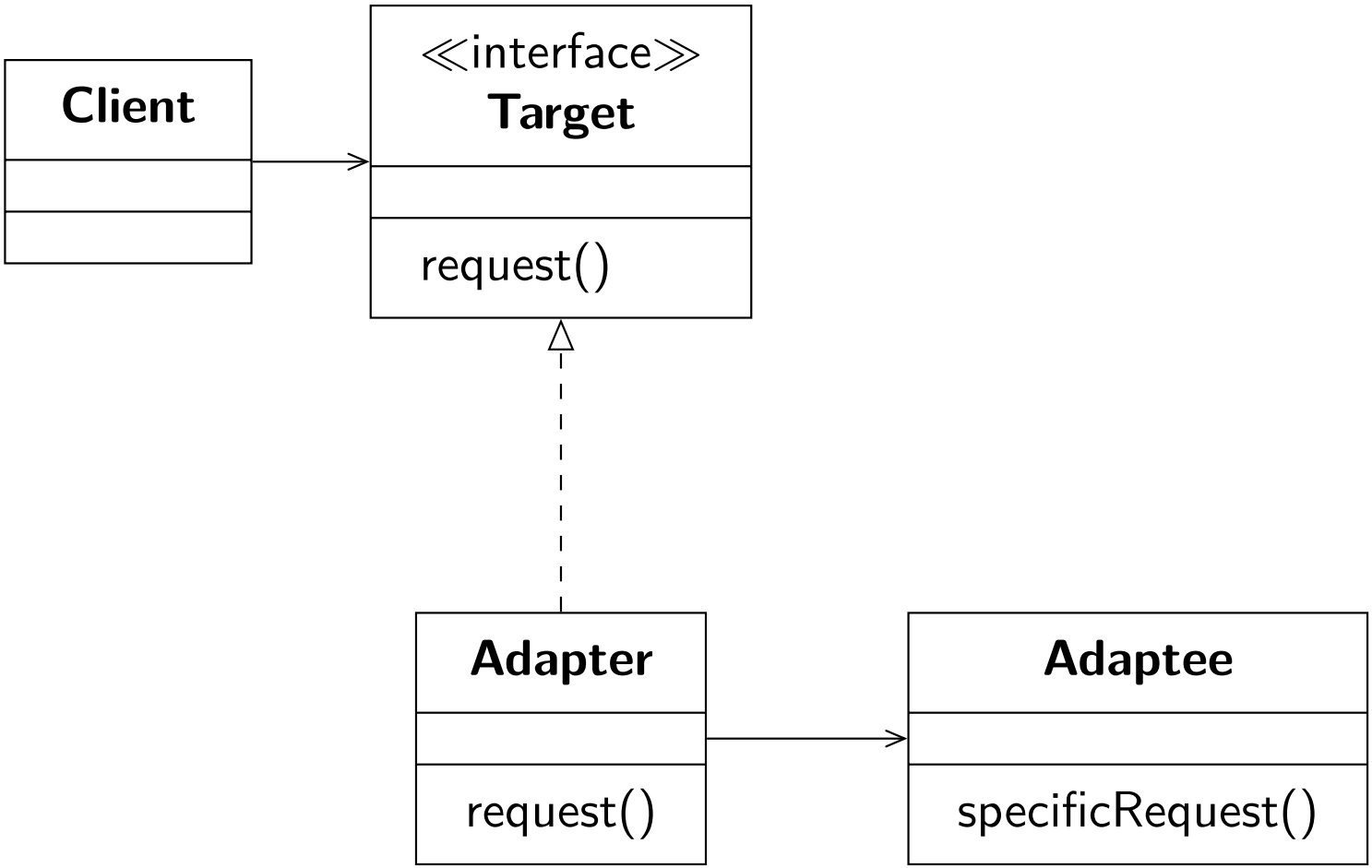
定义适配器模式

适配器模式 (Adapter Pattern)

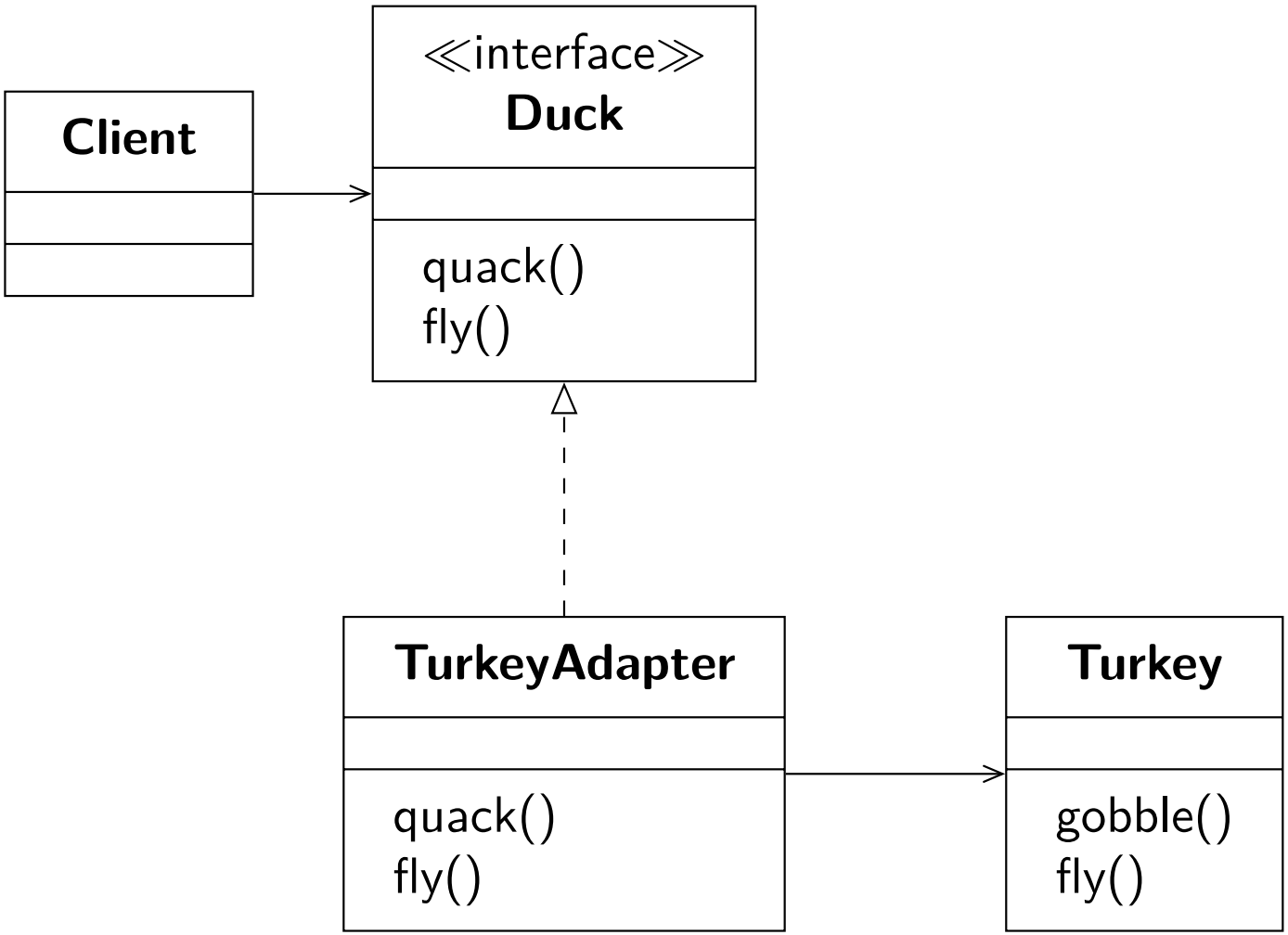
适配器模式将一个类的接口，转换成客户期望的另一个接口。适配器让原本接口不兼容的类可以合作无间

客户和目标接口解耦
适配器可以封装目标接口的改变

定义适配器模式

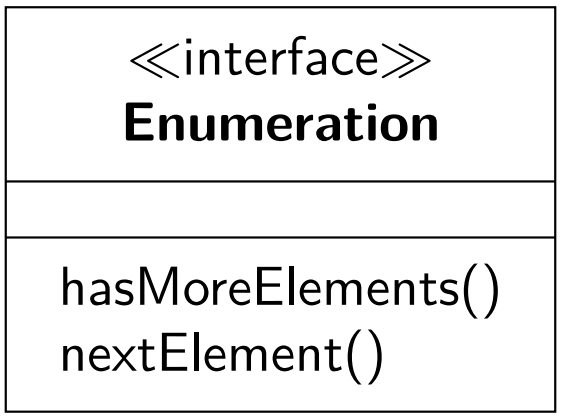

Client
≪interface≫
Target
request()
Adapter
request()
Adaptee
specificRequest()

应用适配器模式的鸭子应用



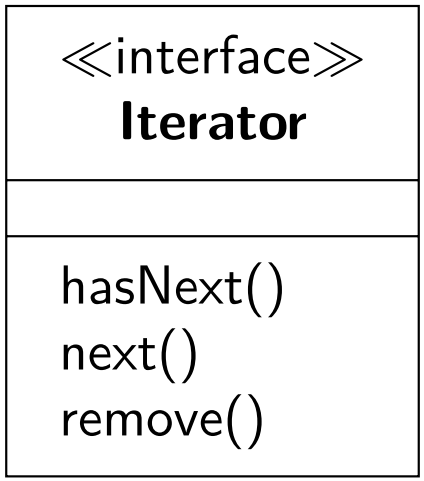
真实世界的适配器

早期 JDK 的 Collection 类型 (Vector、Stack、Hashtable 等) 都实现了一个 elements() 方法，该方法会返回一个 Enumeration 接口，通过该接口可以遍历集合内的所有元素



真实世界的适配器

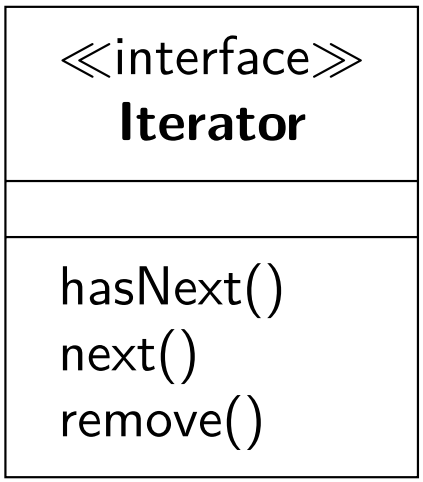
较新的 JDK 更新了 Collection 类型，开始使用 Iterator 接口，该接口和 Enumeration 接口相似，也可以遍历集合中的每个元素，但不同的是该迭代器还可以删除元素

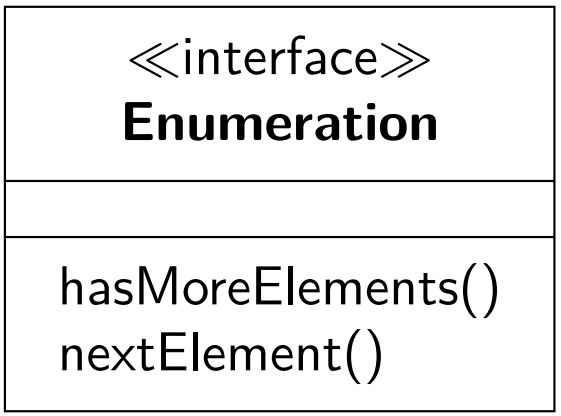


真实世界的适配器

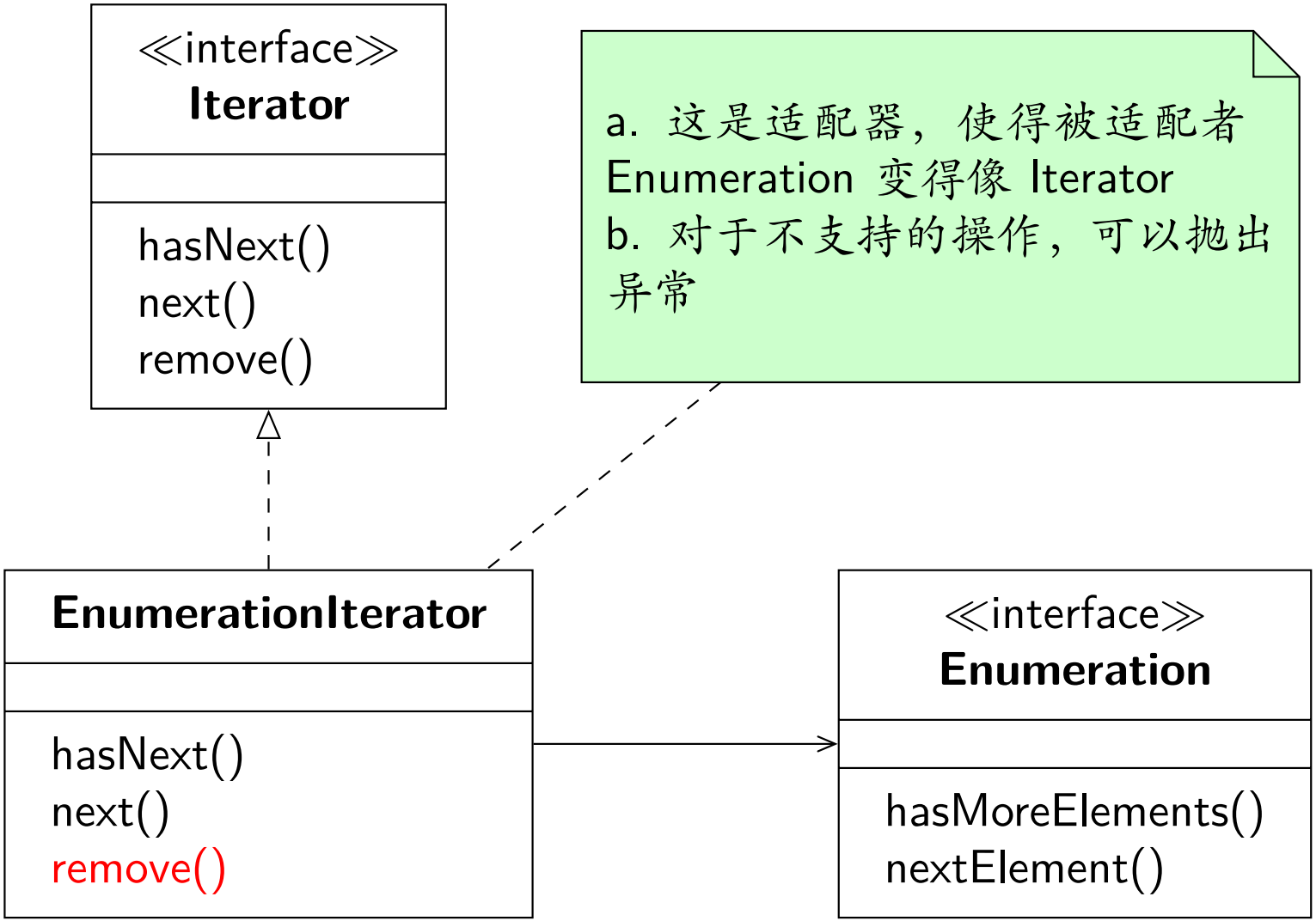
问题:

我们要调用遗留代码提供的服务，但遗留代码暴露的是老式的 Enumeration 接口，而我们希望在新的代码中只使用 Iterator，应该如何处理?





真实世界的适配器


≪interface≫
Iterator
hasNext()
next()
remove()
a. 这是适配器，使得被适配者 Enumeration 变得像 Iterator
b. 对于不支持的操作，可以抛出异常
EnumerationIterator
hasNext()
next()
remove()
≪interface≫
Enumeration
hasMoreElements()
nextElement()

真实世界的适配器

编写适配器代码

```
1  public class EnumerationIterator implements Iterator {
2    Enumeration enum;
3    public EnumerationIterator(Enumeration enum) {
4      this.enum = enum;
5    }
6    public boolean hasNext() {
7      return enum.hasMoreElements();
8    }
9    public Object next() {
10     return enum.nextElement();
11   }
12   public void remove() {
13     throw new UnsupportedOperationException();
14   }
15 }
```

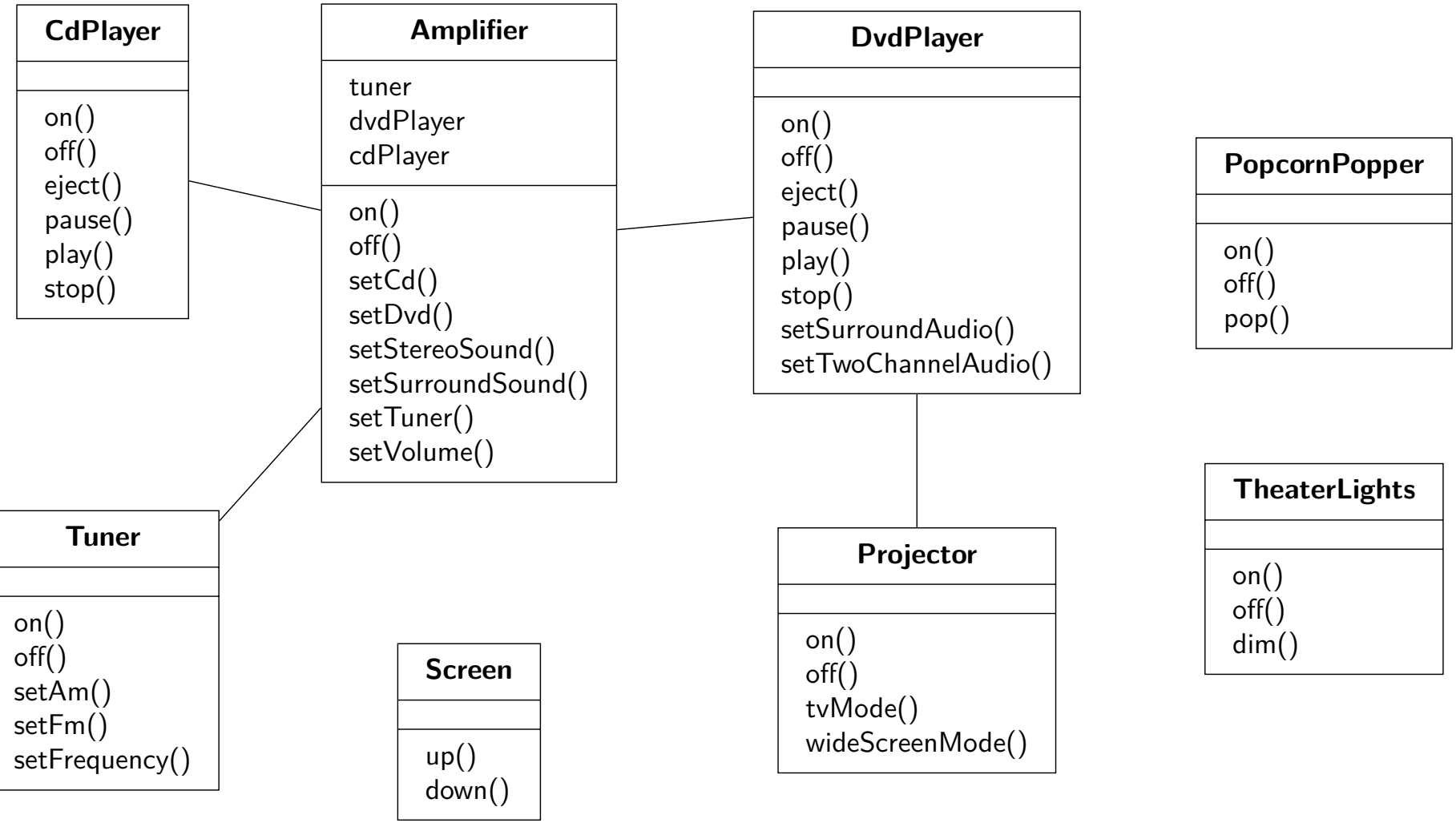
内容提要

简化复杂接口–外观模式 (Facade)

适配器 (Adapter) 模式将一个类的接口转换成另一个符合客户期望的接口。基本做法是将不兼容的对象包装起来，变成兼容对象

有时候某些类的接口非常复杂、庞大，为了简化对类的使用，需要改变接口使之简化，这就需要外观模式 (Facade)

家庭影院的例子


CdPlayer
on()
off()
eject()
pause()
play()
stop()
Amplifier
tuner
dvdPlayer
cdPlayer
on()
off()
setCd()
setDvd()
setStereoSound()
setSurroundSound()
setTuner()
setVolume()
DvdPlayer
on()
off()
eject()
pause()
play()
stop()
setSurroundAudio()
setTwoChannelAudio()
PopcornPopper
on()
off()
pop()
Tuner
on()
off()
setAm()
setFm()
setFrequency()
Screen
up()
down()
Projector
on()
off()
tvMode()
wideScreenMode()
TheaterLights
on()
off()
dim()

家庭影院的例子

欣赏 DVD 影片之前必须执行的任务:

1. 打开爆米花机
2. 开始爆米花
3. 将灯光调暗
4. 放下屏幕
5. 打开投影机
6. 将投影机的输入切换到 DVD
7. 将投影机设置在宽屏模式
8. 打开功放
9. 将功放的输入设置为 DVD
10. 将功放设置为环绕立体声
11. 将功放音量调到中 (5)
12. 打开 DVD 播放器
13. 开始播放 DVD

家庭影院的例子

观赏 DVD 需要完成任务的伪代码

```
1  popper.on();
2  popper.pop();
3  lights.dim(10);
4  screen.down();
5  projector.on();
6  projector.setInput(dvd);
7  projector.wideScreenMode();
8  amp.on();
9  amp.setDvd(dvd);
10 amp.setSurroundSound();
11 amp.setVolume(5);
12 dvd.on();
13 dvd.play(movie);
```

家庭影院的例子

还有:

- 看完电影后，需要把一切都关掉，怎么办?
- 如果要听 CD 或广播，要怎么做?
- 如果要升级影院系统，要重新学习一套全新的操作流程?

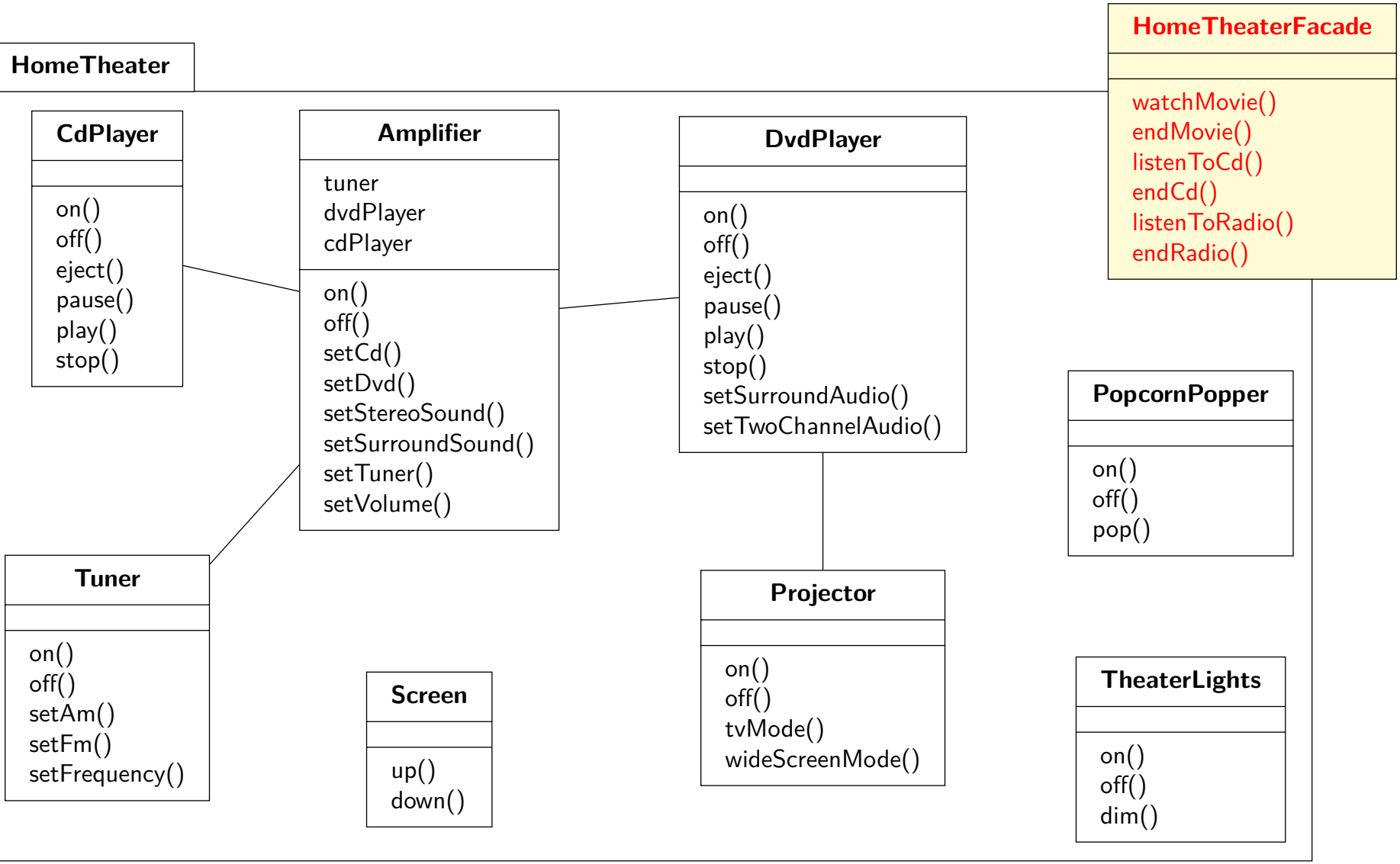
家庭影院的例子

还有:

- 看完电影后，需要把一切都关掉，怎么办?
- 如果要听 CD 或广播，要怎么做?
- 如果要升级影院系统，要重新学习一套全新的操作流程?

让外观模式来解决该问题

为家庭影院设计外观


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

为家庭影院设计外观

家庭影院外观类设计

```
1  public class HomeTheaterFacade {
2    Amplifier amp;
3    Tuner tuner;
4    DvdPlayer dvd;
5    CdPlayer cd;
6    Projector projector;
7    TheaterLights lights;
8    Screen screen;
9    PopcornPopper popper;
10
11   public HomeTheaterFacade(Amplifier amp, Tuner tuner,
12     DvdPlayer dvd, CdPlayer cd, Projector projector,
13     Screen screen, TheaterLights lights,
14     PopcornPopper popper) { ... }
```

为家庭影院设计外观

家庭影院外观类设计

```
15    public void watchMovie(String movie) {
16      System.out.println("Get ready to watch a movie...");
17      popper.on();
18      popper.pop();
19      lights.dim(10);
20      screen.down();
21      projector.on();
22      projector.wideScreenMode();
23      amp.on();
24      amp.setDvd(dvd);
25      amp.setSurroundSound();
26      amp.setVolume(5);
27      dvd.on();
28      dvd.play(movie);
29    }
```

为家庭影院设计外观

家庭影院外观类设计

```
30    public void endMovie() {
31      System.out.println("Shutting movie theater down...");
32      popper.off();
33      lights.on();
34      screen.up();
35      projector.off();
36      amp.off();
37      dvd.stop();
38      dvd.eject();
39      dvd.off();
40    }
```

为家庭影院设计外观

开始测试家庭影院外观

```
1  public class HomeTheaterTestDrive {
2    public static void main(String[] args) {
3      // instantiate components here
4
5      HomeTheaterFacade homeTheater =
6        new HomeTheaterFacade(amp, tuner, dvd, cd,
7          projector, screen, lights, popper);
8
9      homeTheater.watchMovie("Raiders of the Lost Ark");
10     homeTheater.endMovie();
11   }
12 }
```

为家庭影院设计外观

运行 java HomeTheaterTestDrive 测试结果

```
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
```

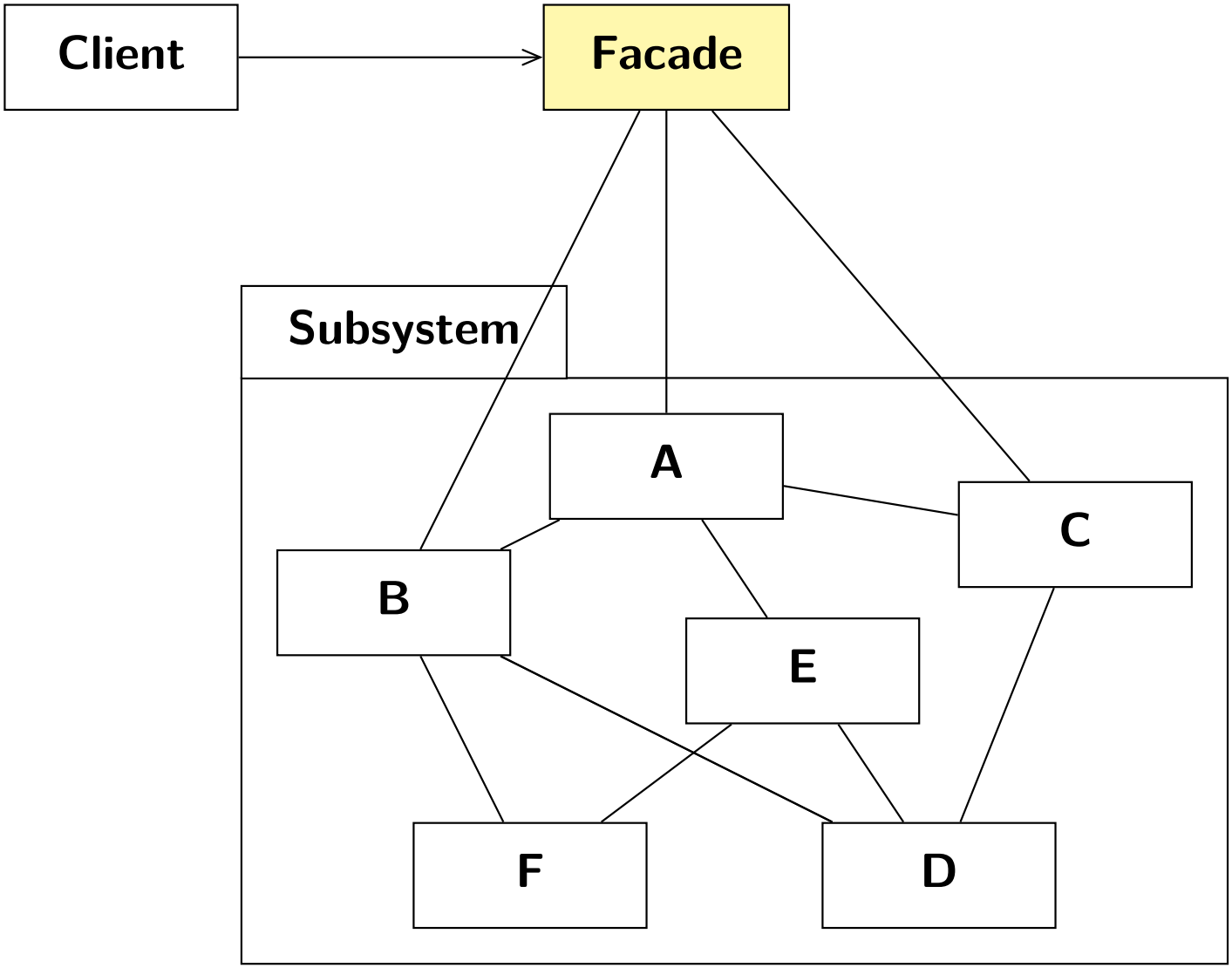
定义外观模式

外观模式 (Facade Pattern)

外观模式提供了一个统一的接口，用来访问子系统中的一群接口。外观定义了一个高层接口，让子系统更容易使用

外观模式可使客户和子系统之间松耦合
外观模式可帮助遵守“最少知识原则”

定义外观模式



最少知识原则

最少知识 (Least Knowledge) 原则告诉我们要减少对象之间的交互，只留下几个“密友”

最少知识设计原则

只和密友交谈

该原则希望在设计系统时，不要让太多的类耦合到一起，免得修改系统中的一部分，会影响到其他部分
如果许多类彼此依赖，则维护成本就会升高

最少知识原则

例:

```
1  public float getTemp() {
2    return station.getThermometer().getTemperature();
3  }
```

Q: 该段代码耦合了多少类?

Q: 哪些类是该代码所属类的朋友?

如何做到“只和密友”交谈?

原则上，一个对象应该只和如下对象交互

- 该对象本身
- 做为方法参数传递进来的对象
- 该对象创建或实例化的对象
- 该对象实例变量所引用的对象 (关联对象，聚合对象)

如何做到“只和密友”交谈?

不采用该原则的代码

```
1  public float getTemp() {
2    Thermometer thermometer = station.getThermometer();
3    return thermometer.getTemperature();
4  }
```

如何做到“只和密友”交谈?

不采用该原则的代码

```
1  public float getTemp() {
2    Thermometer thermometer = station.getThermometer();
3    return thermometer.getTemperature();
4  }
```

采用该原则的代码

```
1  public float getTemp() {
2    return station.getTemperature();
3  }
```

如何做到“只和密友”交谈?

另一个遵守最少知识原则的代码

```
1  public class Car {
2    Engine engine;
3    public Car() { // initialize engine, etc. }
4
5    void start(Key key) {
6      Doors doors = new Doors();
7      boolean authorized = key.turns();
8      if (authorized) {
9        engine.start();
10       updateDashboardDisplay();
11       doors.lock();
12     }
13   }
```

内容提要

关于适配器模式和外观模式的小结

- 当要使用一个类而该类接口不合期望时，使用适配器模式
- 当要简化一个很大的接口或一群复杂接口时，使用外观模式
- 适配器改变接口
- 外观封装子系统，将客户从子系统解耦
- 可以为子系统实现多个外观
- 适配器将对象包装起来改变其接口；装饰者将对象包装起来增加新行为；外观对象将一群对象包装起来以简化接口

关于设计原则的小结

1. 封装变化
2. 多用聚合、少用继承
3. 针对接口编程，不针对实现编程
4. 尽最大可能将要交互的对象设计为松耦合的
5. 对扩展开放，对修改封闭
6. 依赖抽象，不要依赖具体类
7. 只和朋友交谈