Lecture 3
The Adapter Pattern (适配器模式)
(Structural)

- 结构型设计模式的主要目的是将不同的类和对象组合在一起，形成更大或者更复杂的结构体，例如，形成复杂的用户接口或者复杂的账户数据接口。
- 值得注意的是，该模式不是简单地将这些类摆在一起，而是要提供这些类之间的关联方式。

Professor:

Yushan (Michael) Sun

Fall 2020

Contents of this lecture

1. Interface of a class
2. Introduction of the Adapter pattern
3. Class Adapter Pattern (类适配器模式)
4. Object Adapter Pattern(对象适配器模式)
5. Design Examples Using Adapter Pattern
6. Further discussions

java.lang.Math类的接口声明.

类的接口

Field Summary

static double	E The double value that is closer than any other to *e*, the base of the natural logarithms.
static double	PI The double value that is closer than any other to *pi*, the ratio of the circumference of a circle to its diameter.

类的接口

Method Summary

static double	**cos**(double a) Returns the trigonometric cosine of an angle.
static double	**exp**(double a) Returns Euler's number *e* raised to the power of a double value.
static double	**log**(double a) Returns the natural logarithm (base *e*) of a double value.
static double	**sin**(double a) Returns the trigonometric sine of an angle.

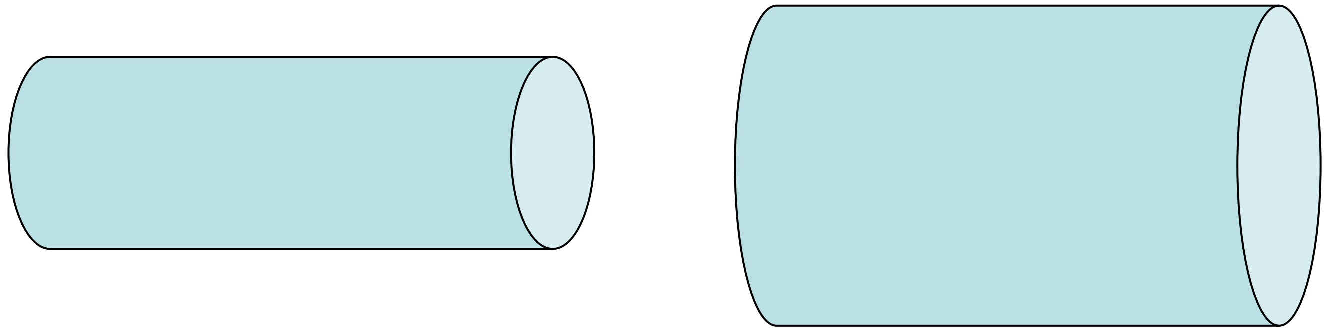
- **类的所有暴露给外界的方法的全体叫做类的接口**
- **类的接口提供外部视图**

Back

Introduction to the Adapter Pattern

Introduction to Adapter Pattern

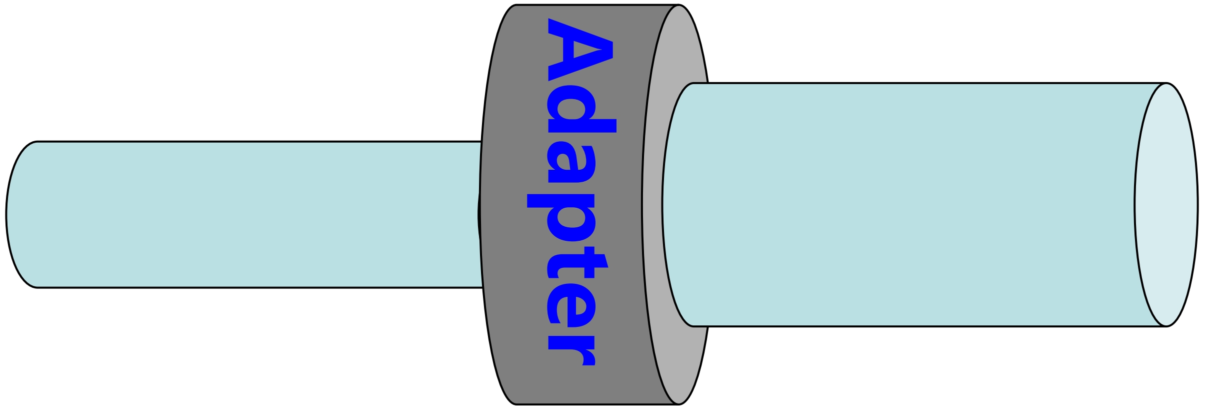
Question: there are 2 water pipes, one is thicker than the other



Question: How to connect these two pipes?
Problem: Incompatible interfaces

Introduction to Adapter Pattern

Solution: Use an adapter to adapt one interface to the other



转换了接口，从而可以将粗细不同的两个水管连接起来。

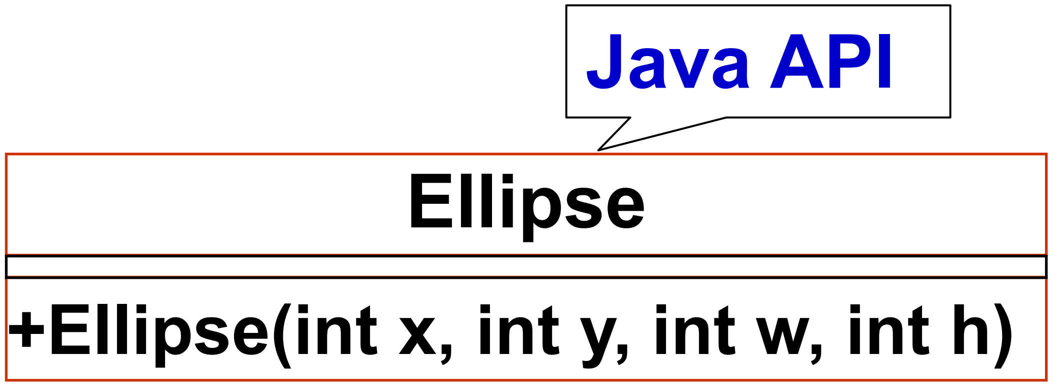
Introduction to Adapter Pattern

- For software design, we often encounter the similar problem: incompatible interface problem

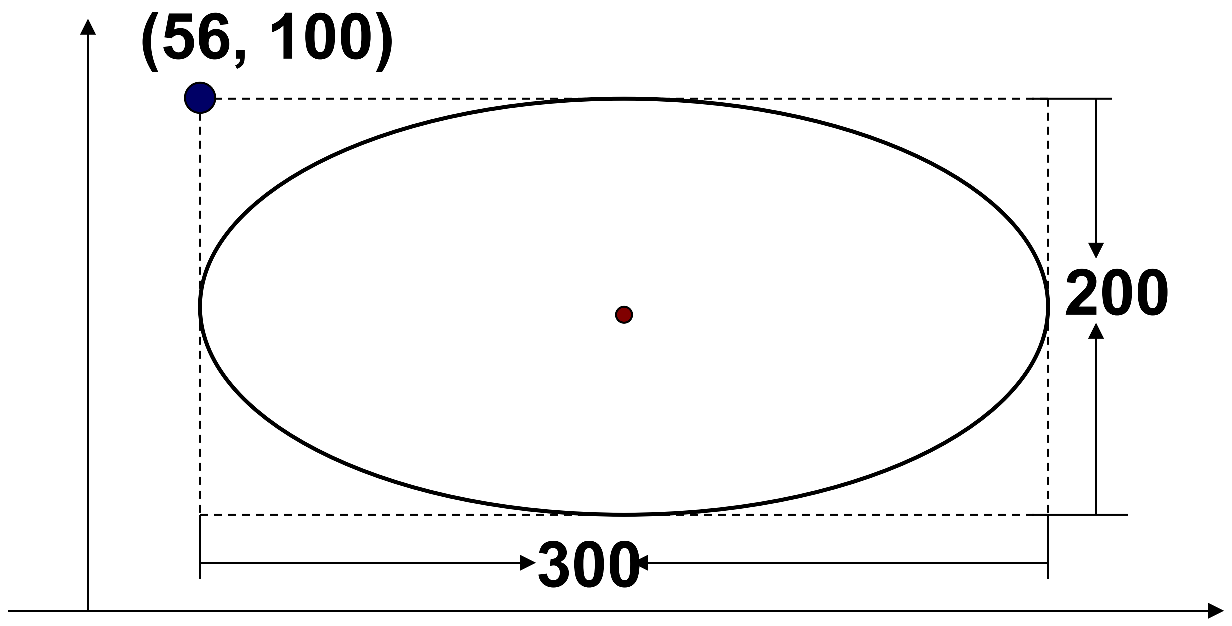
- 在软件设计中，我们也经常会遇到类似的接口不一致的问题

Introduction to Adapter Pattern

例1. 接口转换问题
Interface Conversion for an Ellipse

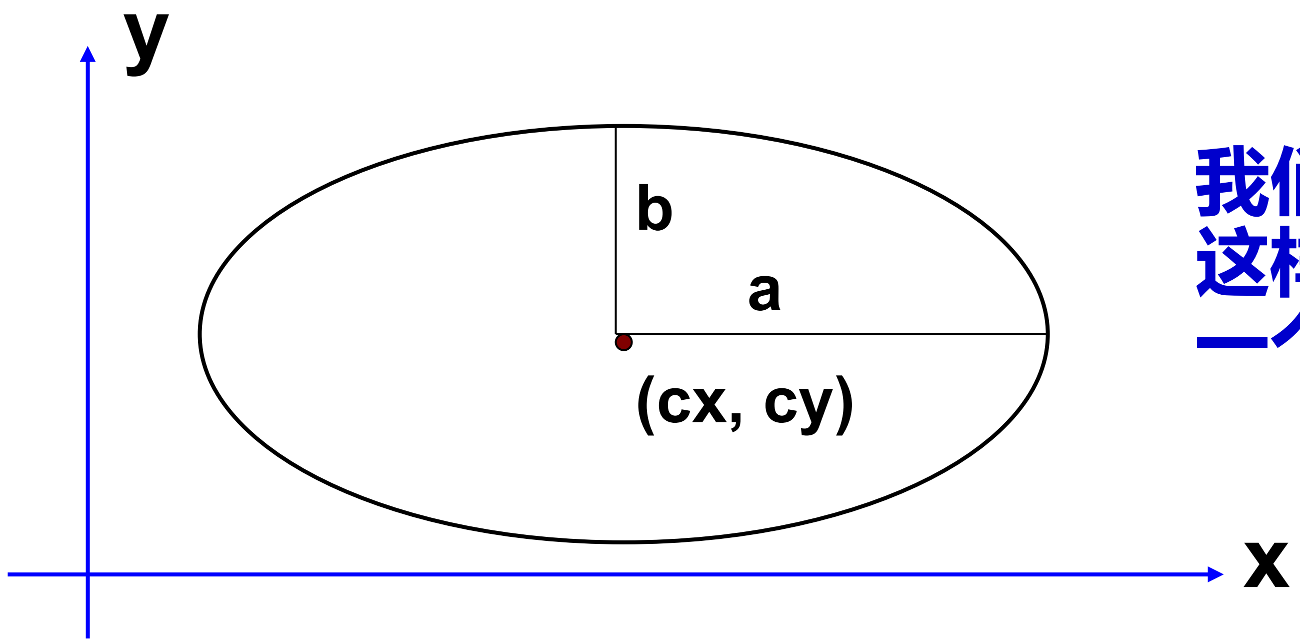


Ellipse e = new Ellipse (56, 100, 300, 200);



Introduction to Adapter Pattern

我们希望以如下的方式声明椭圆

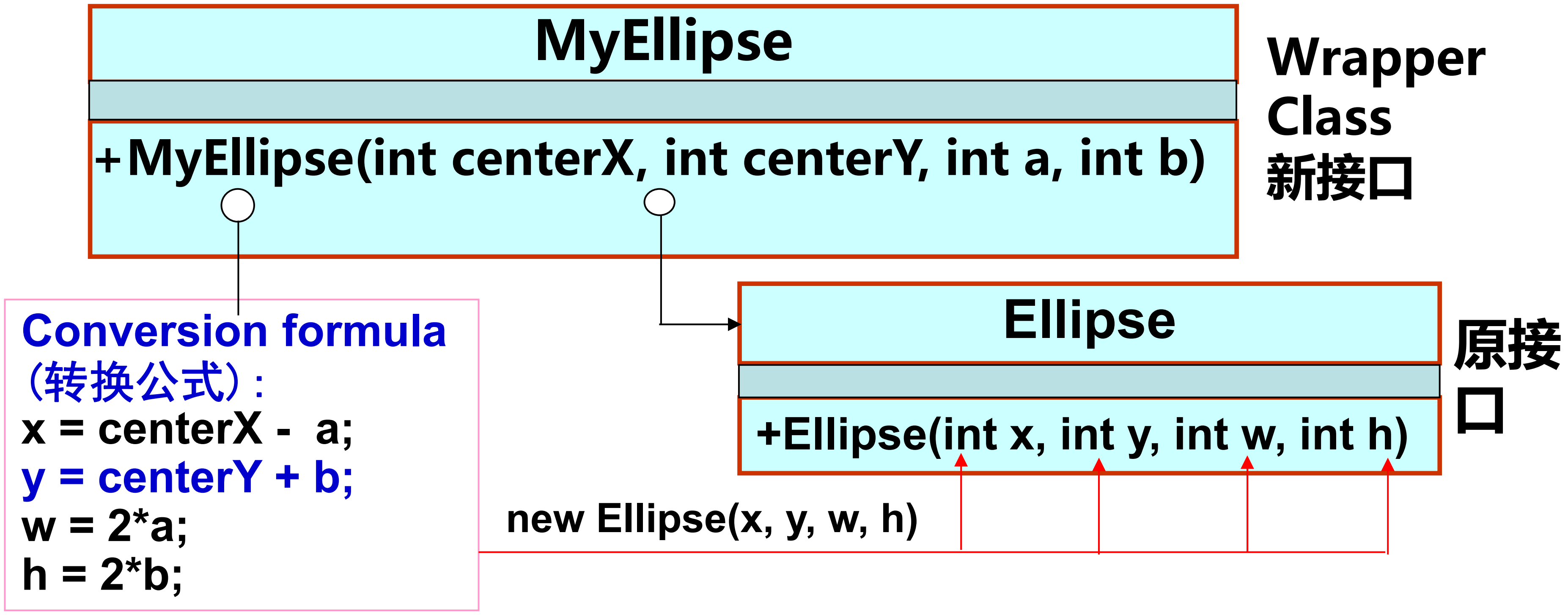


我们需要这样声明一个椭圆

Ellipse e = new Ellipse (cx, cy, a, b);

Question: in Java API, the existing constructor is not compatible with what we want.

Introduction to Adapter Pattern


MyEllipse
+MyEllipse(int centerX, int centerY, int a, int b)
Wrapper Class
新接口
Ellipse
+Ellipse(int x, int y, int w, int h)
原接口
new Ellipse(x, y, w, h)

Conversion formula (转换公式)：
x = centerX - a;
y = centerY + b;
w = 2*a;
h = 2*b;

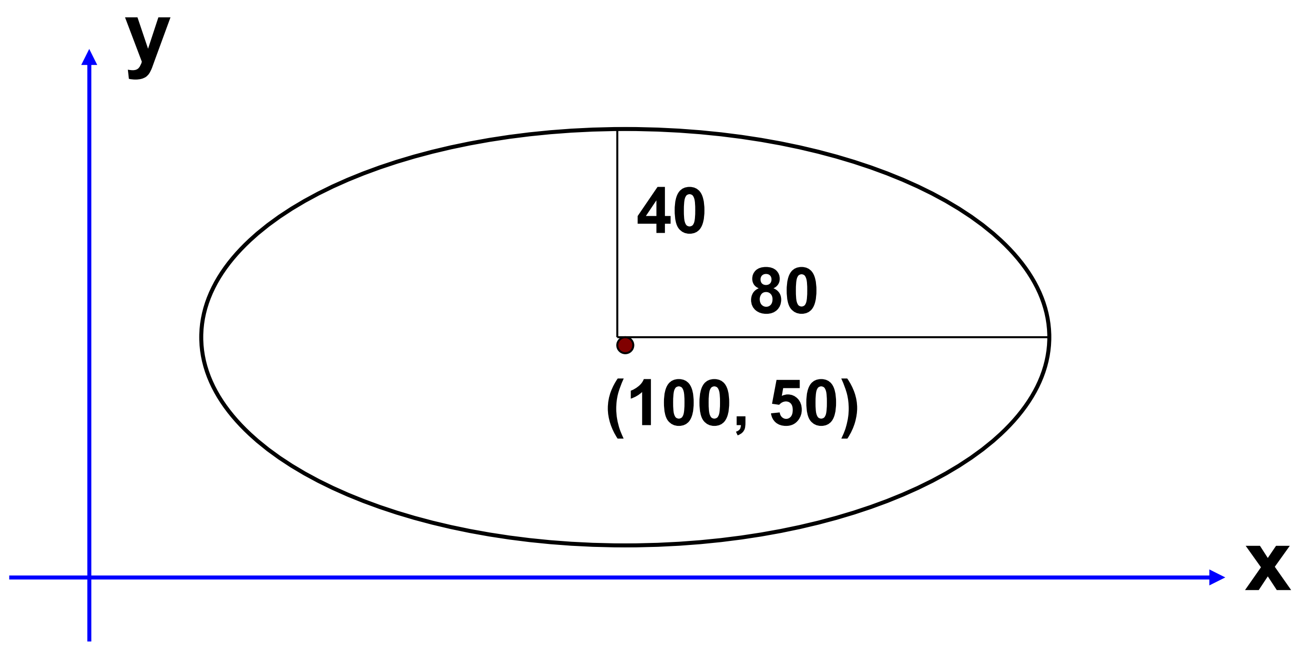
1. Class MyEllipse is initialized by centerX, centerY, a, b
2. Inside the construcor of MyEllipse, the constructor of class Ellipse is called

Introduction to Adapter Pattern

客户类只需要使用构造方法如下

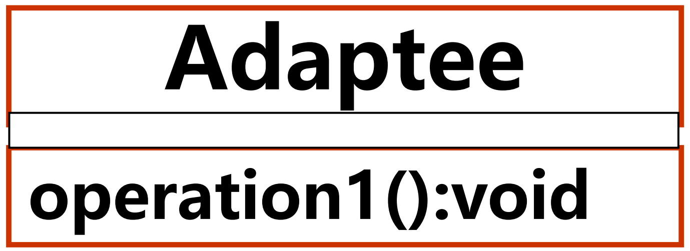
MyEllipse e = new MyEllipse (100, 50, 80, 40);

即可得到如下的椭圆



Introduction to Adapter Pattern

Example 2. 功能不足问题
Insufficient Functionality-New Functionality is needed



Question: we want to use operation1() in an existing class called Adaptee, and also we need another operation operation2(), which is not in class Adaptee.

How to solve this problem?

Introduction to Adapter Pattern

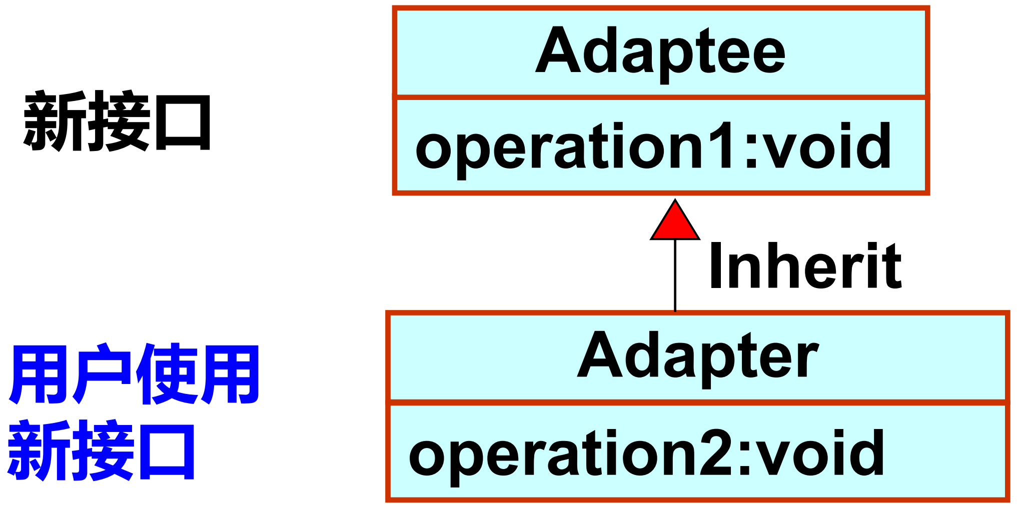
- **Solution 1 (解决方案1):**
  Modify class Adaptee to add method operation2().

- **This solution is not practical, the reason:**
  - You cannot get the source code
  - Even you have the source code, after you modify the file, you need to recompile it, which may cause some new problems.
  - If this class has already been used by some users, then you need to consider side effects of your change.

- **Any other solutions?**

Further Discussion of the Adapter Pattern

- **Solution 2 (解决方案2):**
- Use a class Adapter to inherit Adaptee, and add the operation2() method.

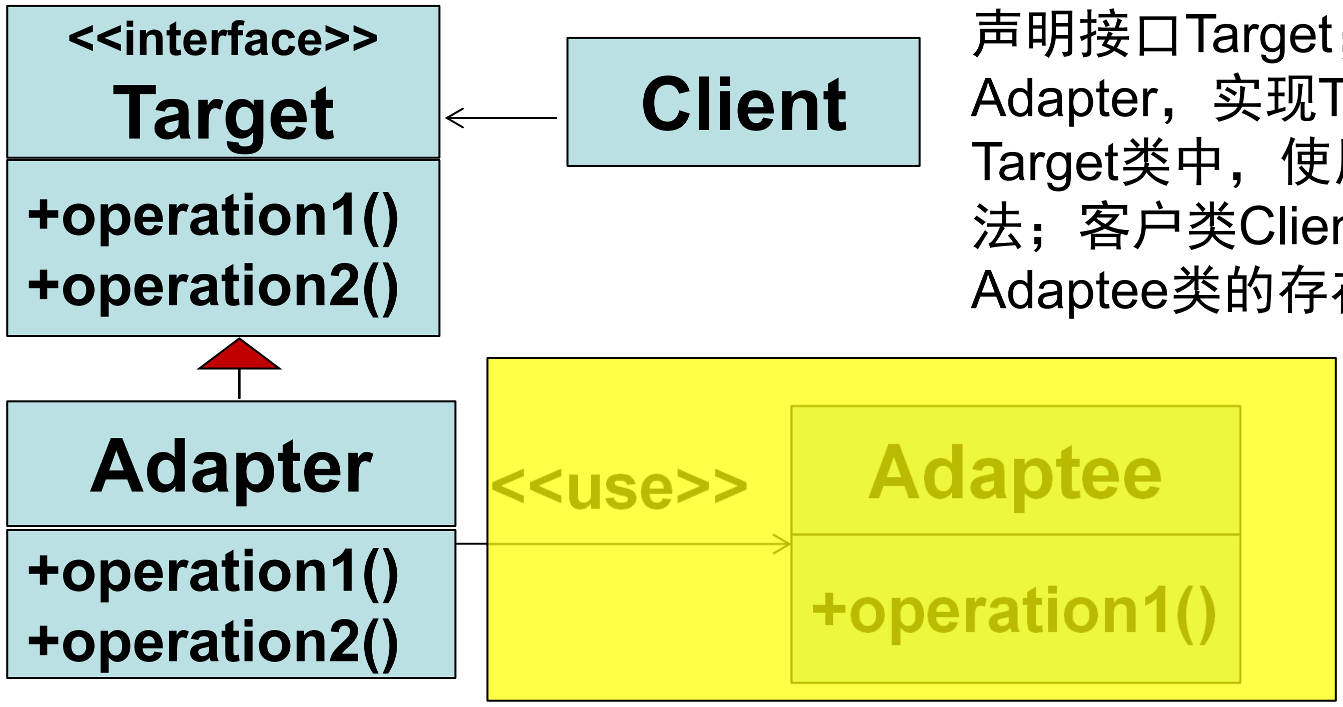


点评：这是1990年代的做法；这是一种传统的通过继承增加功能的方法。这种设计不利于扩展。

Introduction to Adapter Pattern

Solution 3 (解决方案3):

Use an interface called Target to claim all the operations needed, and use a class called adapter to implement all the operations in Target. 这是适配器模式的思想

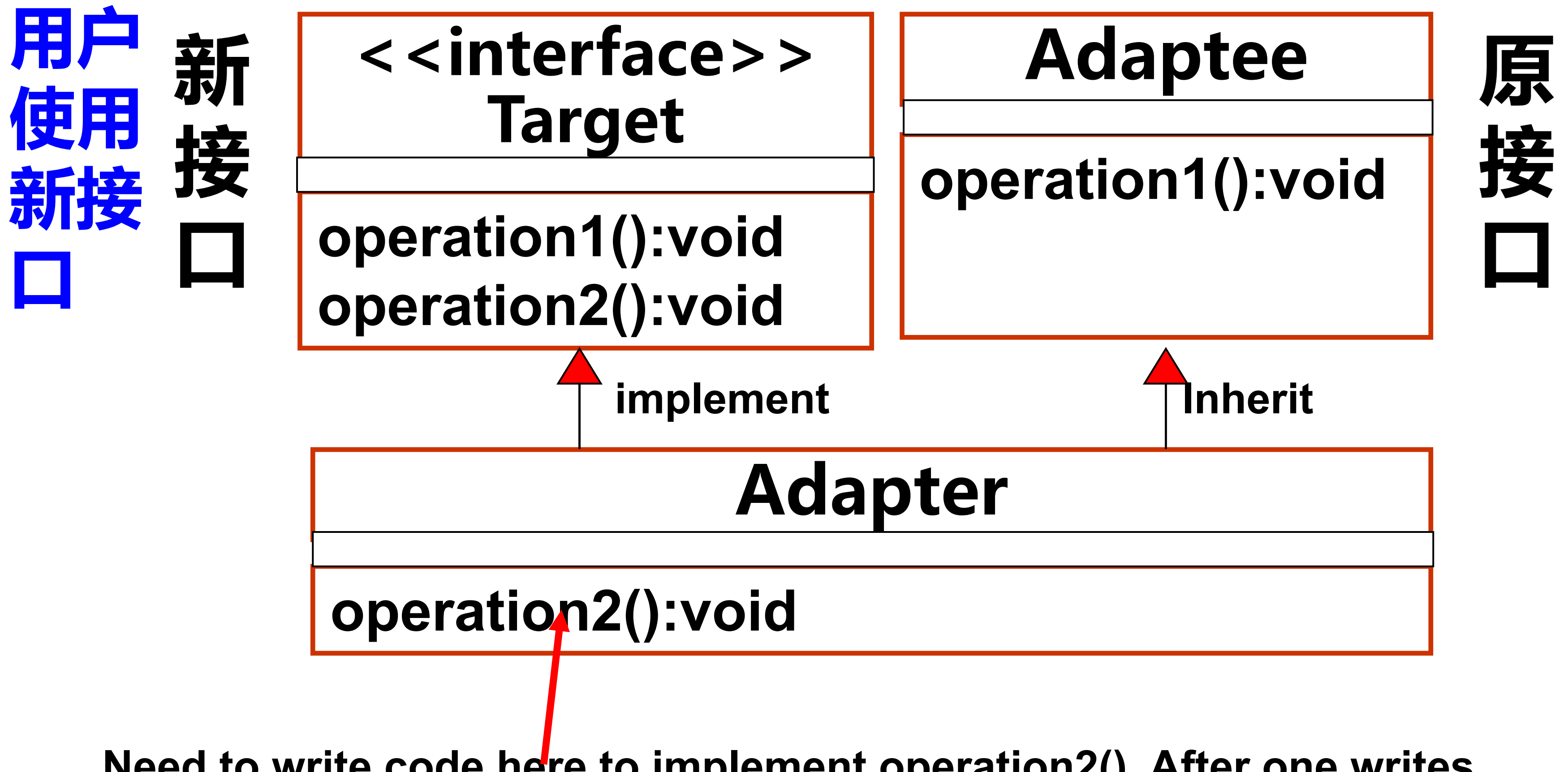


声明接口Target；用一个适配器类Adapter，实现Target接口；在Target类中，使用Adaptee中的方法；客户类Client可以无视Adaptee类的存在。

Back

Class Adapter Pattern
类适配器模式

Class Adapter Pattern



Need to write code here to implement operation2(). After one writes code for operation 2(), then both operation1() and operation2() can be used.

Class Adapter Pattern-class Diagram (类适配器模式)

Class Adapter Pattern

客户程序中，怎样使用适配器模式？

Question: How to use this pattern?

```
Target  tgt;
tgt = new Adapter();
tgt.operation1();
tgt.operation2();
```

Why?

Class adapter pattern

怎样设计适配器模式?

Step1. Create an interface called Target that claims methods operation1() and operation2().

Step2. Create a class called **Adapter** that

- –inherits Adaptee and
- –Implements interface called Target

Note: operation1() has been automatically implemented, and we only need to write code for Operation2() in class Adapter.

Class adapter pattern

Adapter pattern in Java

- **Target:** the expected Java interface
- **Adaptee**: current interface for inheritance
- **Adapter**: convert the Adaptee interface into the Target interface and some functionalities may be added. Adapter is a concrete class.

Class adapter pattern-sample source code

示意性代码

```
public class Adaptee { //原接口
    public void operation1() {
      System.out.print(“ This is an existing method.”)
    }
}

public interface Target { //新接口，声明所有的方法
    void operation1();
    void operation2();
}
```

Class adapter pattern-sample source code

```
//适配器类
public class Adapter extends Adaptee
   implements Target {
   //类Adaptee不包含Operation2(), 所以要在此实现

   public void operation2() {
       // 写代码实现此方法
   }
}
```

Operation1()被认为已经包含在Adapter里面了，因为Adapter类继承了Adaptee.

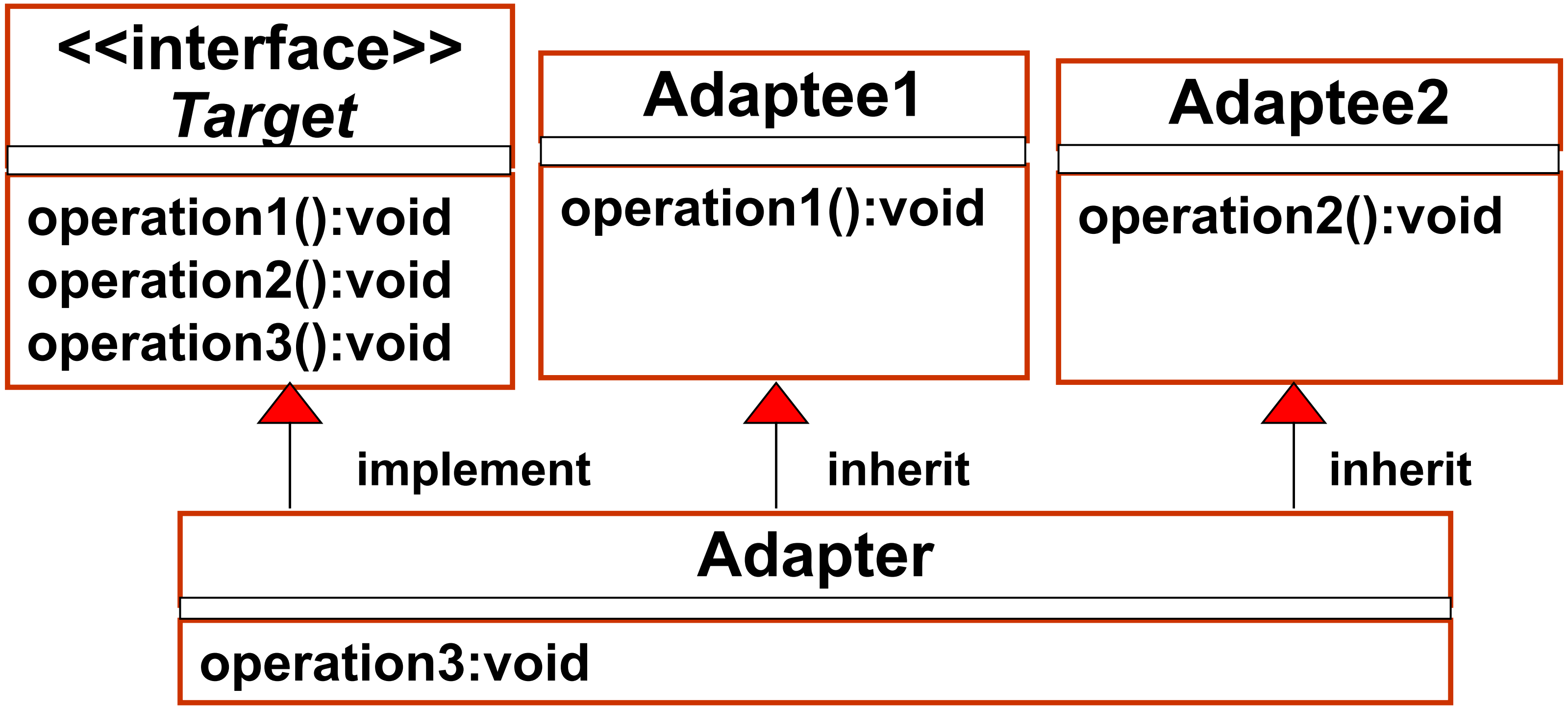
Class adapter pattern-sample source code

```
public class Client {
   private static Target adp;  // use the interface
   as type

    public static void main (String[] args){
            adp = new Adapter();
            adp.operation1();
            adp.operation2();
    }
}
```

Class adapter pattern- question

- **问题: 能否同时适配两个类**?
- If there are two existing classes **Adaptee1** and **Adaptee2,** can we still use the class adapter pattern as below?



Class adapter pattern- answer

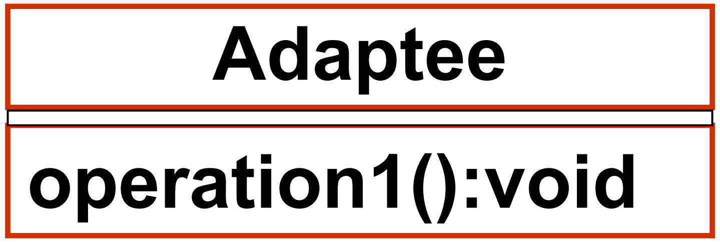
Answer:

1. in C++, this design is **OK**, but multiple inheritance may sometimes introduce complexity
2. In Java, this design is **not allowed** since in Java, multiple inheritance is not allowed

Back

Object Adapter Pattern
对象适配器模式

Object adapter pattern (对象适配器模式)

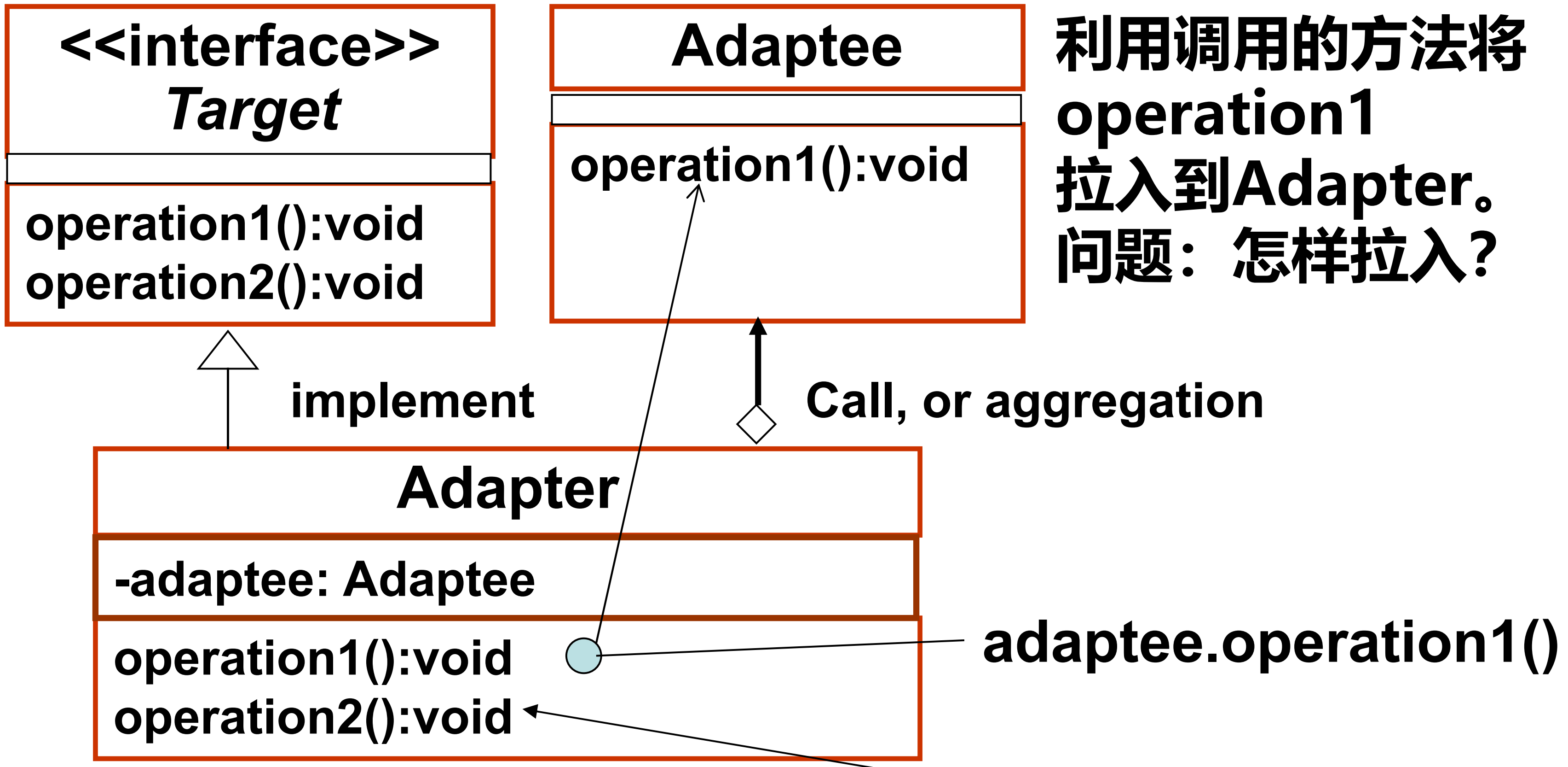


Question (same as before):

- operation1() in Adaptee is what we need, but we also need another method operation2()
- However, there is no method operation2() in class Adaptee

1. Same question as that in using “class adapter pattern”.
2. We now use object adapter pattern, to solve this problem.

Object adapter pattern (对象适配器模式)



利用调用的方法将operation1拉入到Adapter。问题：怎样拉入?

需要写全新的代码

Object adapter pattern-class diagram

How to design the object adapter pattern?

a) Write an interface called Target that claims methods operation1() and operation2().

b) Write a an Adapter that implements methods operation1() and operation2() in the interface Target

c) 怎样实现两个方法?

- **For operation1()**

  Inside the adapter, write code

  ```
  private Adaptee v; // 是否有其它方法？
  v = new Adaptee();
  v. operation1()
  ```

- **For operation2()**

  write new code

Object adapter pattern-sample source code

```
public class Adaptee  {
   public void operation1() {
     System.out.println(“ operation 1 code.”);
   }
}

public interface Target {
   void operation1();
   void operation2();
}
```

Object adapter pattern-sample source code

```
public class Adapter implements Target {
      private Adaptee adaptee;

      public Adapter(Adaptee adaptee) {//由参数传入
            this.adaptee = adaptee;         //adaptee对象
      }
      public void operation1()   {
            adaptee.operation1(); //调用
      }
      public void operation2()  {
         // 写新代码
         System.out.print("Need code operation2 .");
      }
}
```

When to use the Adapter pattern?

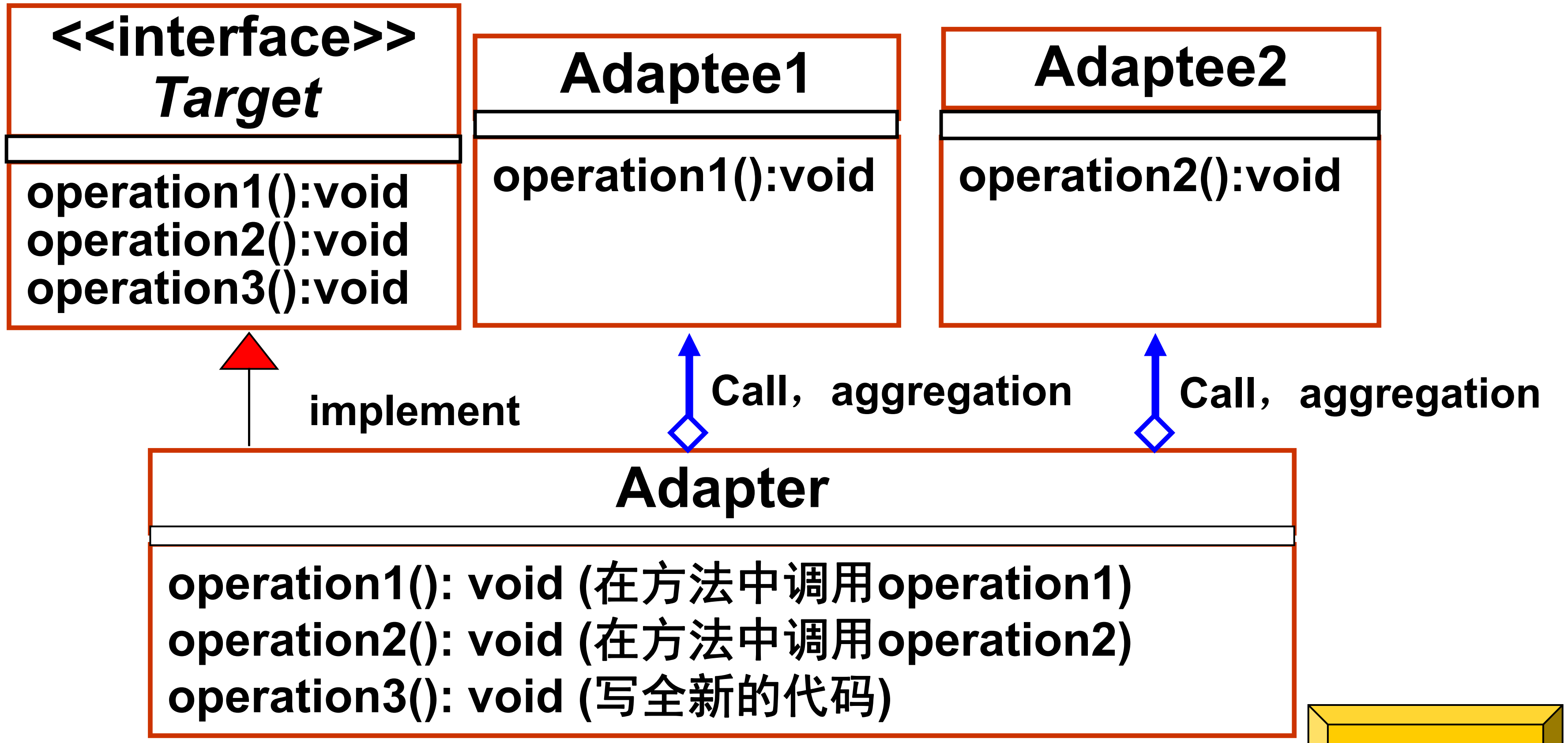
何时使用适配器模式？

Use the Adapter pattern when

- **You want to use an existing class, and its interface does not match the one you need or**
- **You want to create a reusable class that cooperates with unrelated classes with incompatible interfaces, or**
- **In a design, you need to change the interface of many subclasses. In this case, use object adapter.**

Class adapter pattern- question

- **问题: 在对象适配器模式中，能否同时适配两个类**?
- If there are two existing classes **Adaptee1** and **Adaptee2,** can we still use the object adapter pattern as below?



Answer: Yes

Back

使用适配器模式进行设计的例子

使用适配器模式进行设计的例子

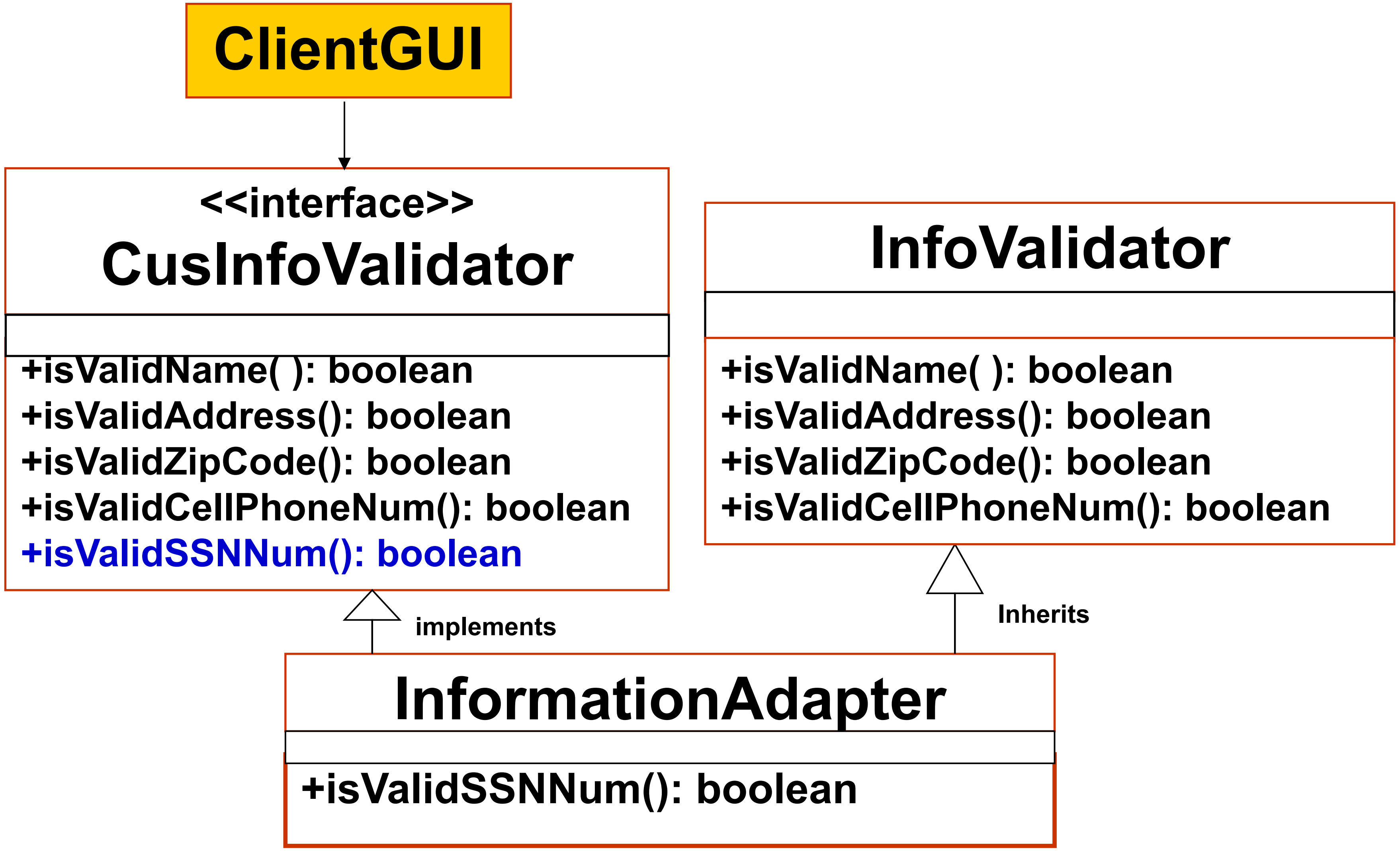
- Example 3. 离架软件，功能不足，欲增加新功能。
  Suppose that we have purchased an off-shelf class InfoValidator that validates customer information (No source code)。

The functions includes:
- Validate user name
- Validate address
- Validate area phone code
- Validate cell phone number

使用适配器模式进行设计的例子

- The above class provides most of the functionalities we need, however, we still need another function to validate social security number (格式 ddd-dd-dddd).
- Class CusInfoValidator doesn’t contain this function and so we need to write it ourselves.
- In such case, we can use class adapter pattern. See the design below.

使用适配器模式进行设计的例子


ClientGUI
<<interface>>
CusInfoValidator
+isValidName( ): boolean
+isValidAddress(): boolean
+isValidZipCode(): boolean
+isValidCellPhoneNum(): boolean
+isValidSSNNum(): boolean
InfoValidator
+isValidName( ): boolean
+isValidAddress(): boolean
+isValidZipCode(): boolean
+isValidCellPhoneNum(): boolean
implements
Inherits
InformationAdapter
+isValidSSNNum(): boolean

Design using class adapter pattern

使用适配器模式进行设计的例子

- In this design, the methods we need
  - isValidName( ): boolean
  - isValidAddress(): boolean
  - isValidZipCode(): boolean
  - isValidCellPhoneNum(): boolean
  - isValidSSNNum(): boolean

  are all included in the interface CusInfoValidator

使用适配器模式进行设计的例子

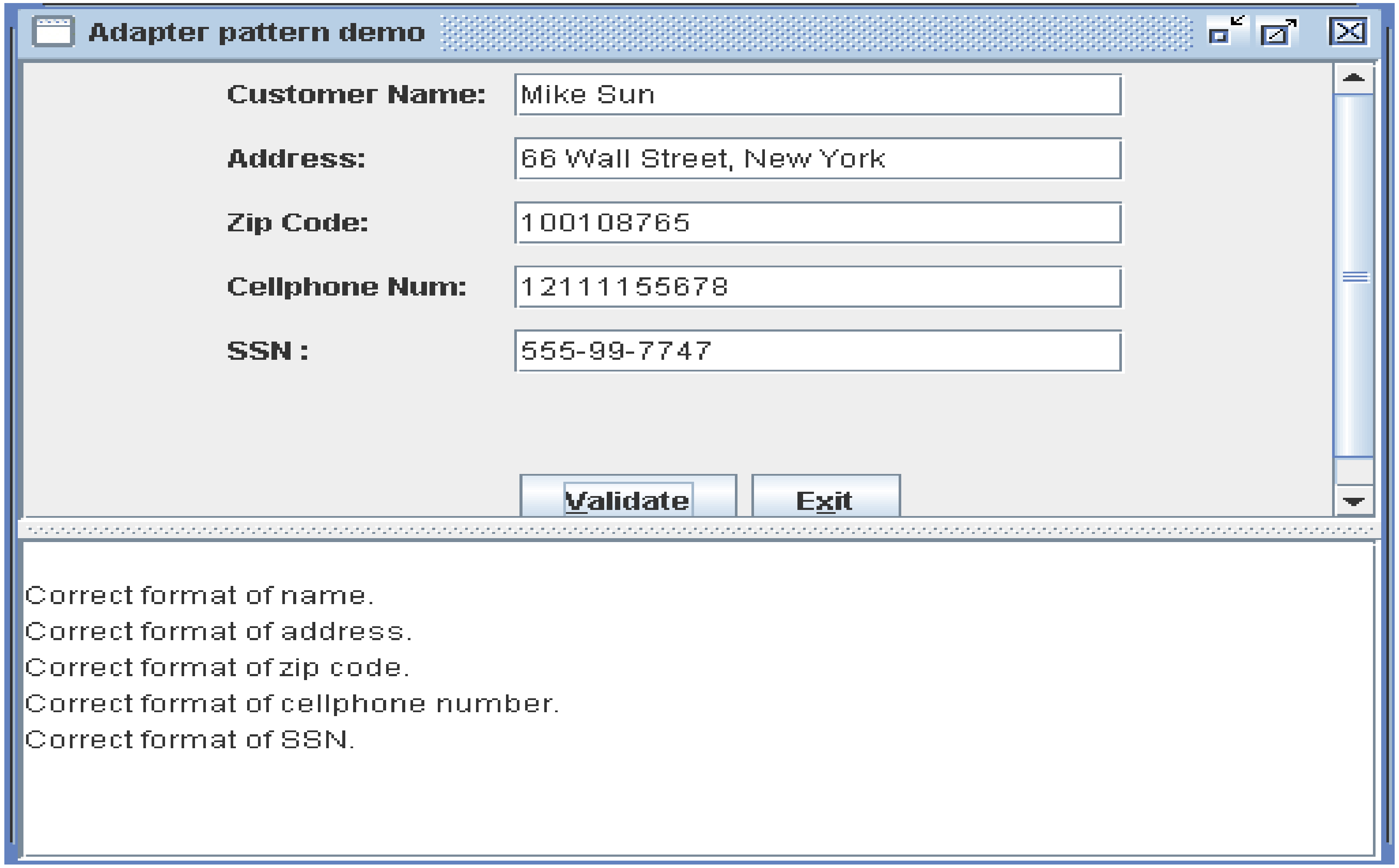
设计类图的解释

- The first 4 methods are included in the off-shelf class **InfoValidator. But the last method** isValidSSNNum() is **not** included in InfoValidator.

- InformationAdapter is responsible for implementing isValidSSNNum()

- Because InformationAdapter has already inherited InfoValidator，the first 4 methods have been implemented

使用适配器模式进行设计的例子



Customer Name:	Mike Sun
Address:	66 Wall Street, New York
Zip Code:	100108765
Cellphone Num:	12111155678
SSN :	555-99-7747

```
Correct format of name.
Correct format of address.
Correct format of zip code.
Correct format of cellphone number.
Correct format of SSN.
```

运行ClientGUI产生的用户图形界面

使用适配器模式进行设计的例子

- **Example 4. 改变接口问题。**
  Customer Address Validation Problem
- **Suppose that we have already had two classes:**
  - **USAddress, a class used to validate a given US address by using a method "isValidAddr"**
  - **CAAddress, a class used to validate a given Canadian address by using a method "isValidCanadianAddr"**

使用适配器模式进行设计的例子

- **在美国客户为主的商业网站上，允许加拿大客户使用该网站进行交易**
- **A website program originally validates US address by using class USAddress.**
- **Now this website also needs to validate Canadian address because some businesses will be expanded to Canadian customers**

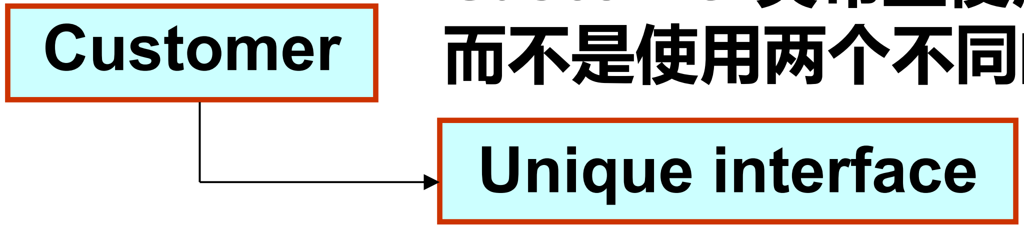
使用适配器模式进行设计的例子

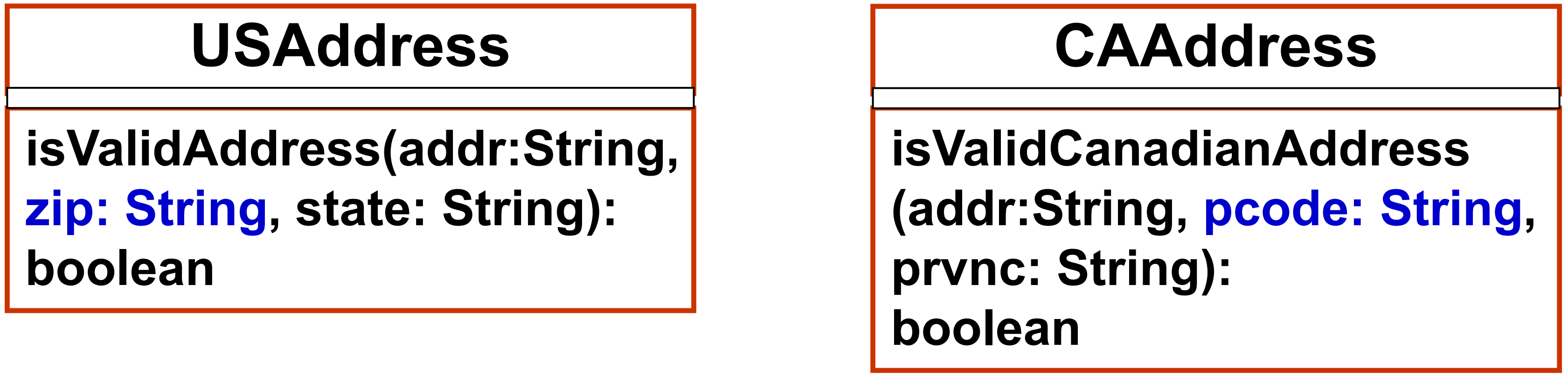
- **客户类需要一个唯一的接口，验证美国地址与加拿大地址。但问题是CAAddress与USAddress有不同的接口。**
- **Customer class needs to use a single interface to validate both addresses.**
- **However, the methods offered by**
  - **CAAddress and**
  - **USAddress**

  have different interfaces

使用适配器模式进行设计的例子

Customer类希望使用一个唯一的接口，而不是使用两个不同的接口。




USAddress
isValidAddress(addr:String, zip: String, state: String): boolean
CAAddress
isValidCanadianAddress (addr:String, pcode: String, prvnc: String): boolean

Zip: 58105-2459

pCode: H1C 3W2

The parameters in isValidAddress and isValidCanadianAddress have different formats

使用适配器模式进行设计的例子

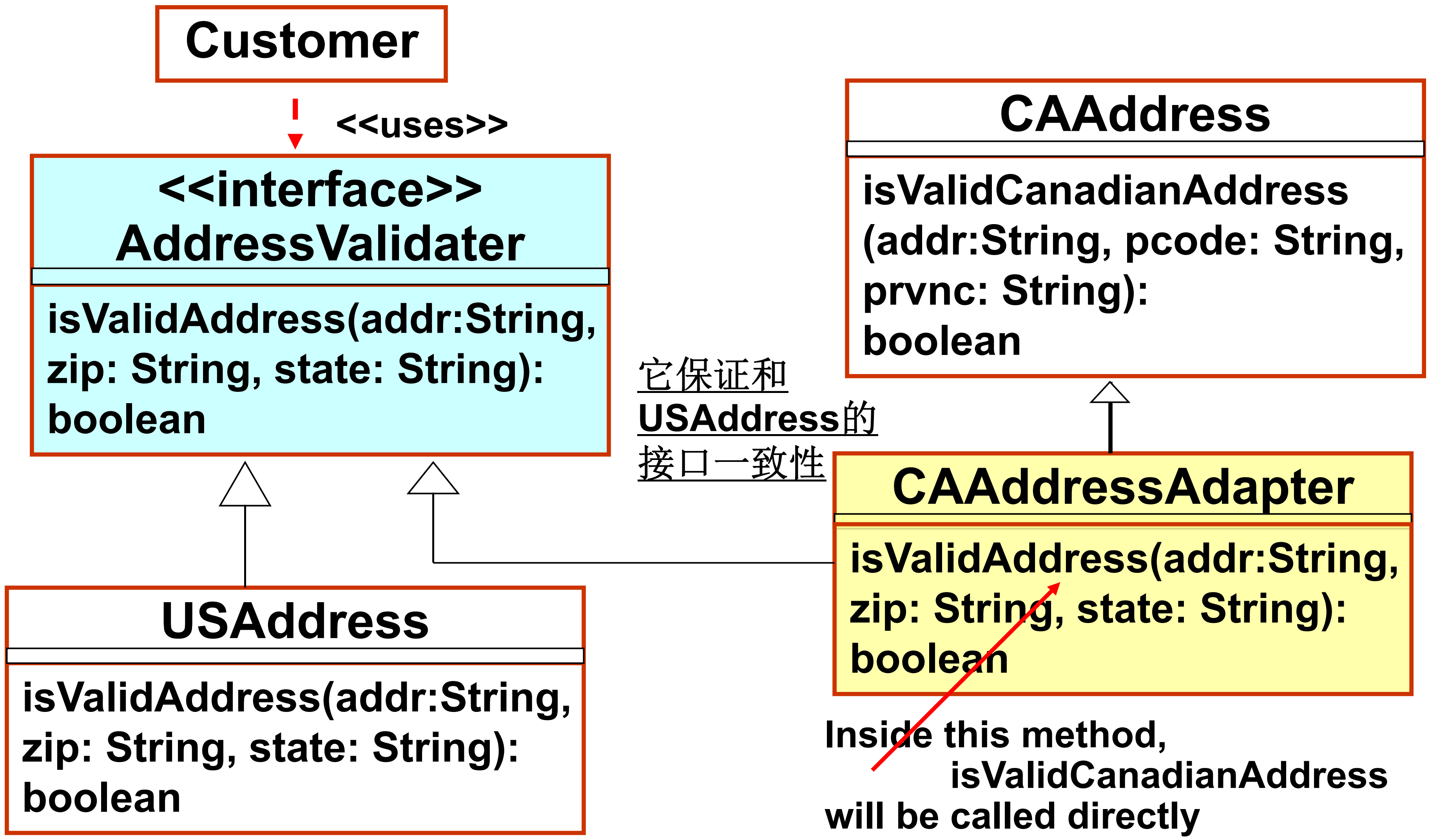
- **不相容的接口使得客户类很难直接使用类 CAAddress**
- **This incompatibility in the interface makes it difficult for a Customer object to use the existing CAAddress class.**
- **How to solve this problem?**

使用适配器模式进行设计的例子

使用类适配器模式：使用一个适配器改变CAAddress的接口

- **Need to design a Java interface called AddressValidator**
- **Need an adapter class CAAddressAdapter, which**
    - **Inherits CAAddress class**
    - **implements the AddressValidator interface**

使用适配器模式进行设计的例子



Customer
<<uses>>
<<interface>> AddressValidater
isValidAddress(addr:String, zip: String, state: String): boolean
CAAddress
isValidCanadianAddress (addr:String, pcode: String, prvnc: String): boolean
它保证和USAddress的接口一致性
CAAddressAdapter
isValidAddress(addr:String, zip: String, state: String): boolean
USAddress
isValidAddress(addr:String, zip: String, state: String): boolean
Inside this method, isValidCanadianAddress will be called directly

使用类适配器模式进行的设计

使用适配器模式进行设计的例子

Inside the method isValidAddress in class CAAddressAdapter,

the method isValidCanadianAddress(addr:String,
pcode: String,
prvnc: String)

is called.

使用适配器模式进行设计的例子

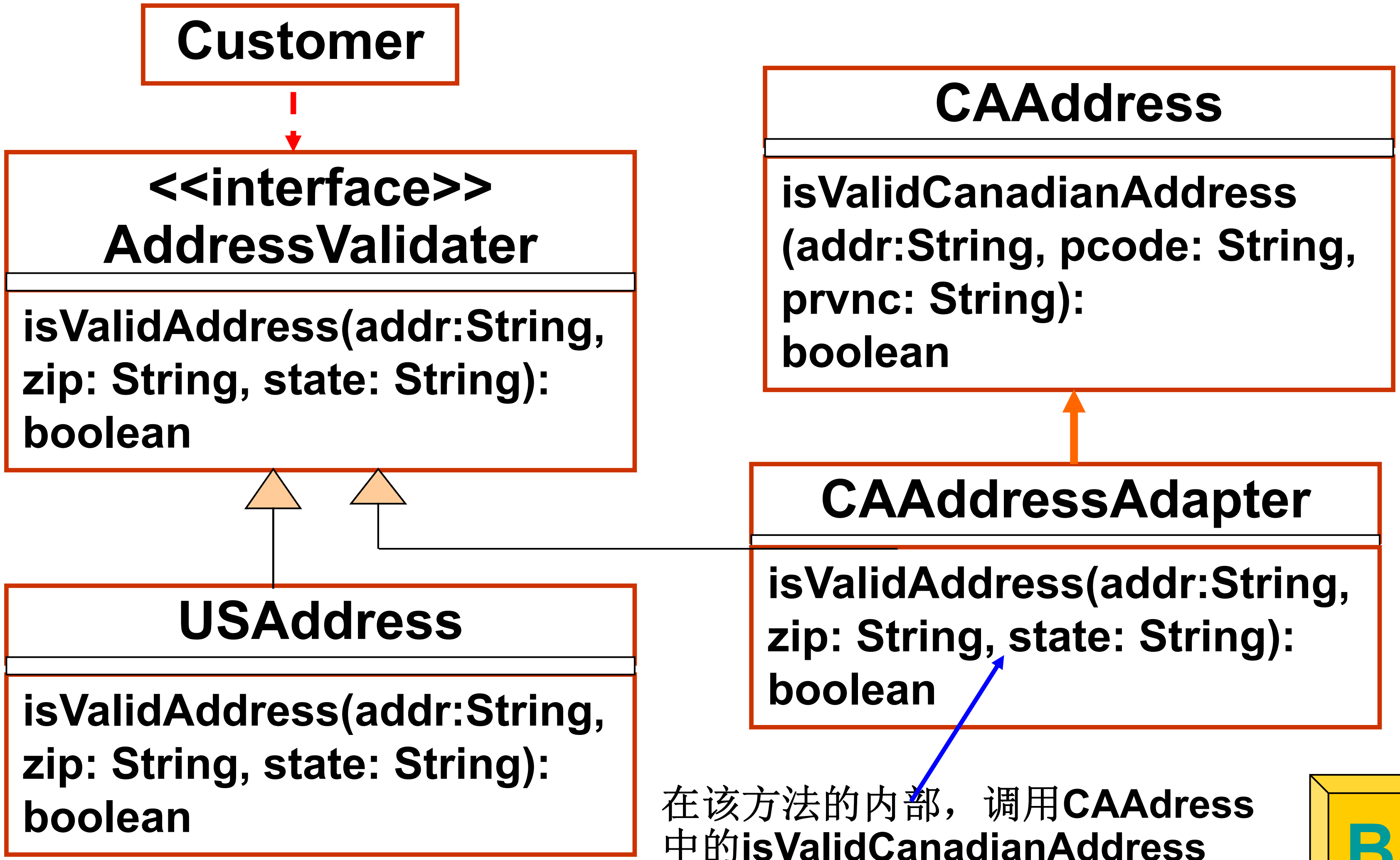
How to use the pattern?

```
AddressValidater av;  //The type of the interface
if(user chooses US)
    av = new USAddress();
else if (user chooses Canada)
    av = new CAAddressAdapter();

av. isValidAddress(addr, pcode, state);
```

使用适配器模式进行设计的例子

也可以使用对象适配器进行设计

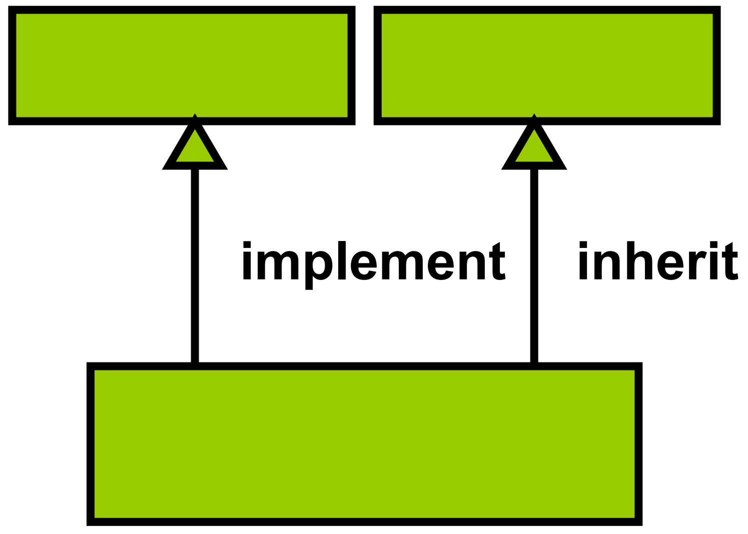

Customer
<<interface>>
AddressValidater
isValidAddress(addr:String, zip: String, state: String): boolean
CAAddress
isValidCanadianAddress (addr:String, pcode: String, prvnc: String): boolean
CAAddressAdapter
isValidAddress(addr:String, zip: String, state: String): boolean
USAddress
isValidAddress(addr:String, zip: String, state: String): boolean

在该方法的内部，调用CAAdress中的isValidCanadianAddress方法

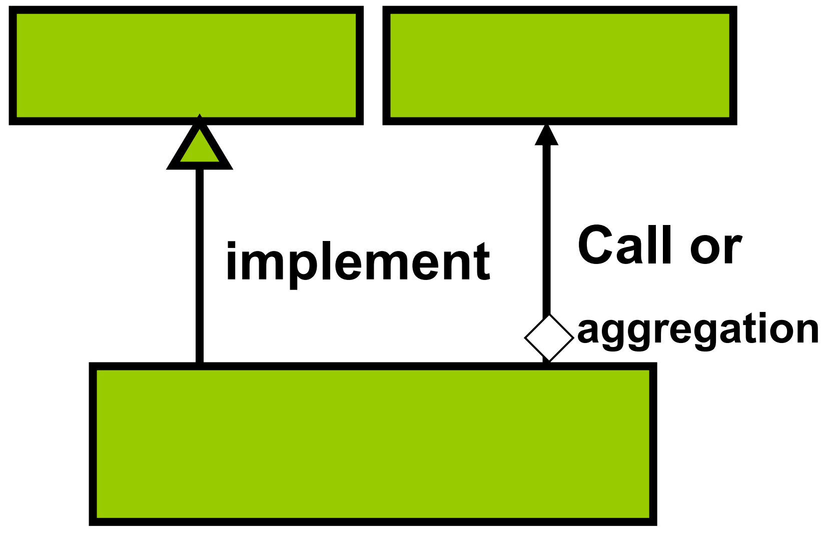
Back

Further Discussion of the Adapter Pattern

Further Discussion of the Adapter Pattern



Class adapter pattern



Object adapter pattern

Comparison of class adapter pattern and object adapter pattern

Further Discussion of the Adapter Pattern

适配器模式的应用主要体现在两个方面

1. 改变接口
2. 增加功能

问题: 这两个方面哪个是主要的呢?

回答: 改变接口。实际上增加功能也可以看作是改变接口，因为增加了功能也就改变了接口。

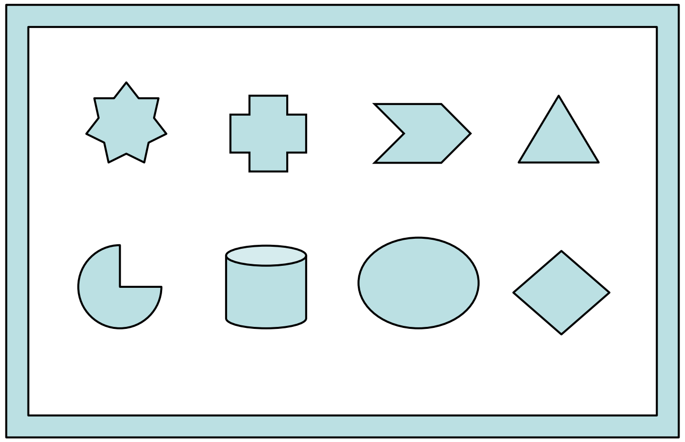
Further Discussion of the Adapter Pattern

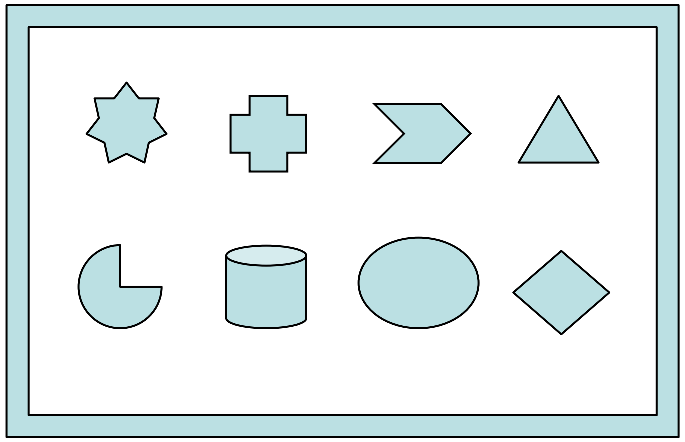
类适配器模式与对象适配器模式的区别

Difference between class adapter pattern and object adapter pattern:

- In the class adapter pattern, all the attributes and methods are inherited 继承类的所有共有方法
- In the object adapter pattern, usually, only one or several methods are chosen to pull into the adapter class 许多类被继承，每个类只是挑选一些方法继承。

Further Discussion of the Adapter Pattern

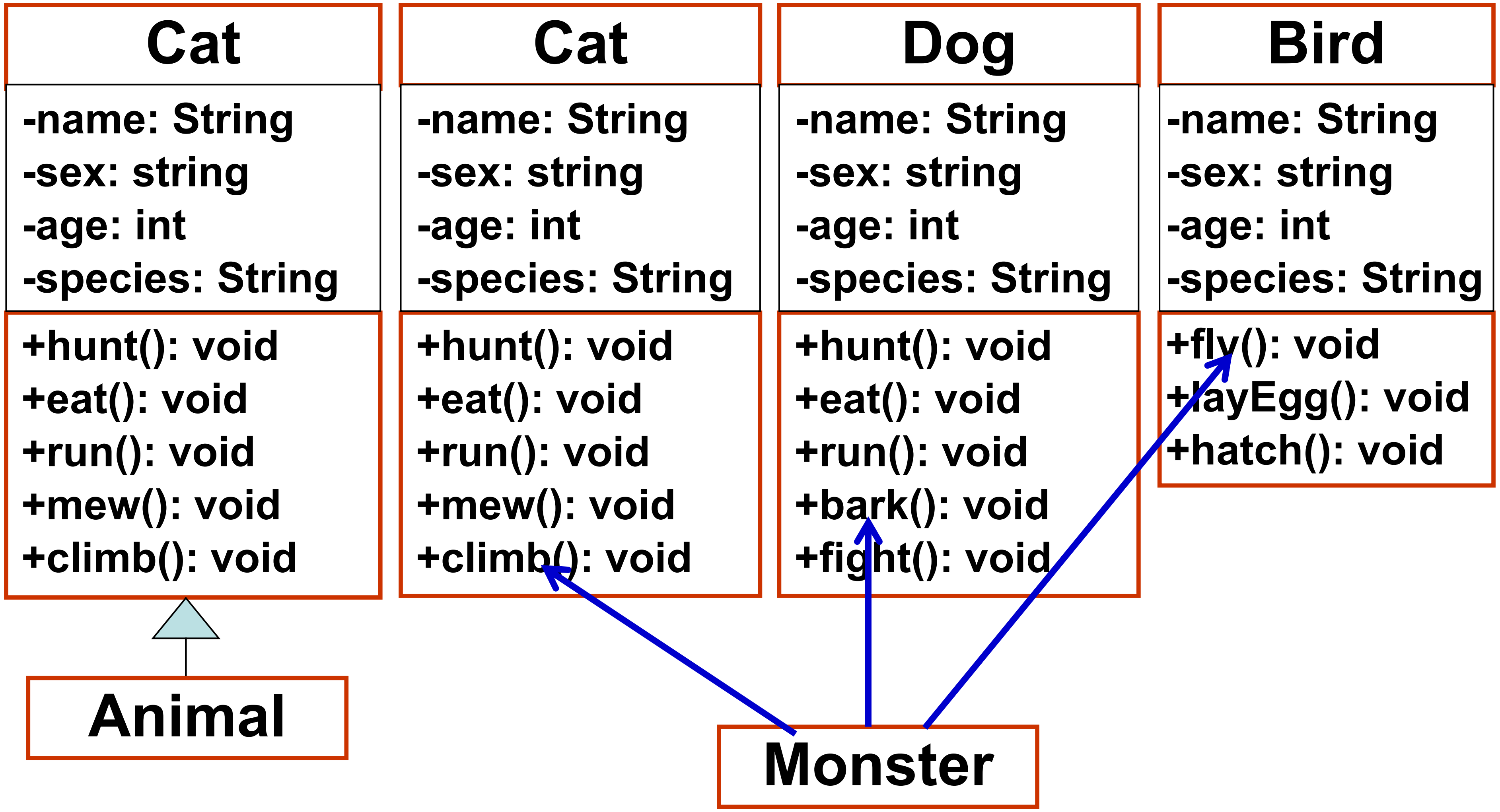




类适配器模式：
继承全部

对象适配器模式：
挑选一个或者几个
东西拉入

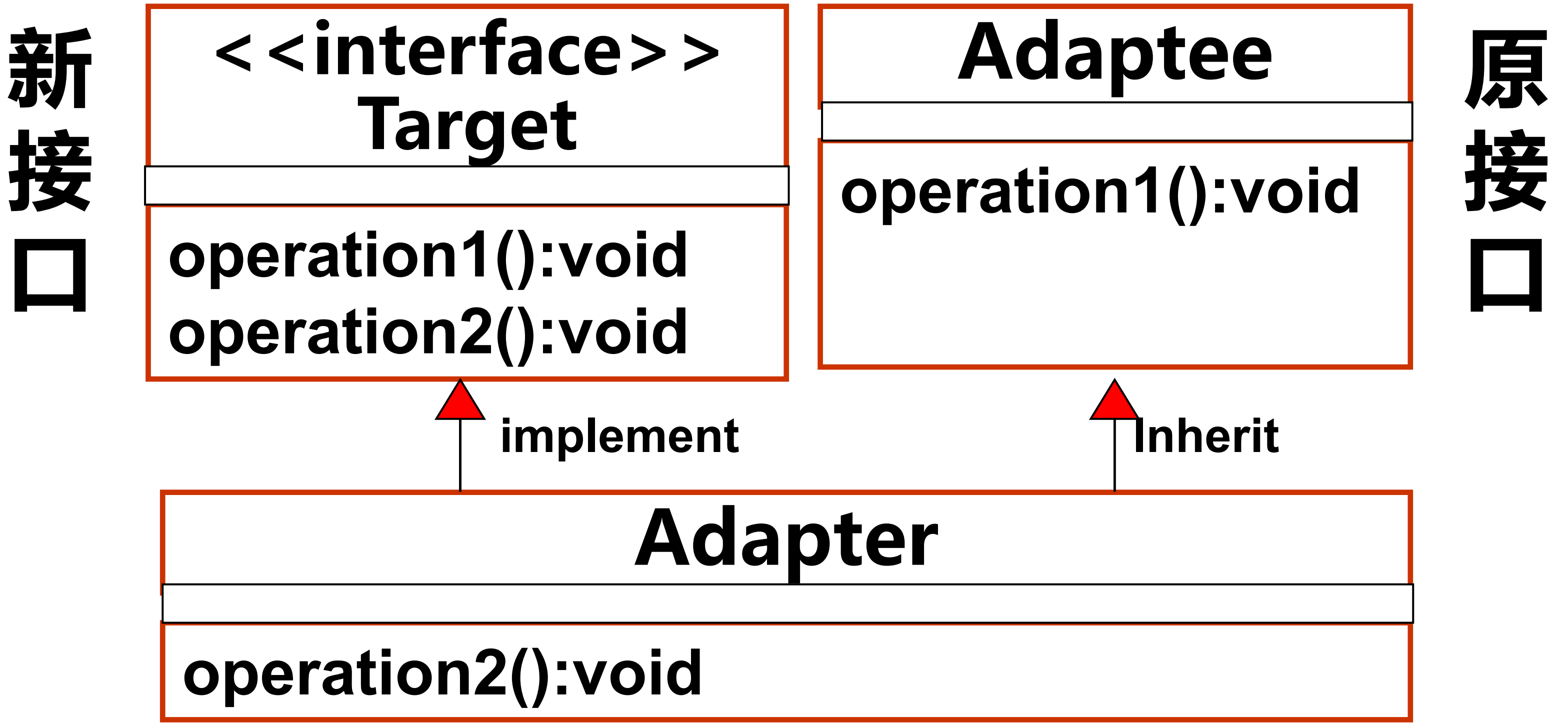
Further Discussion of the Adapter Pattern



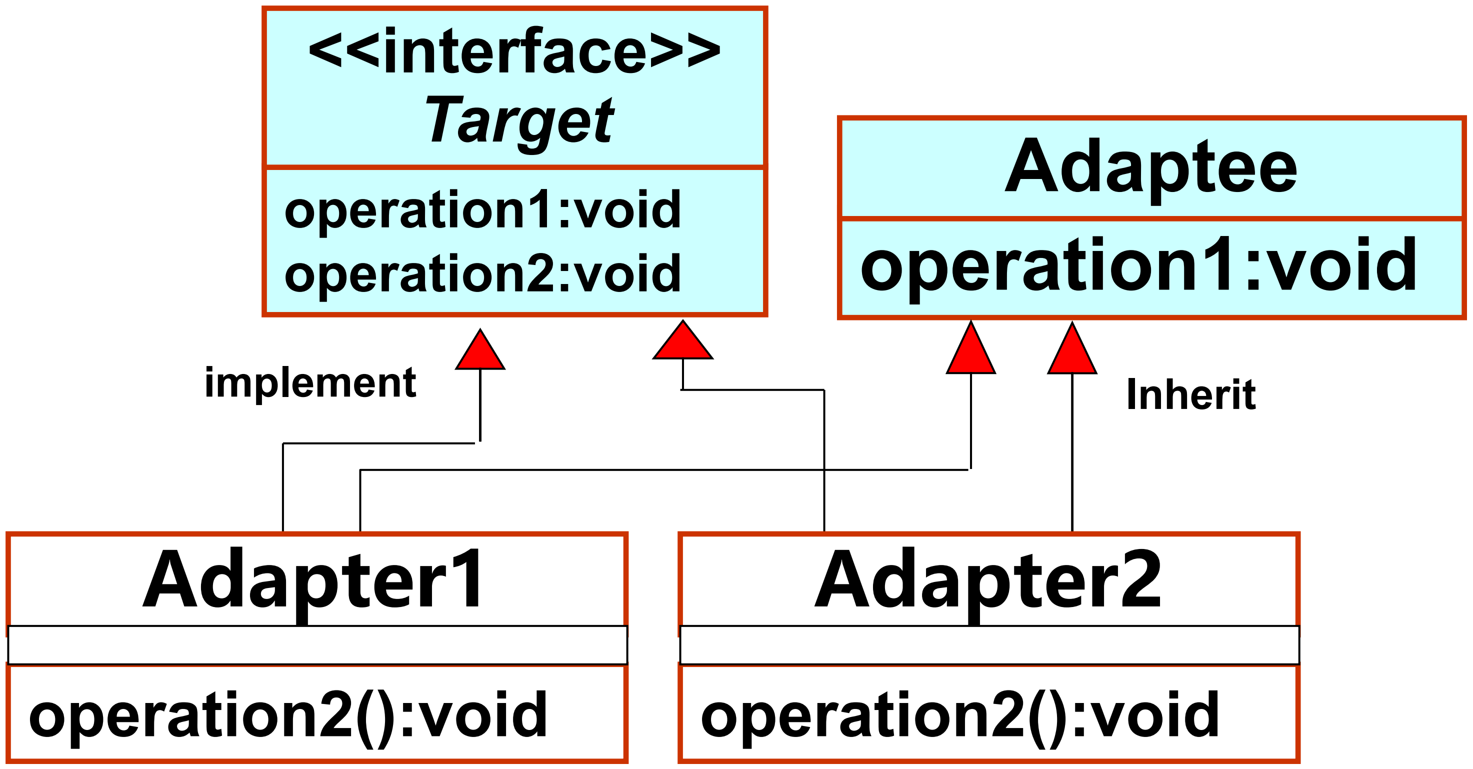
Further Discussion of the Adapter Pattern

- 问题：为什么适配器模式采用了如下奇怪的方式？
- **Discussion**: why in the the adapter pattern, use an interface Target as below?

用户使用新接口



- **Answer**: in the adapter pattern, an interface is used because it can be implemented using many other classes. Example: there are 2 implementations of the design as below.



好处: 同一个接口Target，可以有不同的实现。例如，多种加密算法的实现。

Back