Lecture 4.
Bridge Pattern (桥接模式)
(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

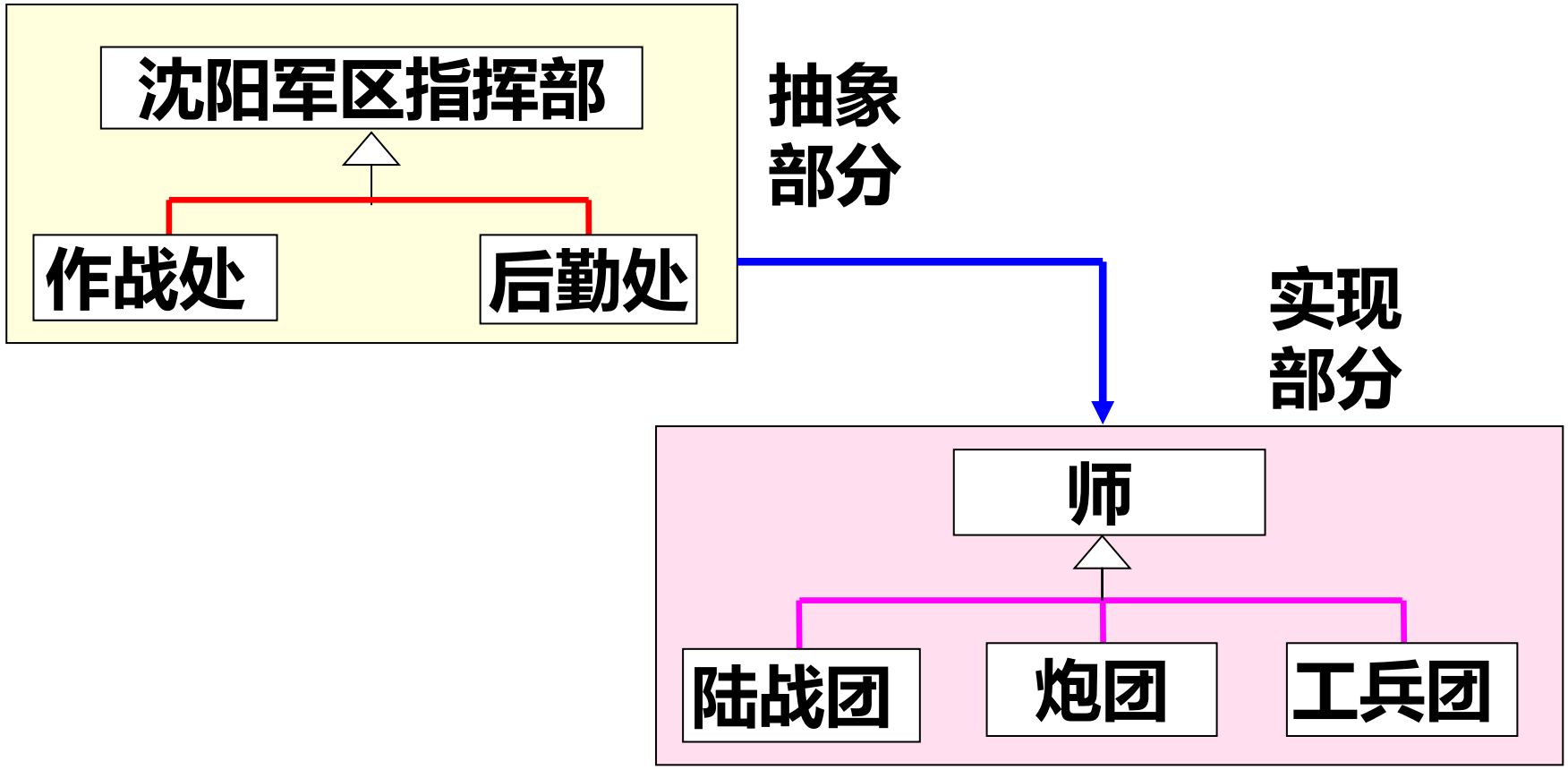
Fall 2020

Contents of this lecture

Introduction of the Bridge Pattern

Introductory Examples

军队机构设置

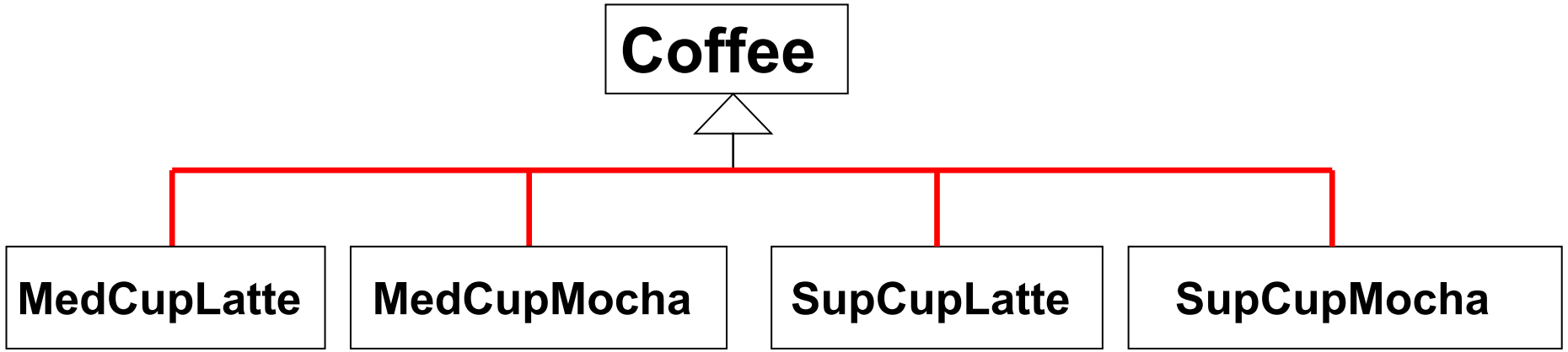


将抽象部分与实现部分分开的军队机构设置

Introductory Examples

Example 1: 咖啡销售机软件
Design of software for coffee seller machine

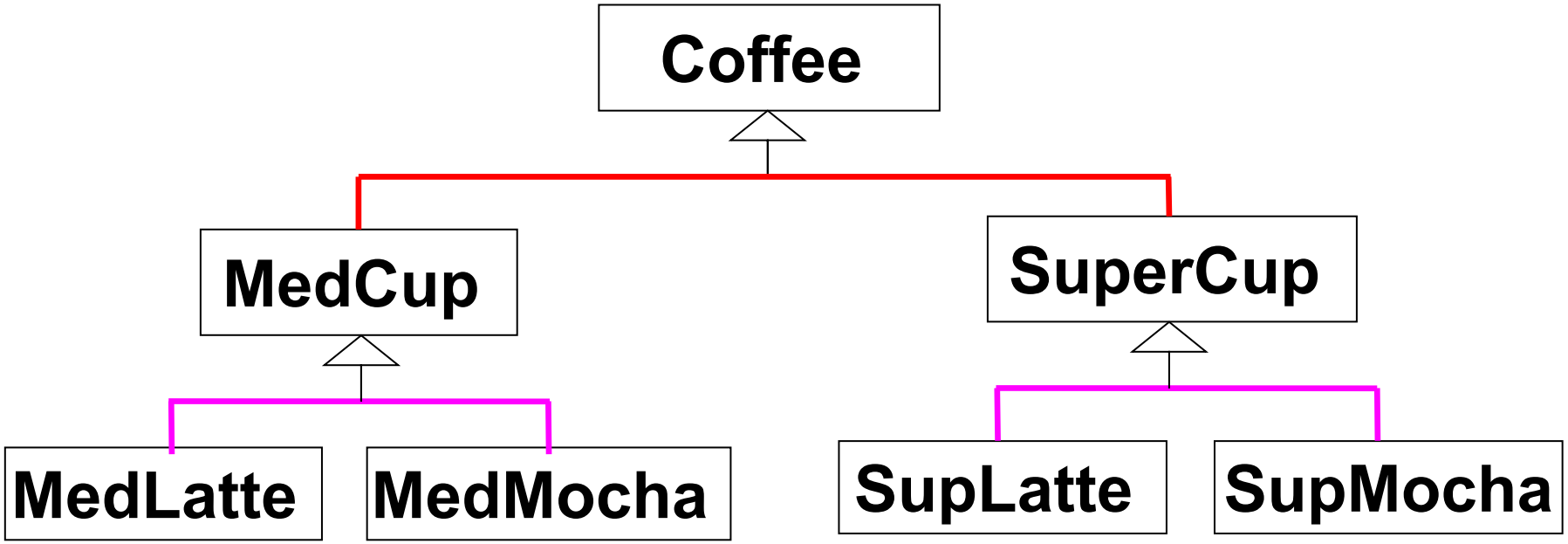
Design 1 （设计1，flat 设计）



缺点：This design is flat, with some repetitions

Introductory Examples

Design 2 (设计2，增加新的层次)

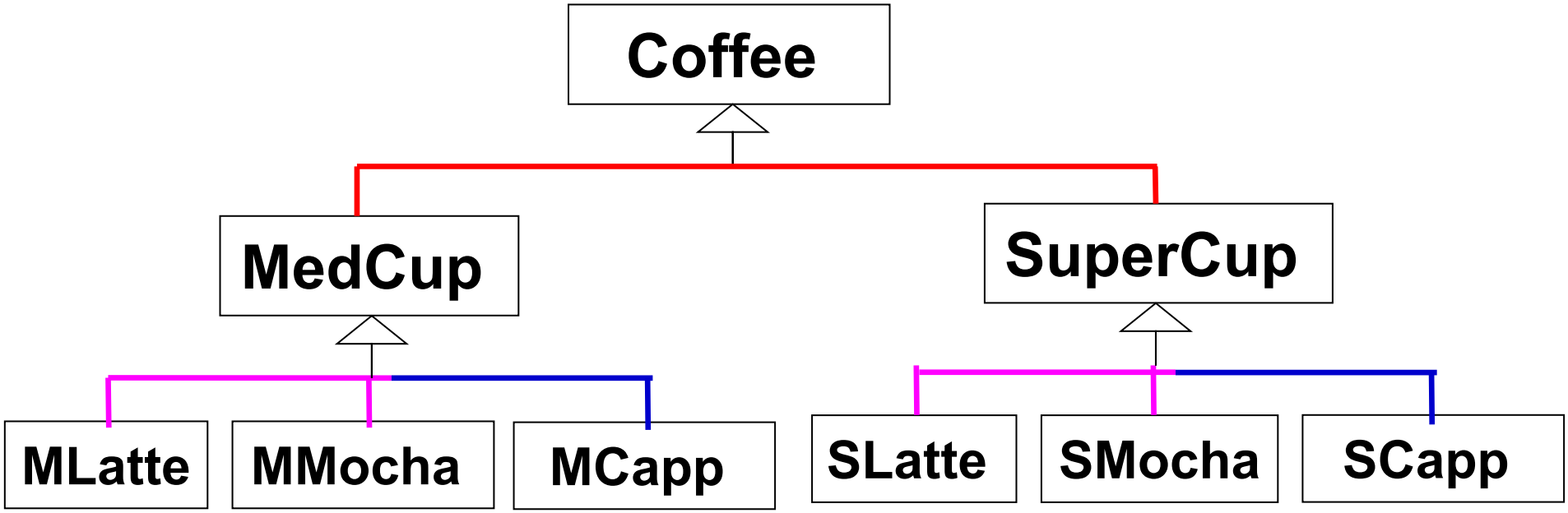


Redesign: Better, but there are three layers. Adding a new coffee needs to add new sub classes in two places

For example: adding Cappuccino

Introductory Examples

Extendibility (可扩展性不好)

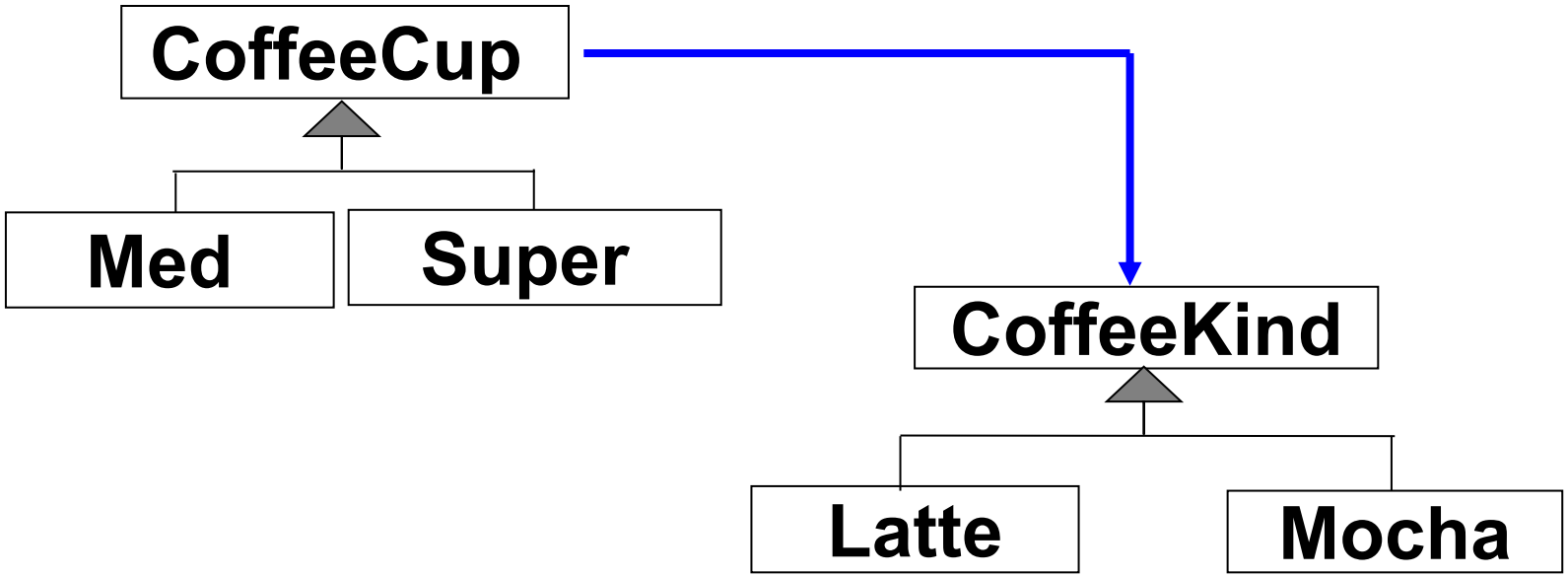


Problem: Adding of a new coffee needs to add a new sub class in two places

Introductory Examples

Design 3 （设计3，将抽象部分与实现部分分开）

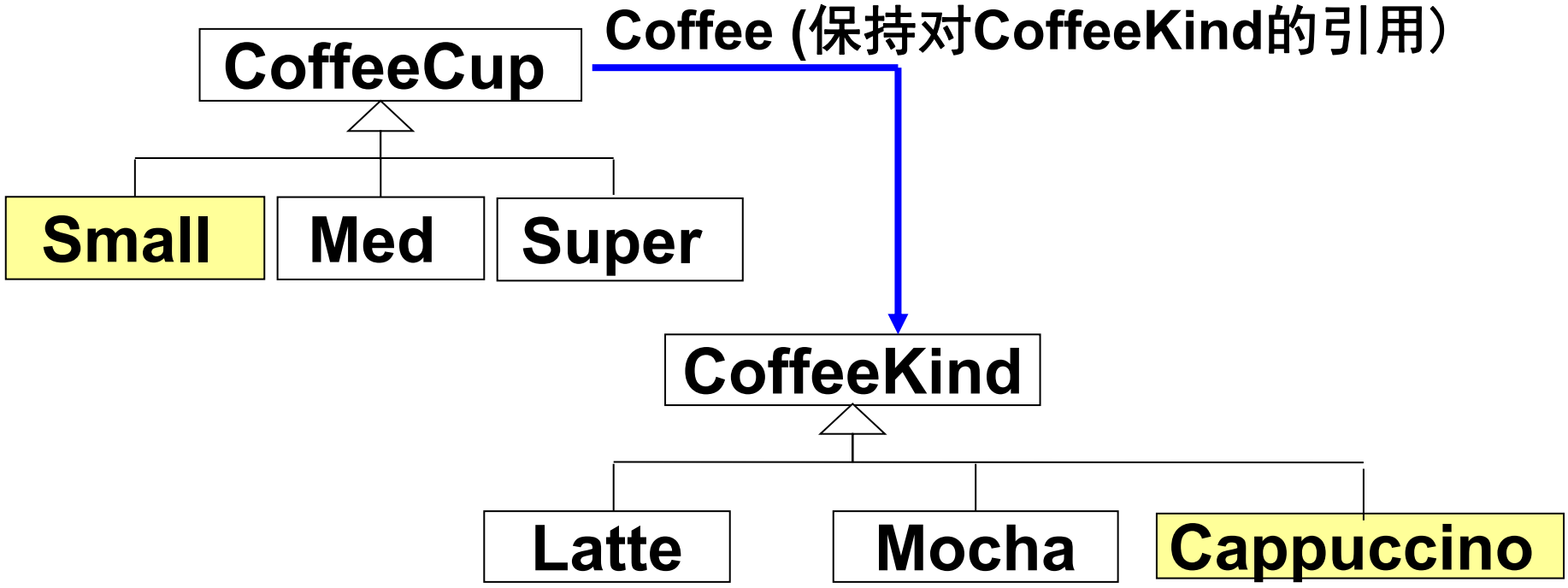
Solution: divide the class hierarchy into two class hierarchies



New design: In classes Med or Super, call Latte or Mocha

Introductory Examples

Extendibility has been improved (可扩展性增强了)

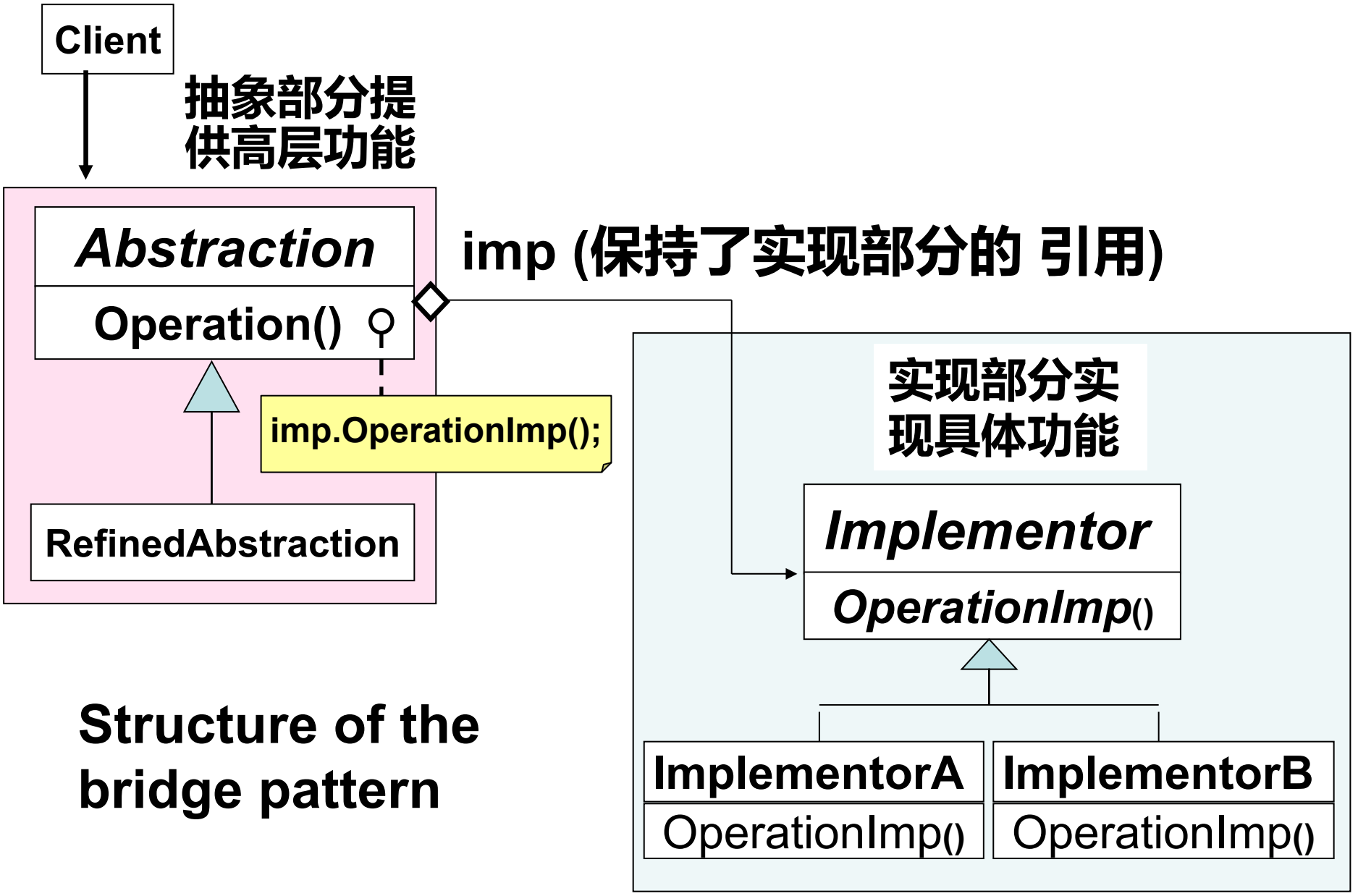


Advantage: One can add new classes in either dimension without affecting other classes。
在两个维度上可以自由地添加类，而不会影响到其它的类

Back

Theory of the Bridge Pattern

Bridge Pattern



Structure of the bridge pattern

Bridge Pattern

桥模式的组件 （Participants）

- **Abstraction**
  - defines the abstraction's interface.
  - **maintains a reference** to an object of type Implementor.
- **RefinedAbstraction**
  - Extends the interface defined by Abstraction.

Bridge Pattern

Implementor

- Defines the interface for implementation classes.
- This interface doesn't have to correspond exactly to Abstraction's interface; in fact the two interfaces can be **quite different**.

Bridge Pattern

Typically

- The Implementor interface provides only primitive operations（实现接口仅提供原始操作)
- The Abstraction defines higher-level operations based on these primitives (抽象接口定义基于原始操作的高层操作)

Bridge Pattern

The ConcreteImplementors:

- **ImplementorA and**
- **ImplementorB**

implement the Implementor interface and defines its concrete implementation.

Advantages of Bridge Pattern

桥模式的优点

1. **将接口与实现解耦。*Decoupling interface and implementation*.**

   An implementation is not bound permanently to an interface
   - The implementation of an abstraction can be configured at run-time （**实现部分动态配置**）
   - Decoupling Abstraction and Implementor also eliminates compile-time dependencies on the implementation （**取消编译时对实现部分的依赖**）
   - Changing an implementation class doesn‘t require recompiling the Abstraction class and its clients（**改变实现类不需要重新编译抽象类与客户类)**

Advantages of Bridge Pattern

2. 改善了可扩展性。*Extensibility Improvement*

   You can extend the Abstraction and Implementor hierarchies **independently**.

3. 对用户隐藏实现细节。*Hiding implementation details from clients*.

   You can shield clients from implementation details, like the sharing of implementor objects and the accompanying reference count mechanism (if any).

Bridge Pattern

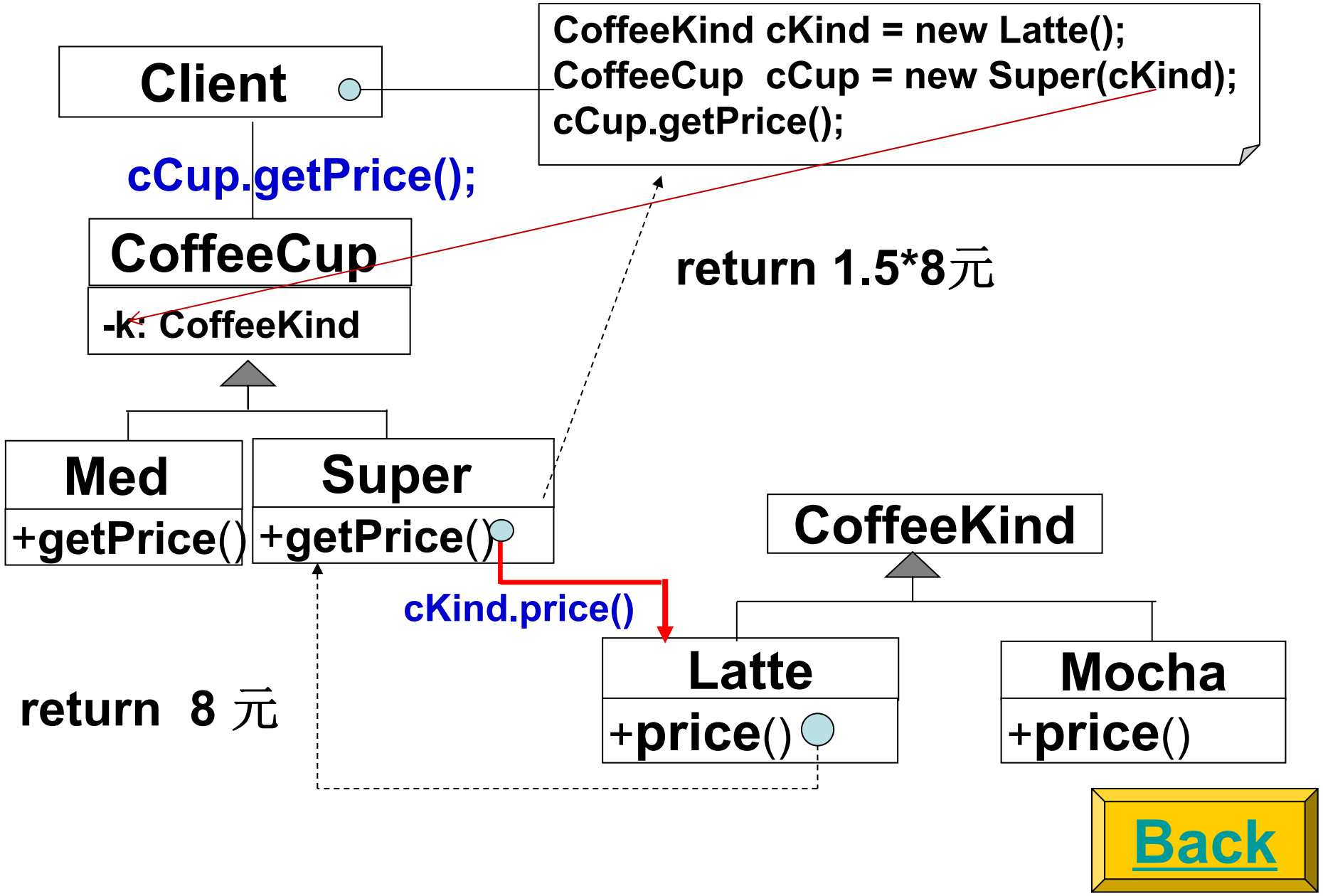
何时使用桥模式?

When to use the bridge pattern?

Applicability: Use the Bridge pattern when you want to avoid a permanent binding between an abstraction and its implementation.

- e. g. when the implementation must be selected or switched at run-time.

桥接模式的典型交互


CoffeeKind cKind = new Latte();
CoffeeCup cCup = new Super(cKind);
cCup.getPrice();
Client
cCup.getPrice();
CoffeeCup
-k: CoffeeKind
return 1.5*8元
Med
+getPrice()
Super
+getPrice()
CoffeeKind
cKind.price()
Latte
+price()
Mocha
+price()
return 8 元
Back

Design Examples Using the Bridge Pattern

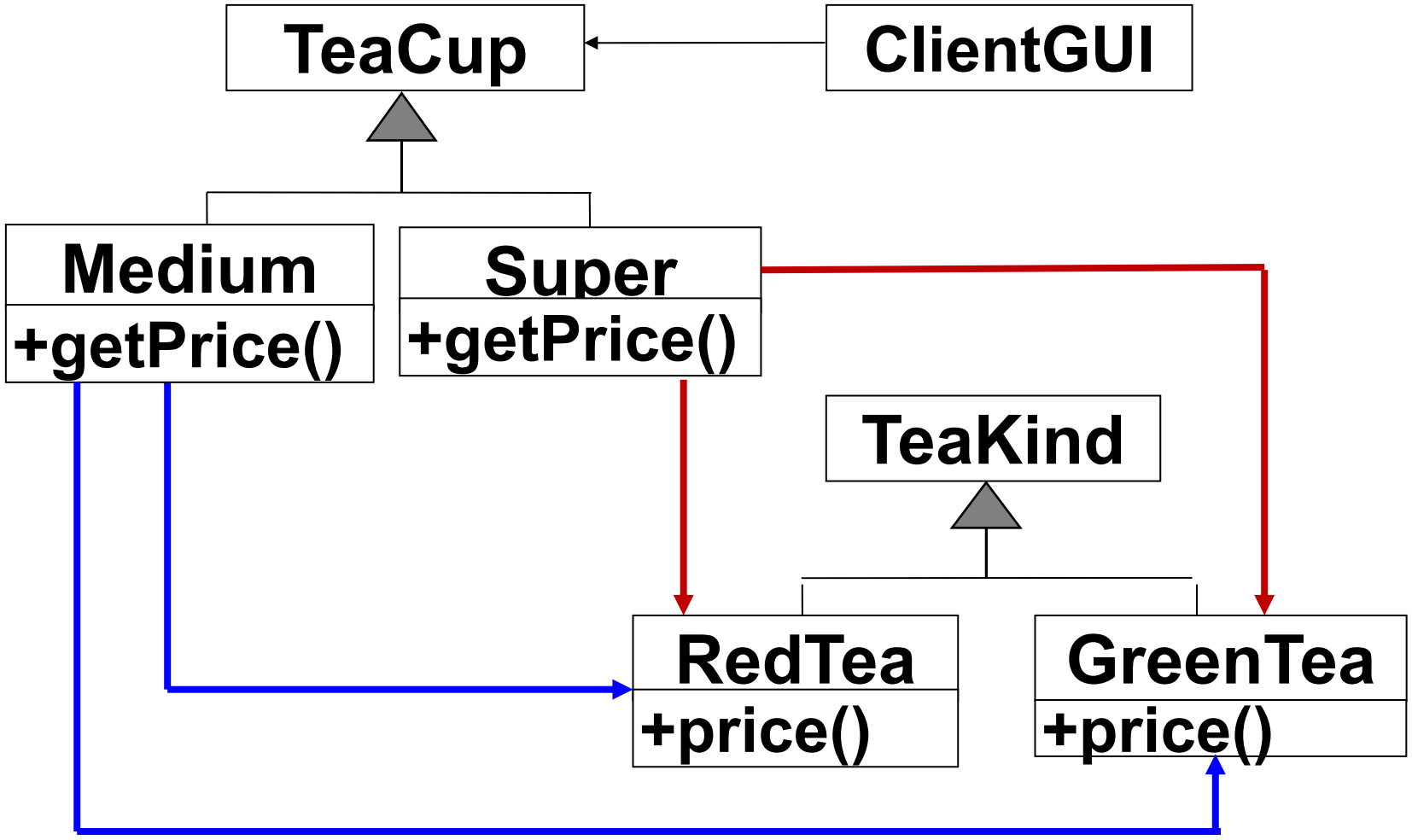
应用桥模式设计的例子

【例2】考虑一个自动茶水销售机的例子，该机器销售的茶水价格与两个维度：杯子大小与茶叶品种有关。

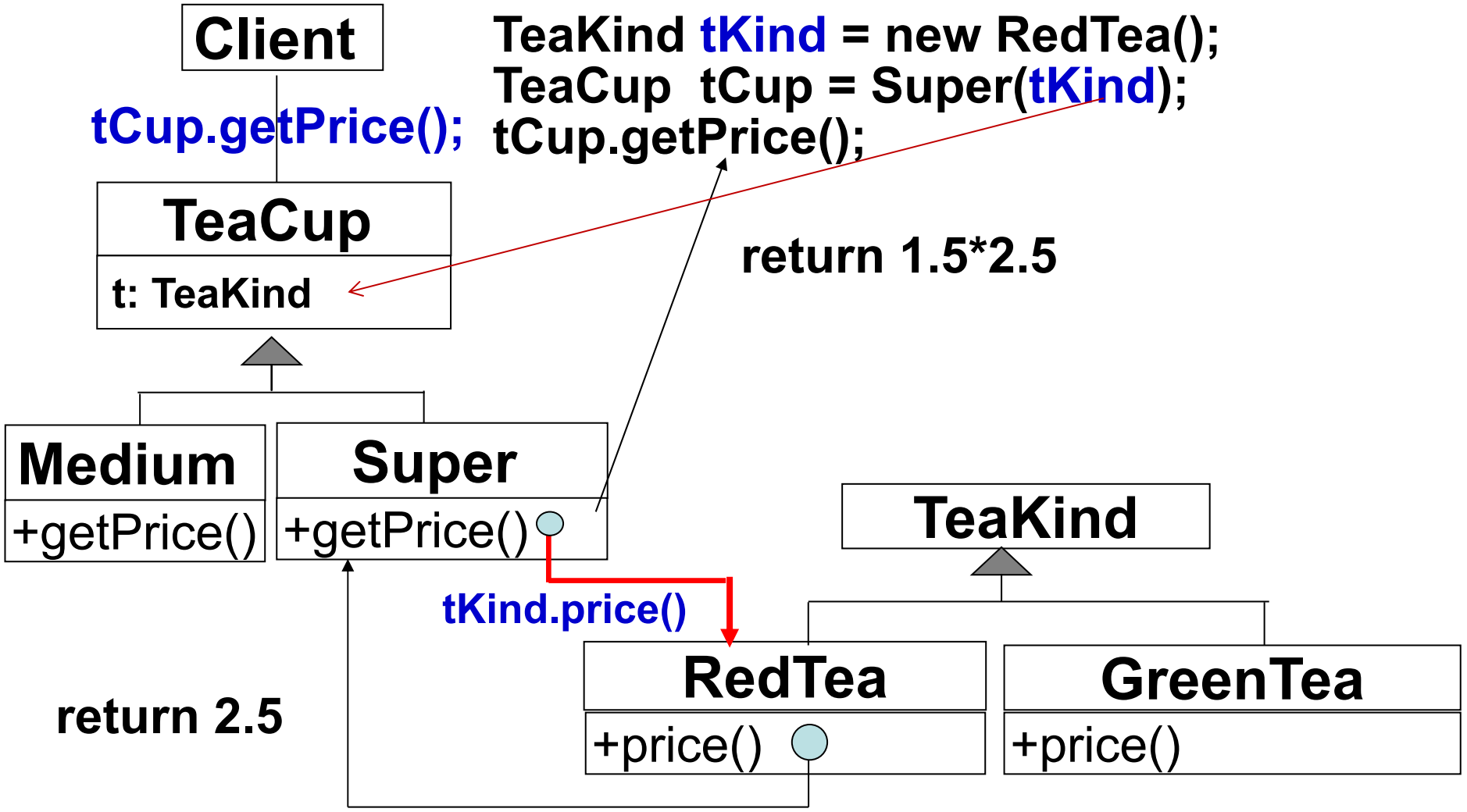
- **杯子的体积上暂时分为中杯(Medium cup)，大杯(Super Cup)；**
- **在茶叶品种上，分为红茶(Red tea)，和绿茶(Green tea)。**

应用桥模式设计的例子

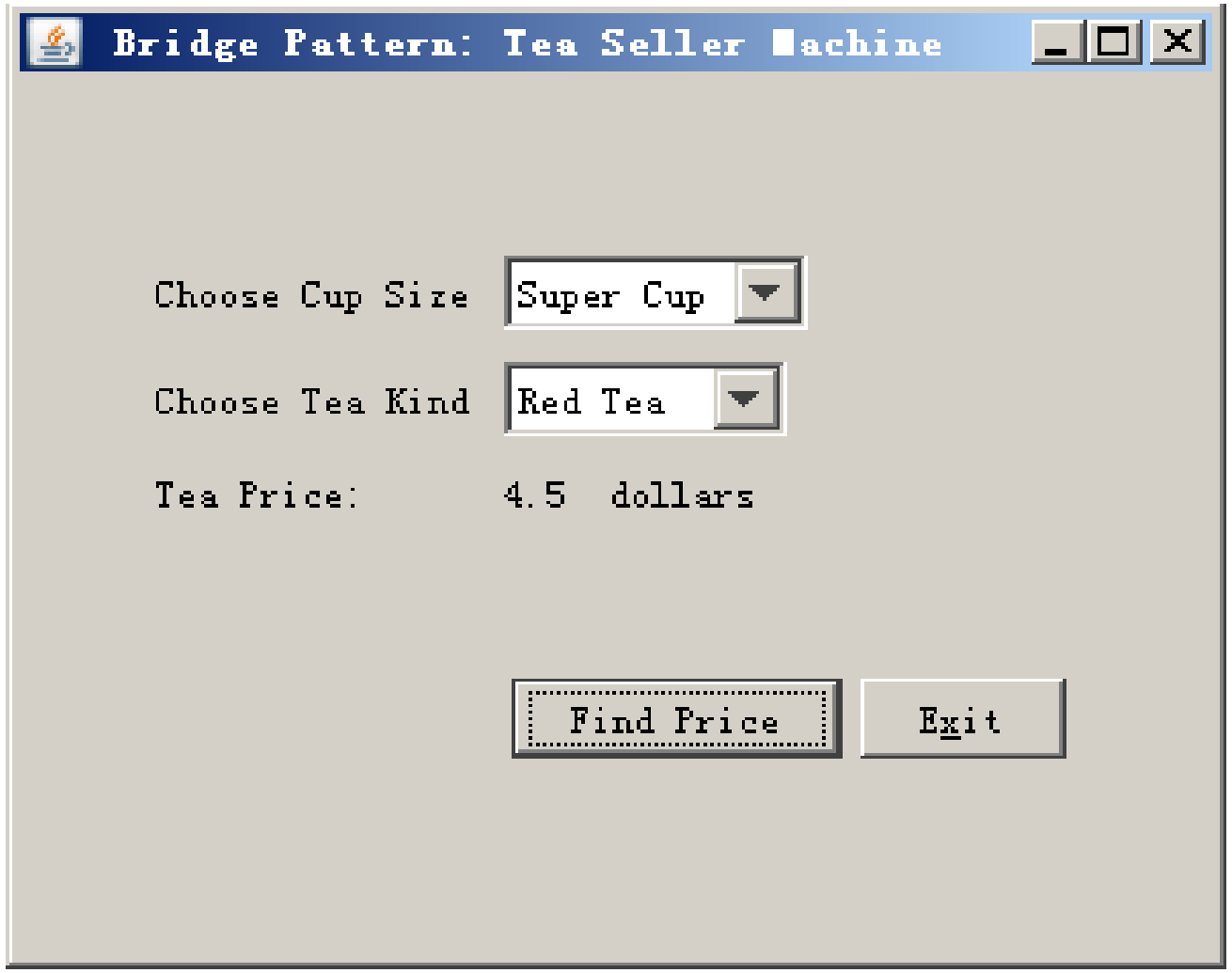
使用桥模式设计，设计类图如下。



桥接模式的典型交互



茶水程序的典型交互情况


Bridge Pattern: Tea Seller Machine
Choose Cup Size
Super Cup
Choose Tea Kind
Red Tea
Tea Price:
4.5 dollars
Find Price
Exit

应用桥模式设计的例子

```
class ButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand().equals ( FINDPRICE )) {
          String size = getTeaCup(); //获得用户输入：杯子大小
          String kind = getTeaKind(); //获得用户输入：茶种类
          //Create a TeaKind object
          if( kind.compareTo(GREENTEA)==0 )
            tKind = new GreenTea();
          if( kind.compareTo(REDTEA)==0 )
            tKind = new RedTea();
          if(size.compareTo(SUPERCUP)==0)
            tCup = new Super (tKind);
          if(size.compareTo(MEDIUMCUP)==0)
            tCup = new Medium (tKind);
          float price = tCup.getPrice();
        }
    }
}
```

创建TeaKind对象

创建TeaCup对象

应用桥模式设计的例子

抽象接口类

```
public interface TeaCup {
   public abstract float getPrice();
}
```

抽象部分子类

```
public class Medium implements TeaCup{
        private TeaKind tk;
        public Medium (TeaKind tKind){
            tk = tKind;
        }
        public float getPrice(){
           float teaPrice = tk.price();
           return teaPrice;
        }
}
```

应用桥模式设计的例子

抽象部分子类

```
public class Super implements TeaCup{
        private TeaKind tk;
        public Super (TeaKind tKind){
          tk = tKind;
        }
        public float getPrice(){
           float teaPrice = 1.5f * tk.price();
           return teaPrice;
        }
}
```

应用桥模式设计的例子

实现部分的抽象接口。

```
public interface TeaKind {
   public abstract float price();
}
```

实现部分的实现类。

```
public class RedTea implements TeaKind{
   private final float PRICE = 2.5f;
   public float price(){
      return PRICE;
   }
}
```

应用桥模式设计的例子

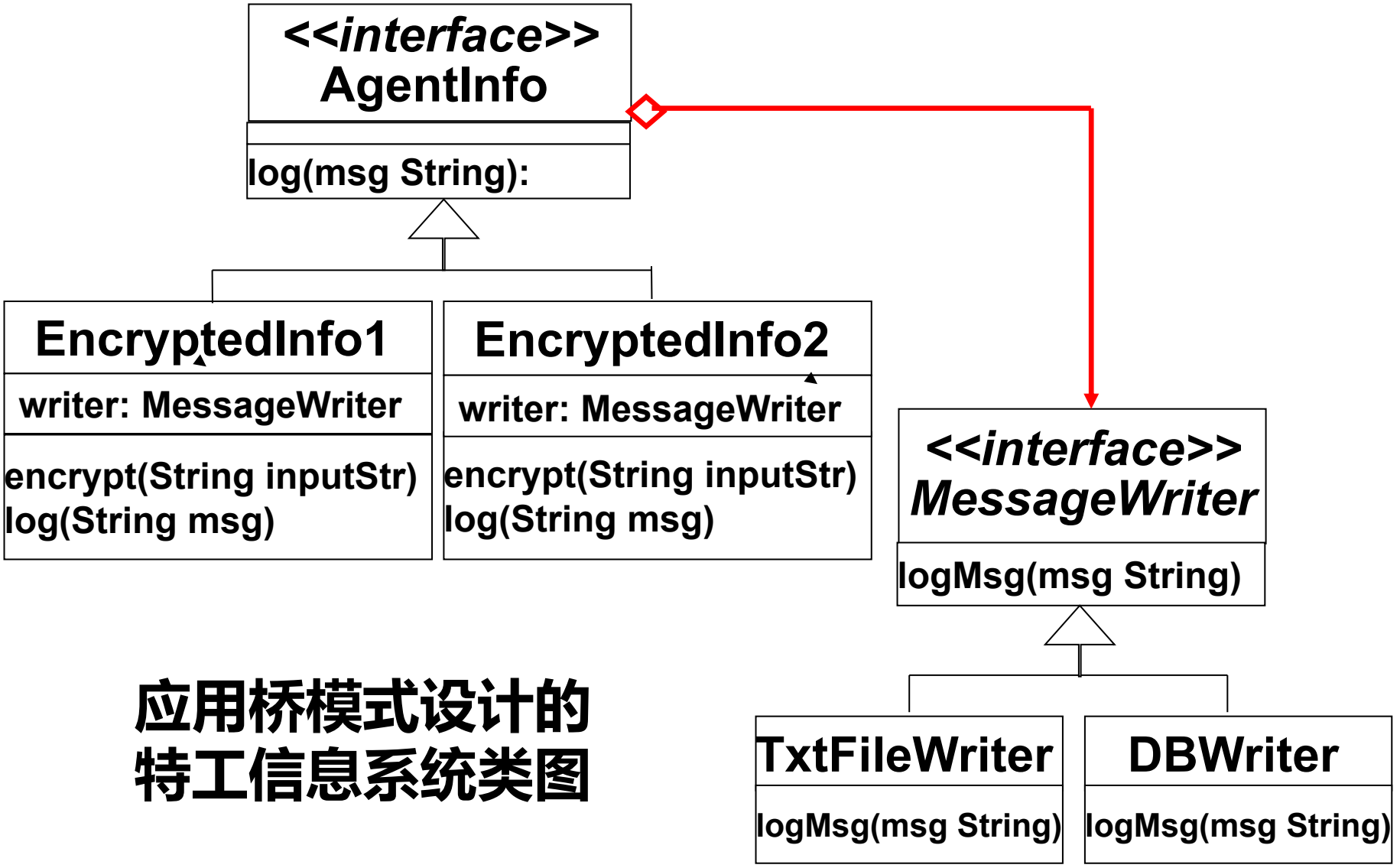
实现部分的实现类。

```
public class GreenTea implementsTeaKind{
private final float PRICE = 3.0f;
   public float price(){
      return PRICE;
   }
}
```

应用桥模式设计的例子

- **【例3】特工信息系统**。The following design represents an agent information system. The agent information is encrypted and saved into text file or a database.
- There are two ways to encrypt agent's name and code number, which are represented by classes
  - EncryptedInfo1 and
  - EncryptedInfo2.

应用桥模式设计的例子



应用桥模式设计的特工信息系统类图

EncryptedInfo1的加密算法：折叠算法

1.将26个英文字母按照顺序排列，加密的时候，以折叠的方式进行：a←→z, b←→y, ...,m←→n

2.大写字母也是如此。大写字母加密为大写字母。

3.数字加密也以此方式进行

0←→9,1←→8,2←→7,3←→6, 4←→5

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

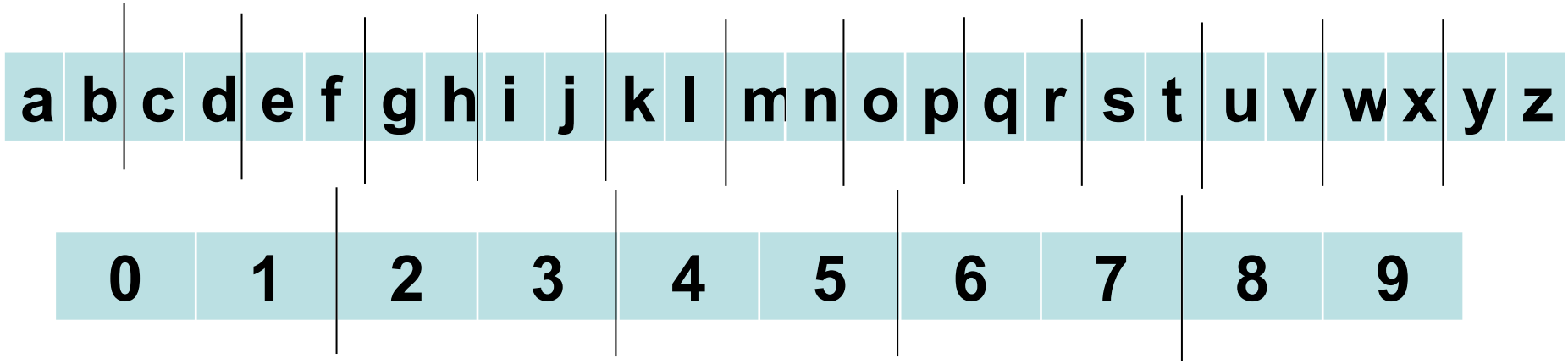
0	1	2	3	4	5	6	7	8	9

EncryptedInfo2的加密算法：**分组互换算法**

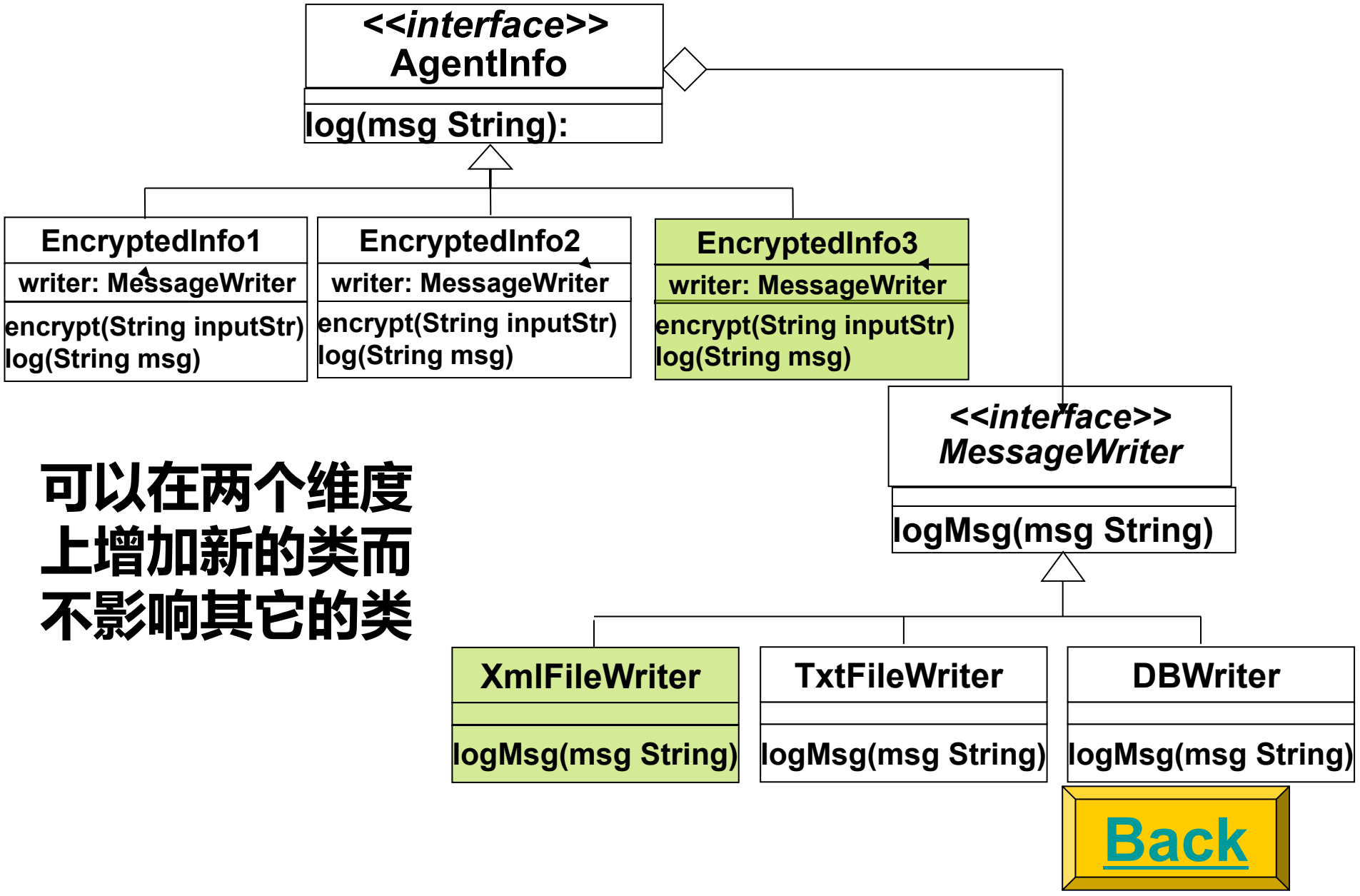
1.将26个英文字母按照顺序排列，按照如下方式两两分成一组，本组内部 加密的时候互换：a←→b, c←→d，..., w←→x, y←→z。

2.大写字母的加密方式也是如此，大写字母加密为大写字母。

3.数字加密：0←→1, 2←→3,4←→5, 6←→7, 8←→9



应用桥模式设计的例子



可以在两个维度上增加新的类而不影响其它的类

思考题

- 是否可以将在抽象工厂模式中房屋信息系统的例子改为由桥接模式设计
- 是个2维度问题
- **Category**: super, medium
- **Kind**: house, condo, semi detacher