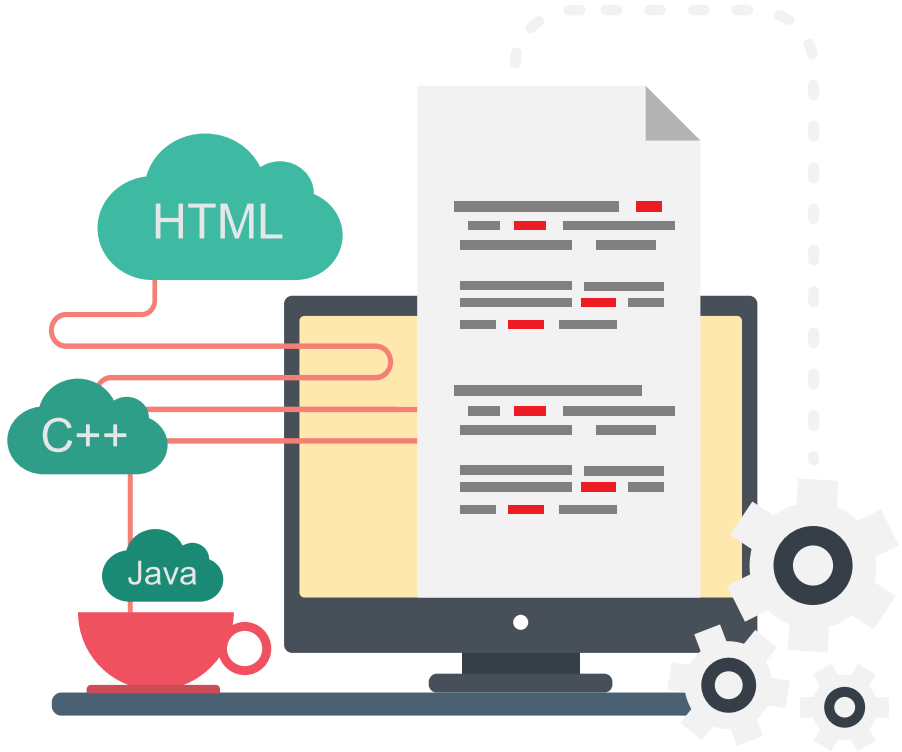
设计模式
七大设计原则

尹洪亮 Kevin

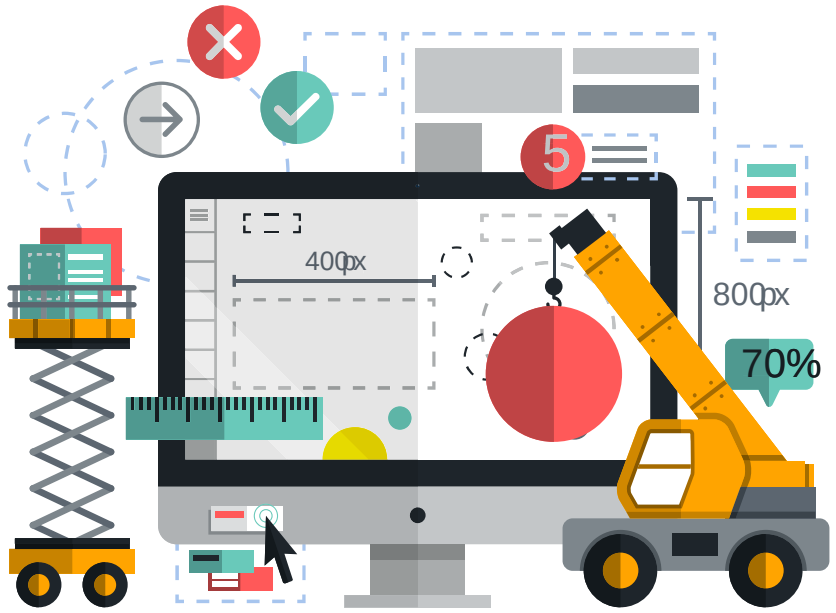
设计模式的目的



- ✓提高可重用性
- ✓提高可读性
- ✓提高可扩展性
- ✓提高可靠性
- ✓高内聚、低耦合

设计模式的设计原则

设计模式是根据七大原则进行设计的，理解了设计模式，也就理解了OOD/面向对象设计的精髓



七大设计原则

① 单一职责原则

② 接口隔离原则

③ 依赖倒置原则

④ 里式替换原则

⑤ 开闭原则

⑥ 迪米特法则

⑦ 合成复用原则

1

单一职责原则

SRP，Single responsibility principle

单一职责原则

定义

单一职责原则（SRP，Single responsibility principle）又称为单一功能原则，一个类应该只有一个发生变化的原因。

解读

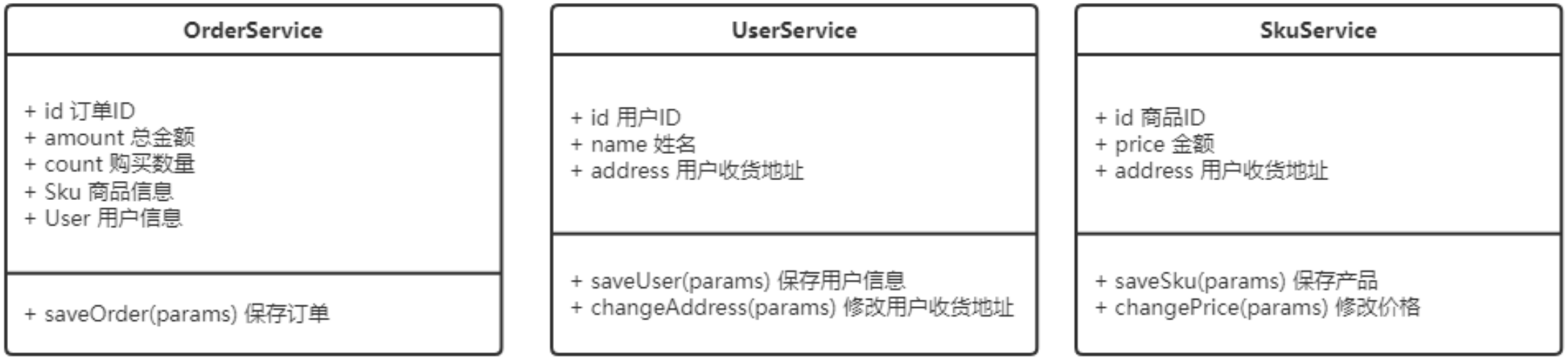
- ✓ 一个类应该只有一个职责，这个原则也可以扩展到方法、模块层级，一个方法或模块也应该只有一个原则。
- ✓ 甚至可以扩展到服务层级，一个服务应该只有一个职责（DDD中领域服务的概念、SOA/微服务中的服务概念）。
- ✓ 单一职责其实是对高内聚低耦合的扩展。
- ✓ 一个类或方法的职责过多，当修改其中一个职责的时候，就会影响到其他职责。

单一职责原则（反例）

违反了单一职责原则

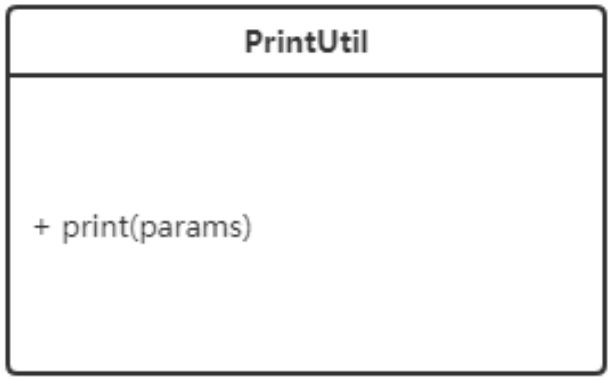
单一职责原则（正例）

按照单一职责进行拆分



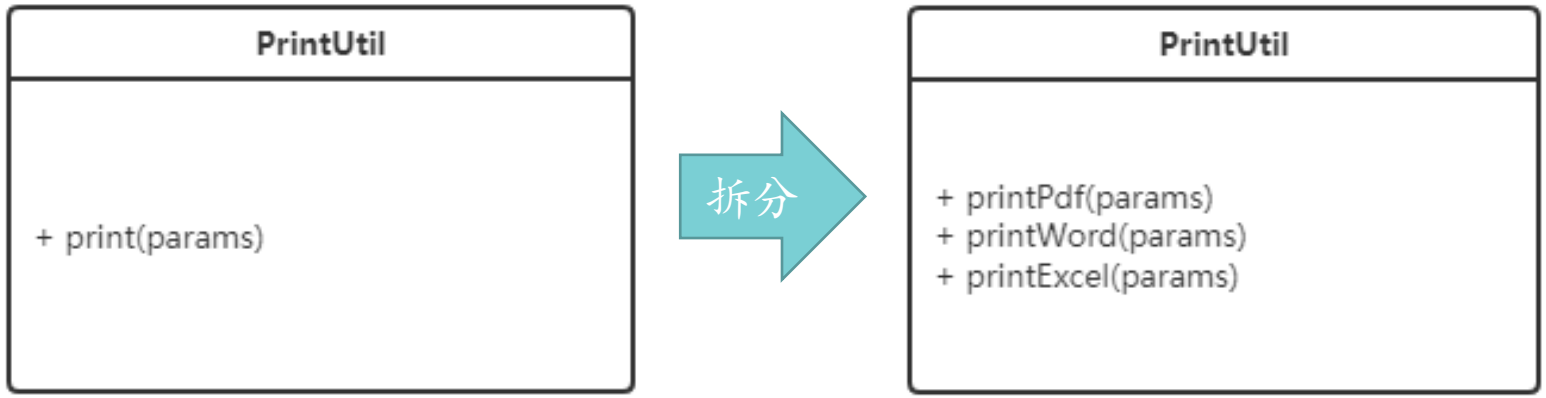
单一职责原则（反例）

违反了单一职责原则



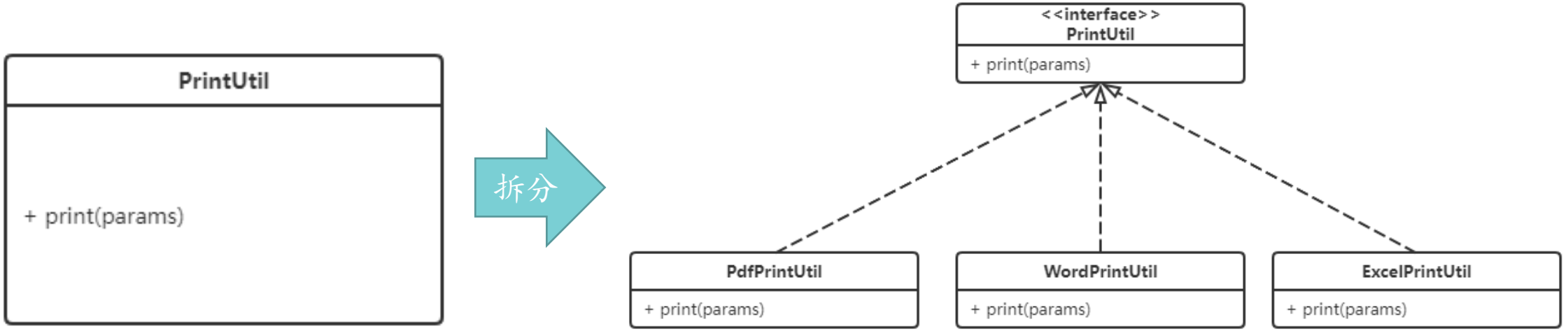
- ✓ 打印工具类，提供了print(params)打印方法，该方法可以支持PDF、Word、Excel打印等多种能力，十分强大。

单一职责原则（正例）



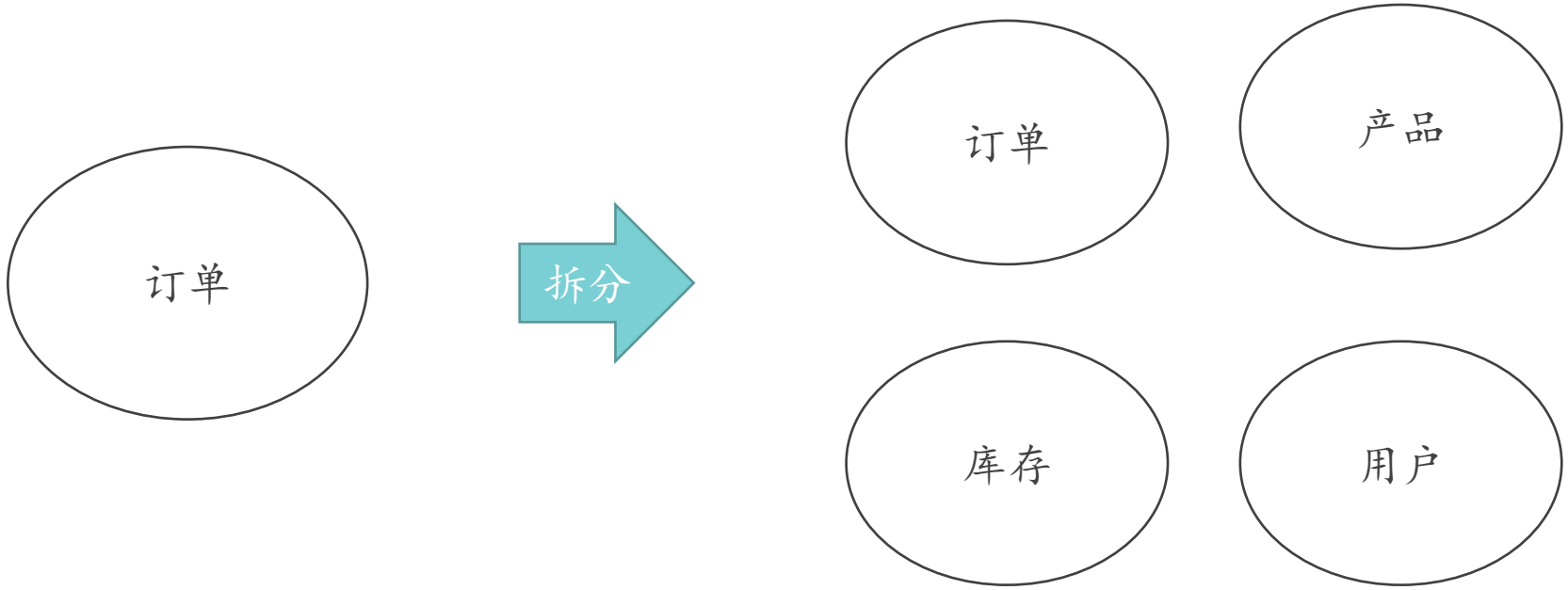
- ✓ 方案一：将print方法拆分为prnitPdf、printWord、printExcel多个方法，每个方法只负责一种文件的打印。

单一职责原则（正例）



✓ 方案二：将print方法抽象为接口，分别新建PdfPrintUtil、WordPrintUtil、ExcelPrintUtil类实现接口。

单一职责原则



✓ 完全可以扩展到模块、服务级别，同样使用单一职责原则进行服务/模块的拆分

2

接口隔离原则

ISP，Interface isolation principle

接口隔离原则

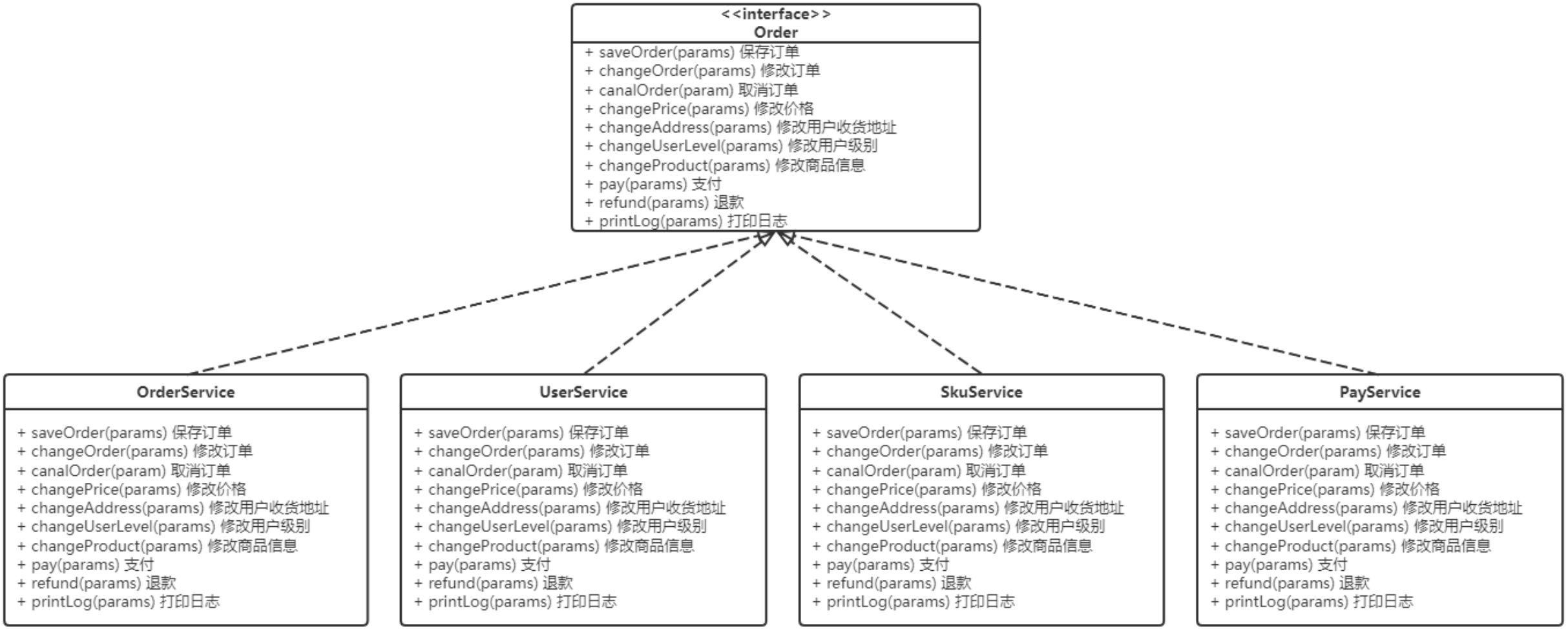
定义

接口隔离原则（ISP，Interface isolation principle）客户端不应该被迫依赖于它不使用的方法，一个类对另一个类的依赖应该建立在最小的接口上

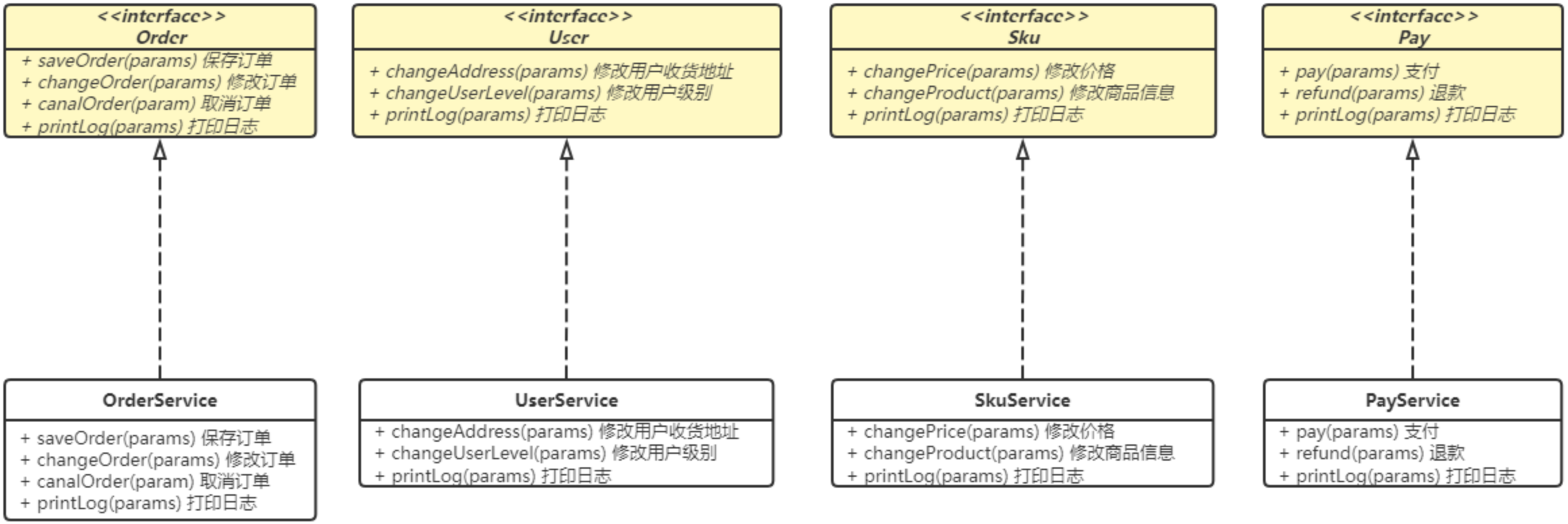
解读

- ✓ 不要试图使用一个庞大的接口包含各种方法。
- ✓ 不要让类去实现它们根本不需要实现或使用的方法。
- ✓ 接口隔离也是对高内聚低耦合的扩展，只是站在了比单一职责更高的层级视角之上，在程序框架层级上进行了约束。
- ✓ 违反接口隔离原则会导致接口膨胀，影响程序的可维护性。

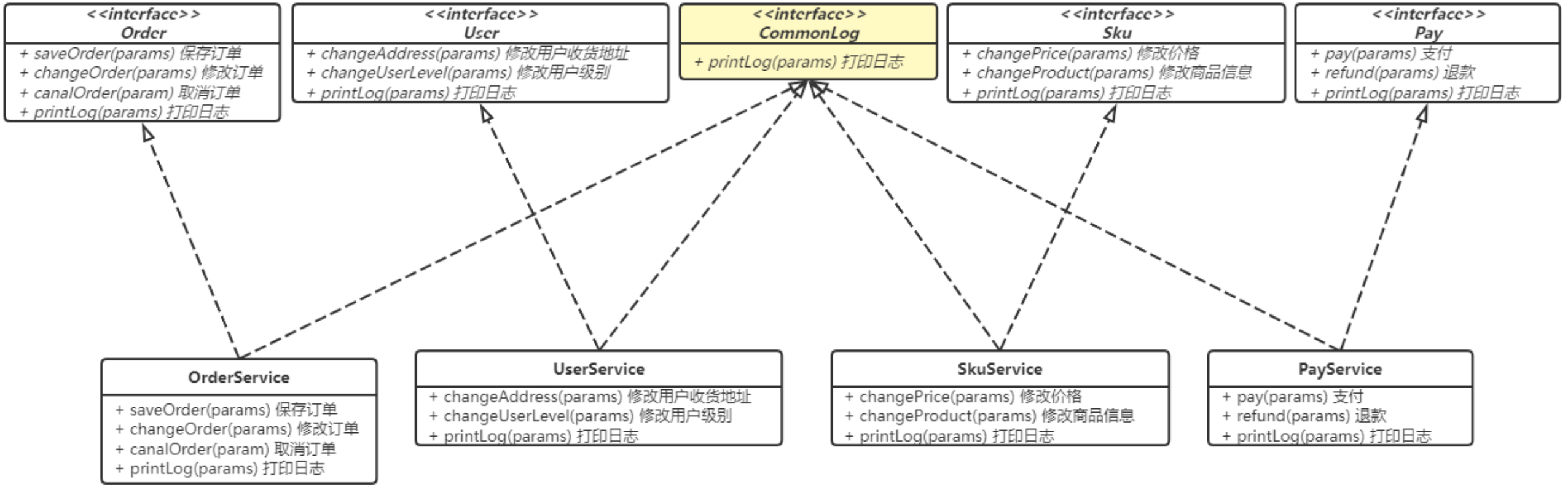
接口隔离原则（反例）



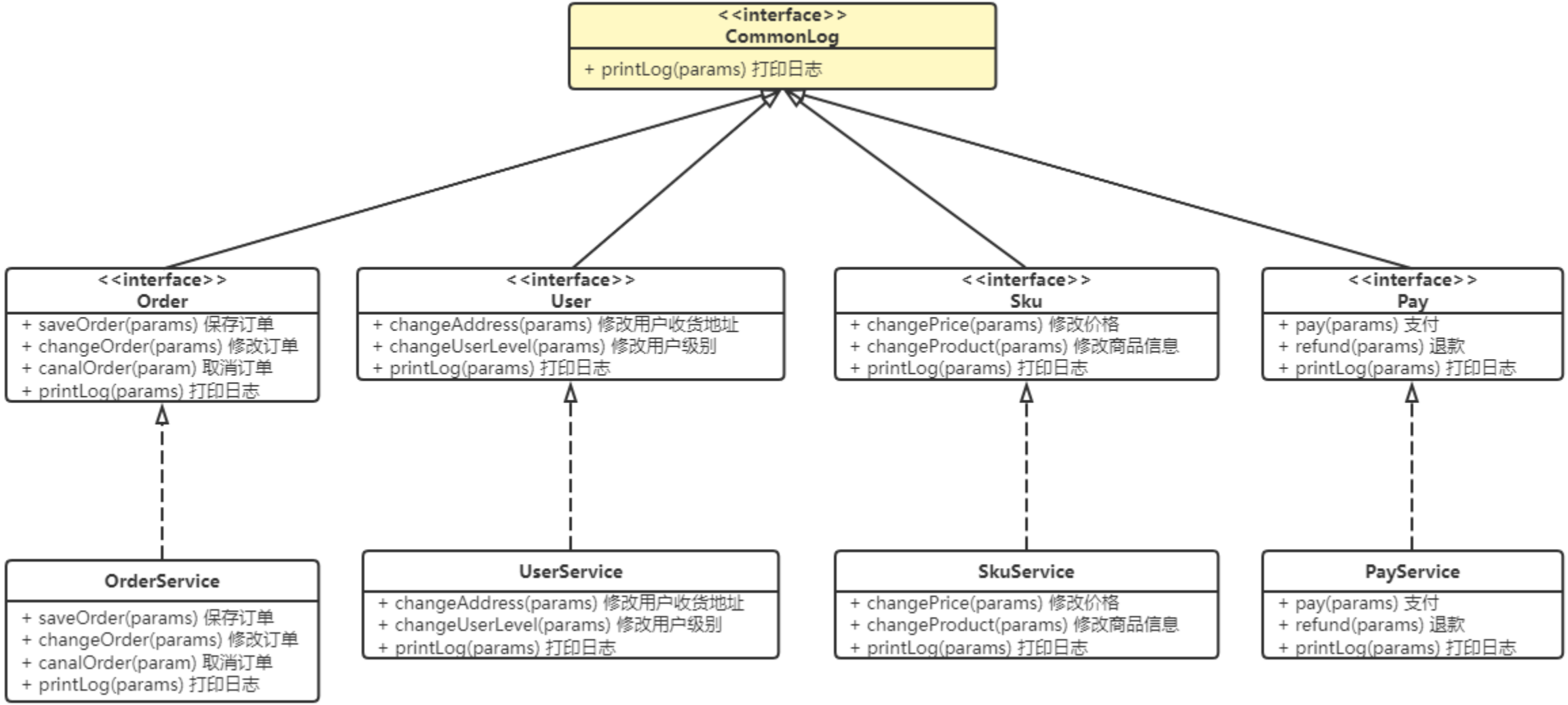
接口隔离原则（正例）



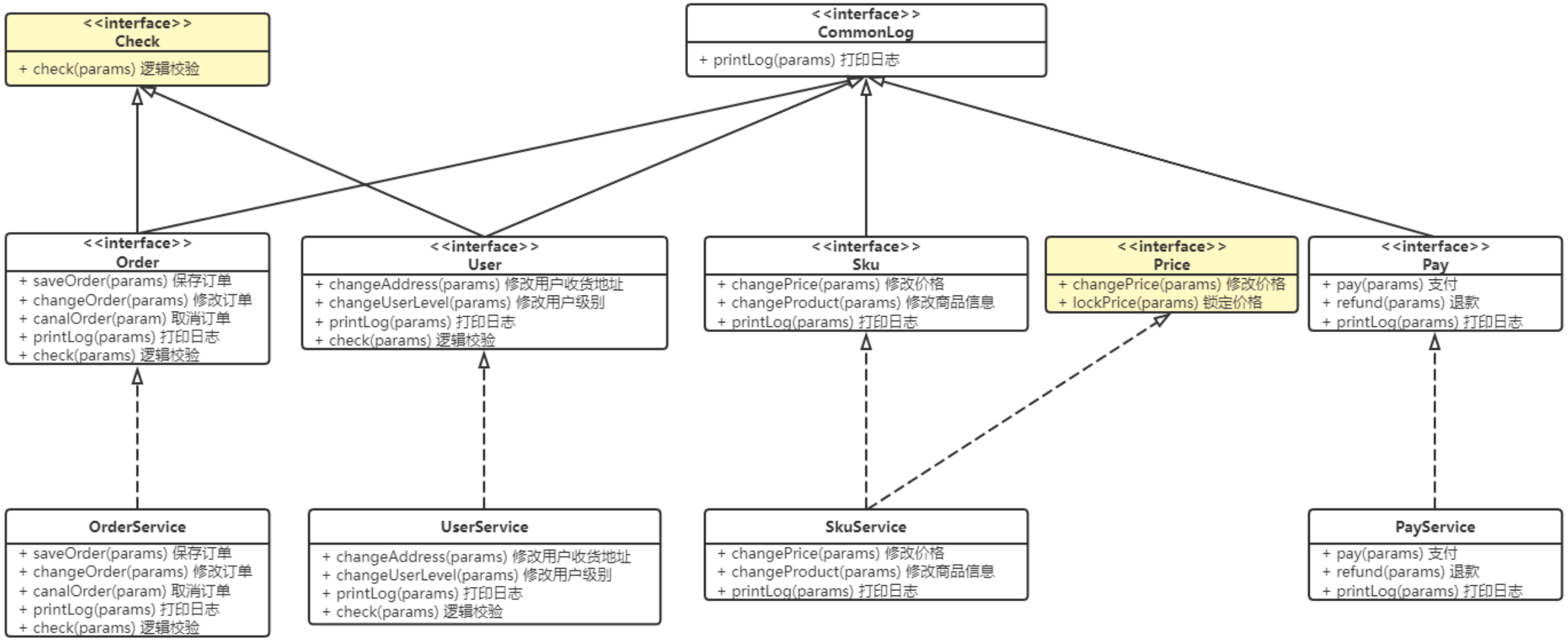
接口隔离原则（正例）



接口隔离原则（正例）



接口隔离原则（灵活运用）



接口隔离原则（总结）

① 无论采用哪种方式进行接口拆分，每个接口自身都要符合单一职责原则。

② 利用实现类实现多个接口

③ 利用利用接口继承能力

④ 提前进行接口规划

3

依赖倒置原则

DIP，Dependence Inversion Principle

依赖倒置原则

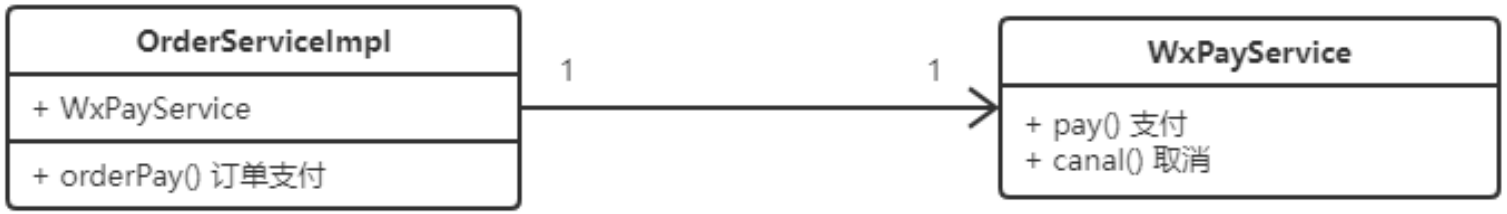
定义

依赖倒置原则（DIP，Dependence Inversion Principle）高层模块不应该依赖底层模块，两者都应该依赖抽象

解读

- ✓ 核心思想就是面向接口编程
- ✓ 程序要依赖于抽象接口，不要依赖于具体实现。
- ✓ 面向实现编程，会导致程序耦合度提高，维护性变差。
- ✓ 依赖倒置原则是JAVA多态特性的完美呈现

依赖倒置原则（反例）

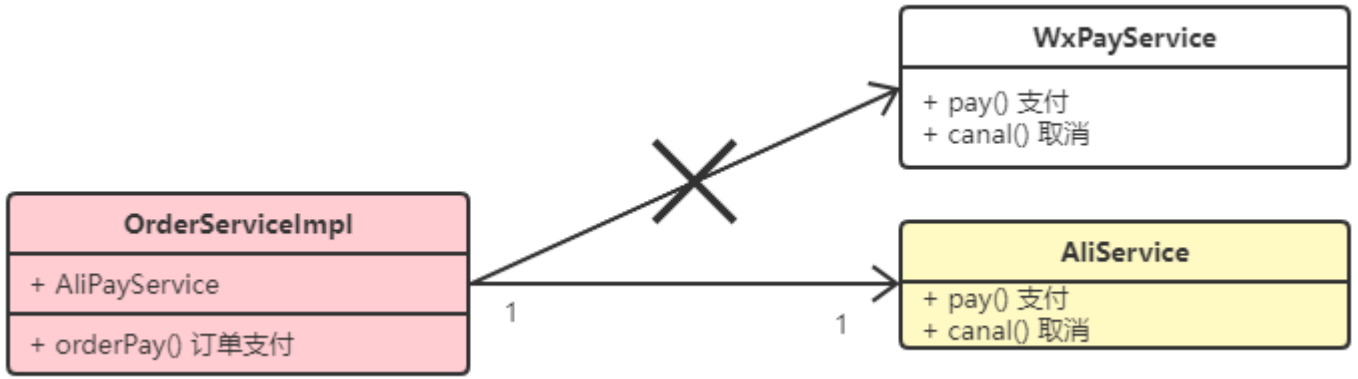



```
public class OrderServiceImpl {

    public WxPayServce wxPayServce;

    public void orderPay(){
        wxPayServce.pay();
    }

}
```

```
class WxPayServce{

    //微信支付
        public void pay(){
         //微信支付
        }

    //微信支取消
        public void canal(){
         //微信支付取消
        }

}
```

依赖倒置原则（反例）

✓ 不使用微信支付了，改为支付宝支付，怎么办？



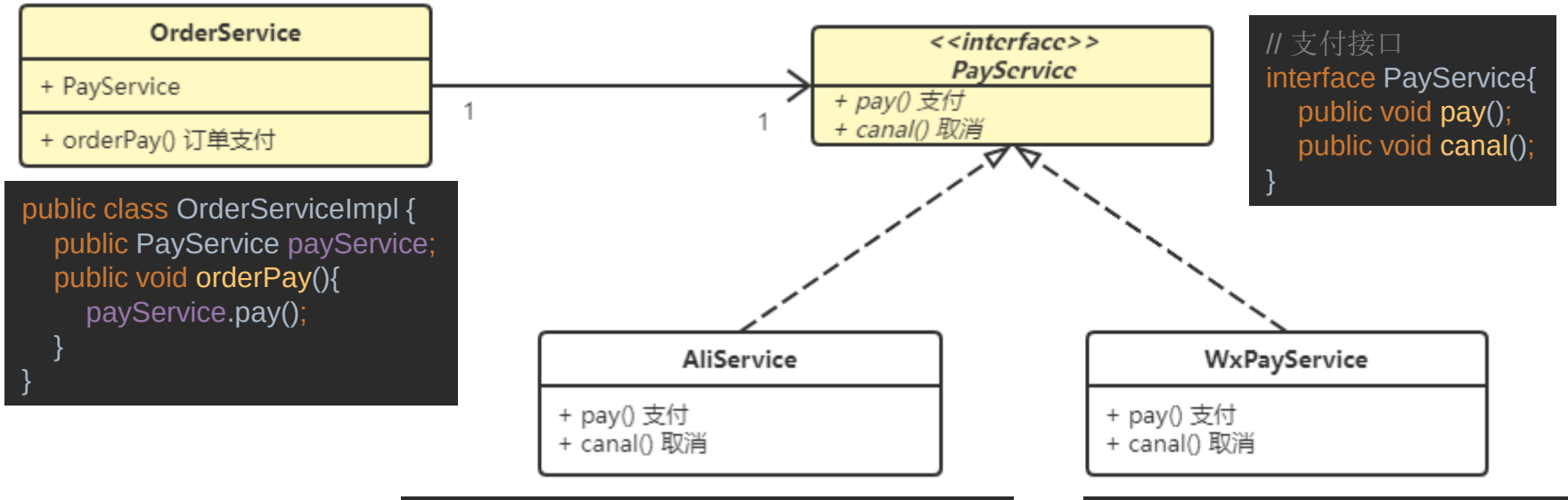
代码需大量修改

```
public class OrderServiceImpl {
    public AliPayServce aliPayServce;
    public void orderPay(){
        aliPayServce.pay();
    }
}
```

```
//支付宝支付实现
class AliPayServce{
    public void pay(){}
    public void canal(){}
}
```

依赖倒置原则（正例）

✓ 面向接口编程，依赖于上层接口/抽象，不再依赖实现类




```
public class OrderServiceImpl {
    public PayService payService;
    public void orderPay(){
        payService.pay();
    }
}
```

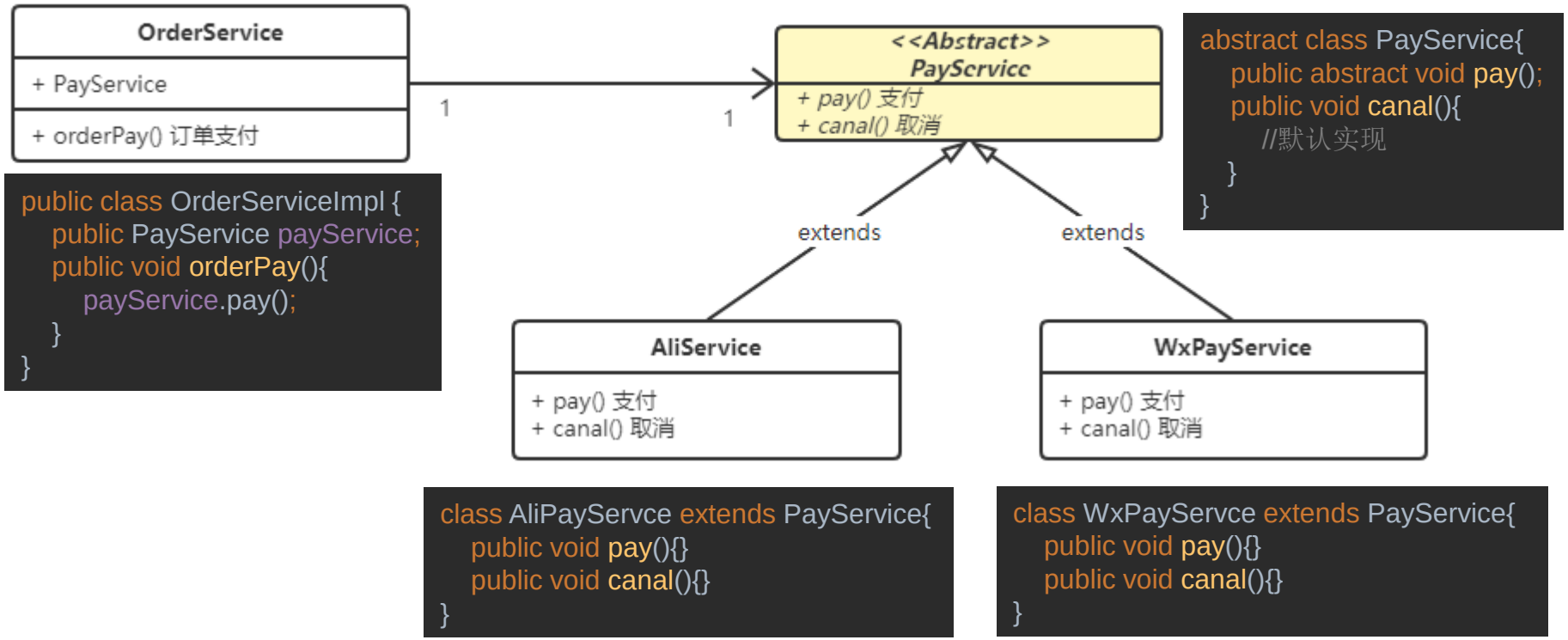
```
// 支付接口
interface PayService{
    public void pay();
    public void canal();
}
```

```
//支付宝支付实现
class AliPayServce implements PayService{
    public void pay(){}
    public void canal(){}
}
```

```
//微信支付实现
class WxPayServce implements PayService{
    public void pay(){
    }
    public void canal(){
    }
}
```

依赖倒置原则（正例）

✓ 同理，使用抽象类也能够达到相同的效果

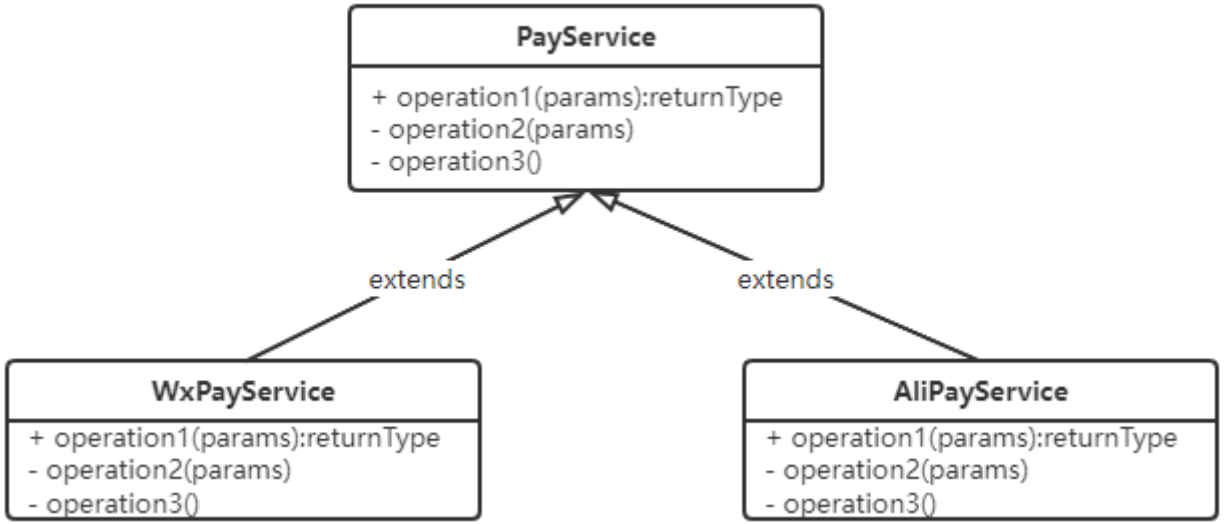


4

里式替换原则

LSP，Liskov Substitution principle

为什么要使用里式替换原则



- ✓ 继承使得类之间的耦合程度升高，当要修改基类的时候，必须要考虑所有子类的修改。
- ✓ 基类发生变化时，可能会引起所有的基类功能不正常。
- ✓ **如何正确的使用“继承”，就要考虑“里式替换原则”**

里式替换原则

定义

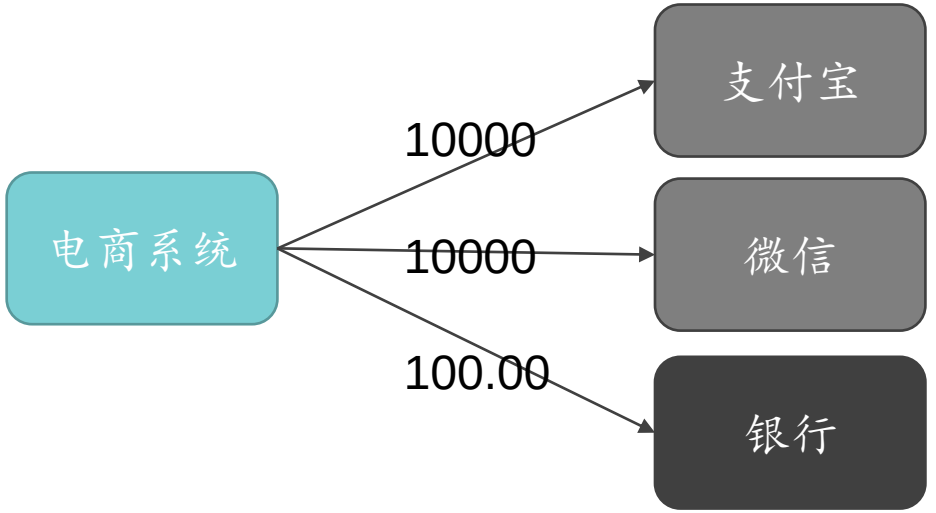
里式替换原则（LSP，Liskov Substitution principle）派生类（子类）对象可以在程序中代替其基类（超类）对象。

解读

- ✓ 里式替换原则是对继承的思考，提出相应的约束。
- ✓ 子类可以扩展父类的功能，但不能改变父类的原有功能。
- ✓ 在程序中使用子类替换其父类，程序应该不产生任何错误，而反过来则不成立。
- ✓ 程序中应该尽量使用基类类型来进行定义，而运行时再替换为具体的子类。
- ✓ 子类可以实现父类中的抽象方法，而不要覆盖父类的非抽象方法。
- ✓ 子类中可以添加自己特有的方法，但是尽量不要重写父类的方法。
- ✓ **总结：“子类只扩展父类，而不修改父类”**

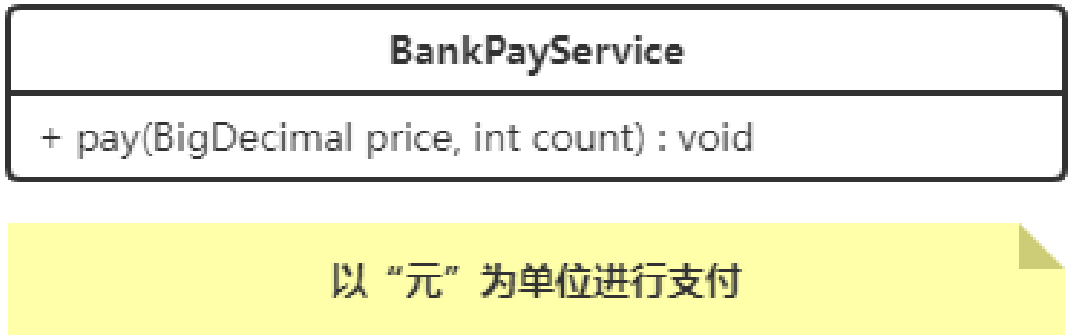
里式替换原则（用户故事）

- ✓ 某电商系统的支付功能。
- ✓ 需要支持微信、支付宝、银企直连三种付款方式。
- ✓ 必须要先计算需要支付的总金额（总金额=单价*数量）
- ✓ 其中微信和支付宝需要以“分”为单位支付，银行需要以“元”为单位进行支付。
  - ■ 例如支付100元，微信和支付宝系统需要接收金额参数为“10000”分，而银行系统需要“100.00”元。



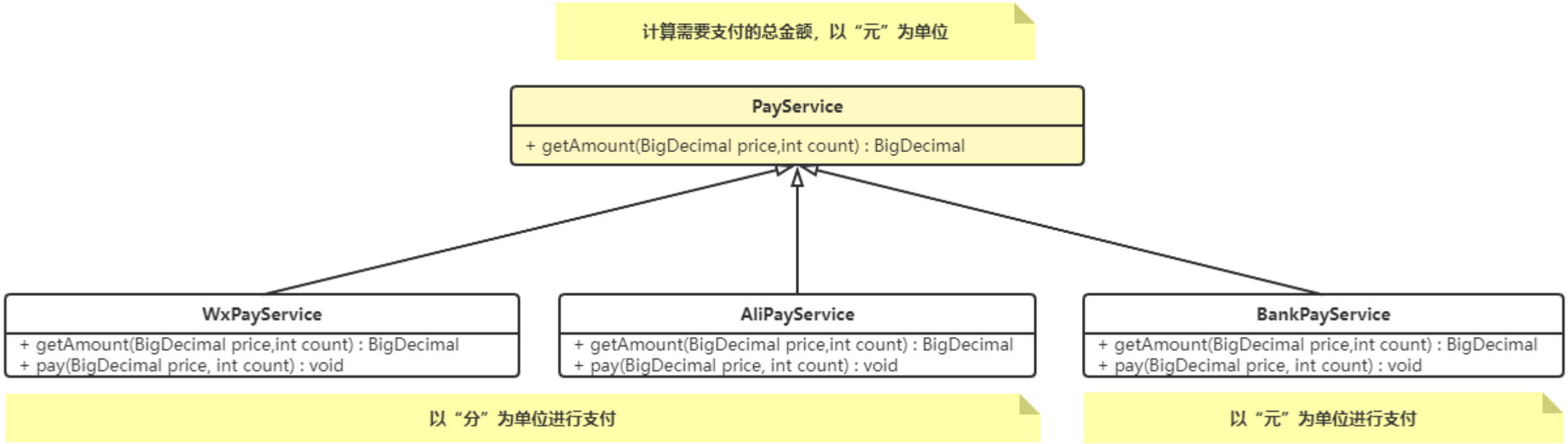
里式替换原则（反例&反例）

第1年：只有银企支付



里式替换原则（反例&反例）

第2年：添加微信、支付宝支付



5

开闭原则

OCP， Open Close Principle

开闭原则

定义

开闭原则（**OCP，Open Close Principle**）软件中的对象（类、方法、模块）等应该对于扩展开放，而对修改封闭。

解读

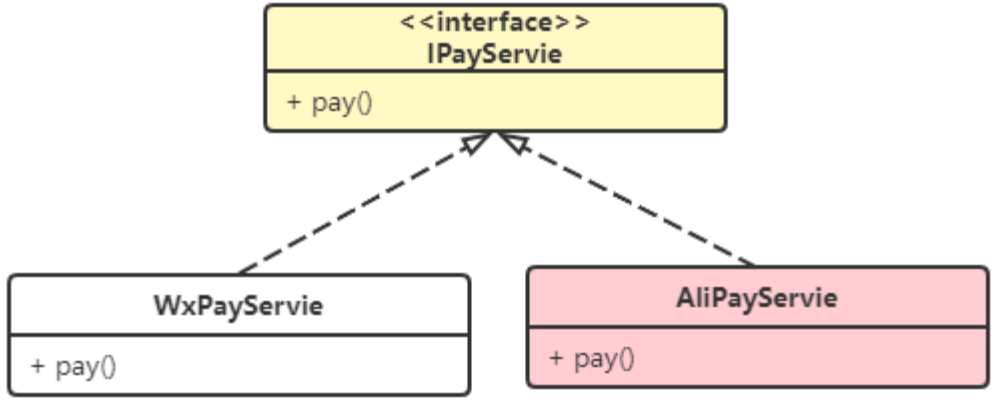
- ✓ 应该在不修改源码的前提下修改或扩展其行为。
- ✓ 只有在存在错误的情况下才会修改一个类的代码。
- ✓ 新增或修改程序的功能，应该通过新建不同的实现类来实现。

开闭原则（示例）

起初：只有一种支付方式

多种支付方式，用if来判断处理

多种支付方式，用方法做区分



✓ 开闭原则，新增支付方式，则新建实现类

6

迪米特法则

LOD， Law of Demeter

迪米特法则

定义

迪米特法则（Law of Demeter）又叫作最少知识原则（The Least Knowledge Principle），一个类对于其他类知道的越少越好，只和朋友通信，不和陌生人说话。

解读

- ✓ 对自己依赖的类知道越少越好，不要管其内部多么复杂，都要将逻辑封装在其内部，对外只提供public方法。
- ✓ 只和朋友通讯，是指只与具有耦合关系的类通讯，包括依赖、关联、组合、聚合关系等。
  - ✓ 出现在成员变量、方法参数、返回值中的类为朋友。
  - ✓ 出现在方法中的局部变量，尽量不要使用不了解的类。
- ✓ 关注的是封装特性、内聚特性。

7

合成复用原则

CARP，Composition/Aggregate Reuse Principle,

合成复用原则

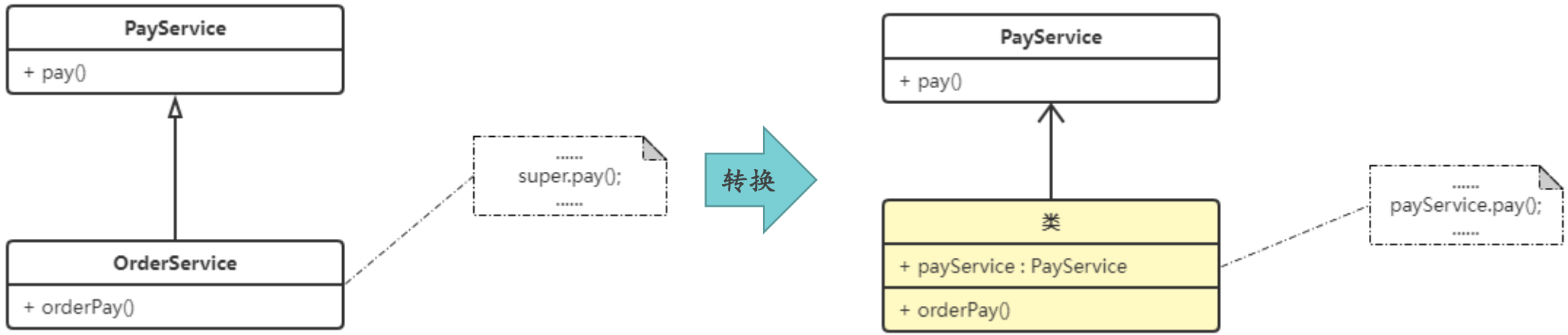
定义

合成复用原则又称为组合/聚合复用原则(Composition/Aggregate Reuse Principle, CARP)，尽量使用对象组合，而不是继承来达到复用的目的。

解读

✓ 尽量不使用继承，而使用聚合

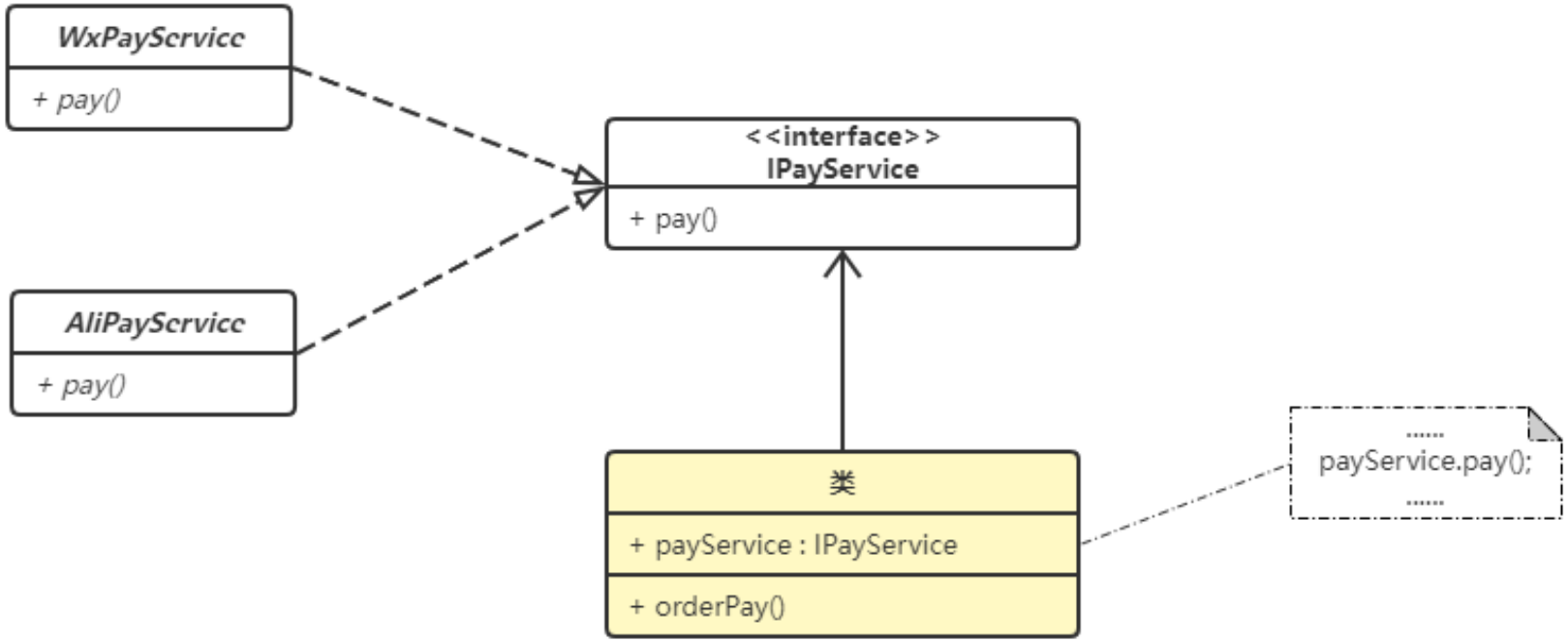
合成复用原则（示例）



- 继承关系

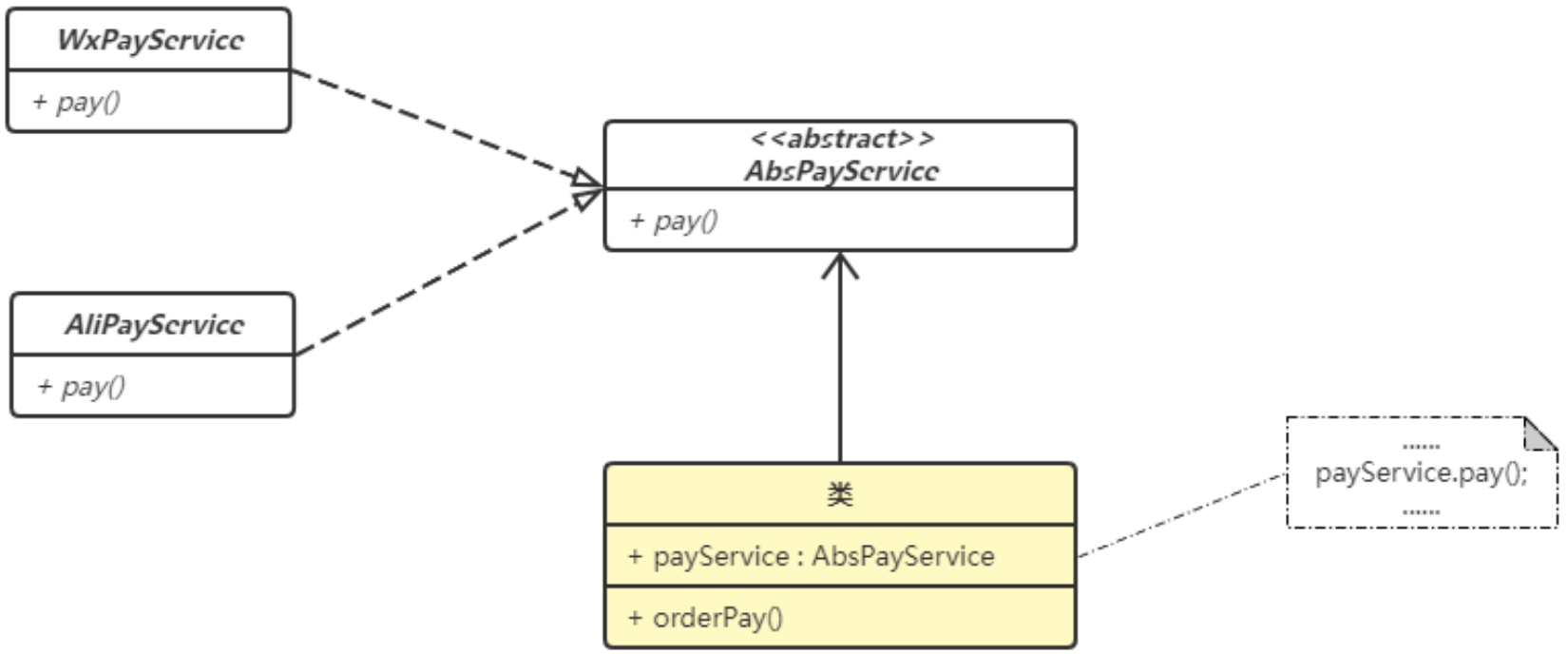
✓ 聚合关系

合成复用原则（示例）



✓ 合成复用原则，再结合依赖倒置原则(利用接口）

合成复用原则（示例）



✓ 合成复用原则，再结合依赖倒置原则（利用抽象类）

8

总结

Summary

联系作者，加我为好友

总结

- ✓ 忘掉所有的原则，不要死记硬背每个原则的名称和定义，牢记以下3点即可。
- ✓ 坚持“高内聚、低耦合”
- ✓ 坚持“面向接口编程”
- ✓ 坚持“多用聚合，少用继承”

Thank you for watching.