**第九章 适配器模式**

**一、结构性设计模式**

GoF设计模式中包含了7种结构性设计模式，分别是：适配器模式、桥接模式、组合模式、装饰模式、外观模式、享元模式、代理模式。

结构性设计模式关注如何将现有的类或对象组织在一起形成更加强大的结构。不同的结构性设计模式从不同的角度来组合类或对象，它们在尽可能满足各种面向对象设计原则的同时为类或对象的组合提供一系列巧妙的解决方案

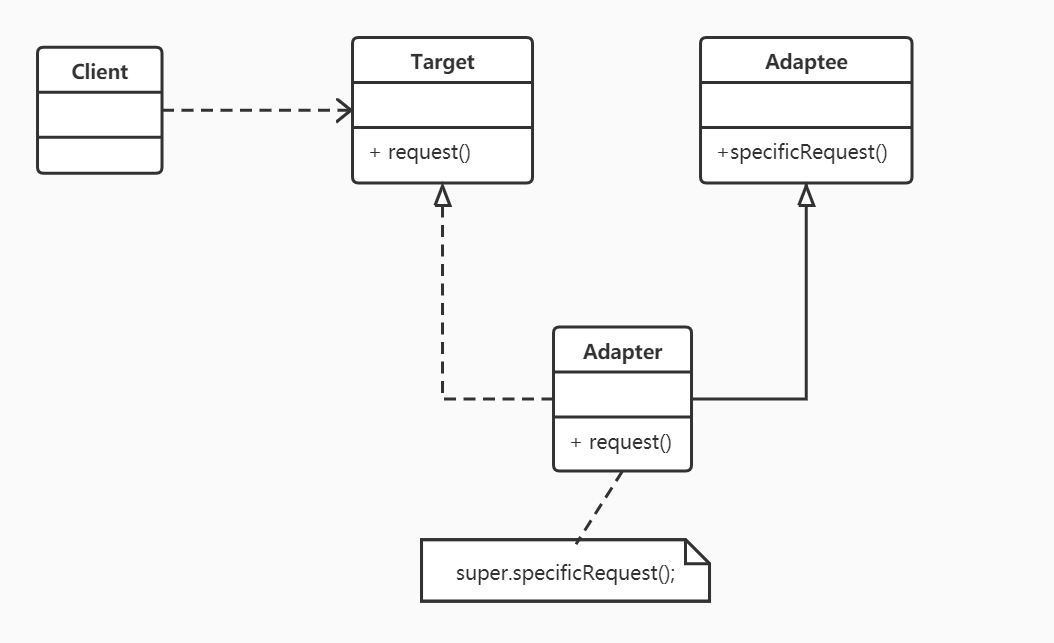
**二、适配器模式概述**

当系统存在不兼容的接口，需要引入一个适配器来使原本因为接口不兼容而不能一起工作的两个类能够协同工作（可以联想一下笔记本、手机等的充电器，也就是电源适配器，有了这个适配器，原本不能工作的生活用电和笔记本电脑就可以兼容了）。

适配器模式就是引入了一个叫适配器（Adapter）的包装类，它所包装的对象称为适配者（Adaptee）,即被适配的类。当客户类调用适配器的方法时在适配器类的内部将调用适配者类的方法，客户类并不直接访问适配者类。

**三、适配器模式结构与实现**

**类适配器模式**



（1）Target（目标抽象类）：定义客户所需的接口，在类适配器模式中，它只能是接口类，因为Java语言不支持多重继承，而Adapter已经继承了Adaptee类，不能再继承另一个类，所以Target只能是接口。

（2）Adapter（适配器类）：它可以调用另一个接口，作为一个转换器，对Adaptee和Target进行适配。适配器Adapter是适配器模式的核心，类适配模式中，Adapter通过实现Target接口并继承Adaptee类来使二者产生联系。

典型代码如下：

•Java中可以同时使用**implements**和**extends**关键字，但必须**先写extends，后写implements。**

•Java中的**super**关键字是一个引用变量，每当创建子类的实例时父类的实例被隐式创建，由**super**关键字引用变量引用。

**super**可以用来调用父类方法。但如果子类包含与父类相同的方法，就应用**super**关键字来指定父类方法。

**public** **class** Adapter **extends** Adaptee **implements** Target{

**public** **void** request() {

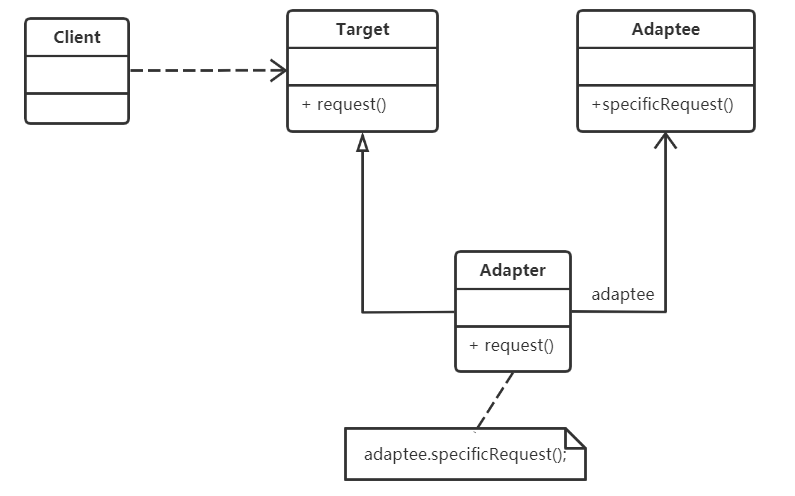
**super**.specificRequest();

}

}

（3）Adaptee（适配者类）：适配者即被适配的角色，它定义了一个已经存在的接口，这个接口需要适配，适配者类一般是一个具体类，包含了客户希望使用的业务方法。

**对象适配器模式**

****

（1）Target（目标抽象类）：定义客户所需的接口，在对象适配器模式中，可以是一个抽象类或接口，也可以是具体类。

（2）Adpter（适配器类）：在对象适配模式中，Adapter通过继承Target并关联一个Adaptee对象使二者产生联系

典型代码如下：

**public** **class** Adapter **extends** Target{

**private** Adaptee adaptee;//维持一个对适配者对象的引用

**public** Adapter(Adaptee adaptee) {

**this**.adaptee = adaptee;

}

**public** **void** request() {

adaptee.specificRequest();//转发调用

}

}

（3）Adaptee（适配者类）：适配者即被适配的角色，它定义了一个已经存在的接口，这个接口需要适配，适配者类一般是一个具体类，包含了客户希望使用的业务方法。

**四、适配器模式优缺点及适用环境**

**优点：**

（1）让两个没有关联的类一起运行

（2）增加了类的透明性和复用性。具体的业务实现过程放在了适配者类中，对于客户端来说是透明的；同时同一个适配者类可以在多个不同的系统中复用，提高了适配者的复用性

（3）灵活性和扩展性都非常好，通过使用配置文件可以很方便地更换适配器，也可以在不修改原有代码的基础上增加新的适配器类，完全符合开闭原则。

**缺点：**

类适配器模式：

（1）Java语言不支持多重类继承，一次最多只能适配一个适配者类，不能同时适配多个适配者。

（2）类适配器模式中的目标抽象类只能为接口，不能为类，其使用有一定的局限性。

**适用环境：**

（1）系统需要使用一些现有的类，而这些类的接口（例如方法名）不符合系统的需要，甚至没有这些类的源代码

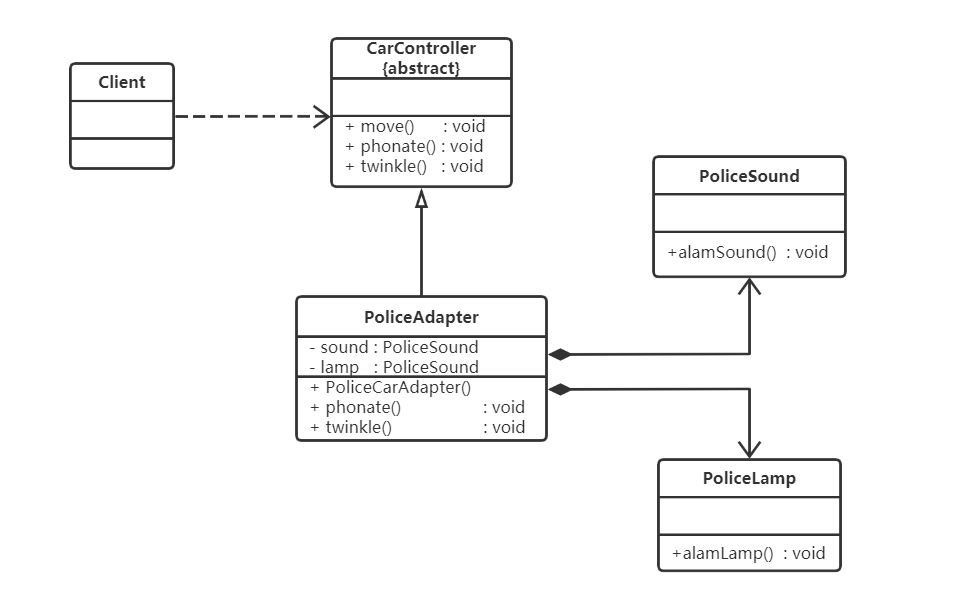
（2）想创建一个可以重复使用的类，用于和一些彼此之间没有太大联系的类（包括一些可能在将来引进的类）一起工作。

**五、适配器模式应用实例**

1.实例说明

某公司要开发一款儿童玩具汽车，为了更好地吸引小朋友的注意力，该玩具汽车在移动过程中伴随着灯光闪烁和声音提示。在该公司以往的产品中已经实现了控制灯光（例如警灯闪烁）和声音提示（例如警笛音效）的程序，为了重用先前的代码并且使汽车控制软件具有更好的灵活性和扩展性，现使用适配器模式设计该玩具汽车控制软件。

2.实例类图



本实例中目标抽象类是一个抽象类，而不是接口，并且实例中的适配器类PoliceCarAdapter同时适配了两个适配者，由于Java语言不支持多重类继承，所以只能通过**对象适配器模式**实现，CarController类充当抽象目标，Policesound和PoliceLamp类充当适配者，PolceCarAdapter充当适配器。

3.实例代码

（1）CarController：汽车控制类，充当目标抽象类(Target)。

**public** **abstract** **class** CarController{

**public** **void** move() {

System.***out***.println("玩具汽车移动！");

}

**public** **abstract** **void** phonate();//抽象方法，要在子类PolceCarAdapter

**public** **abstract** **void** twinkle(); 中实现

}

(2)PoliceSound：警笛类，充当适配者（Adaptee）。

**public** **class** PoliceSound{

**public** **void** alarmSound() {

System.***out***.println("发出警笛声音！");

}

}

（3）PoliceLamp：警灯类，充当适配者（Adaptee）。

**public** **class** PoliceLamp{

**public** **void** alarmLamp() {

System.***out***.println("呈现警灯闪烁！");

}

}

（4）PoliceCarAdapter：警车适配器，充当适配器（Adapter）。

**public** **class** PolicCarAdapter **extends** CarController{

**private** PoloceSound sound;//定义适配者PoliceSound对象

**private** PoliceLamp lamp; //定义适配者PoliceLamp对象

**public** PoliceCarAdapter() {

sound = **new** PoliceSound();

lamp = **new** PoliceLamp();

}

//发出警笛声音

**public** **void** phonate() {

sound.alarmSound();//通过关联的对象调用适配者类PoliceSound的方法

}

//呈现警灯闪烁

**public** **void** twinkle() {

lamp.alarmLamp();//通过关联的对象调用适配者类PoliceLamp的方法

}

}

（6）Client：客户端测试类

**public** **class** Client{

**public** **static** **void** main(String arg[]) {

CarController car;

car = (CarController)XMLUtil.getBean();

car.move();//move的调用是直接在CarController中实现的方法

car.phonate();//phonate的调用是通过PoliceCarAdapter中关联的对象来调用 适配者类（PoliceSound和PoliceLamp）中的函数.

car.twinkle();//twinkle的调用同phonate

}

}

**六、课后习题答案**

1. B

2. C

3. A

4. A

