适配器模式

基本概念

适配器，通俗来讲，就是一个转接头，使得原本不能兼容的事物变得兼容

例如目前大部分Android手机均提供Type-C耳机接口，但是大多数有线耳机都是3.5mm接口，所以需要一个转接头把它们连接起来

苹果的雷电接口转3.5mm接口

历史原因

同样以耳机接口为例，3.5mm耳机接口诞生于上世纪80、90年代，多用于随声听，一至延续至今日

在2012年左右，Type-C技术逐渐成熟，部分手机制造商认为可以将充电插口与耳机插口合二为一，进而降低手机尺寸以及内部电路空间

而Intel则实现了传输速度更快、多通道统一的雷电(Thunderbolt)接口，目前Apple的所有产品线均使用该接口作为电源接口，iPhone则取消了原有的3.5mm耳机接口，转而使用雷电接口

可以看到，技术的进步是必然的，那么新技术替换掉老技术也只是时间问题。但是在完全替换之前，仍需要兼容原有的3.5mm耳机接口 所以才有了耳机转接头

适配器模式 而代码中的适配器模式和现实生活中的转接头没什么区别，新的接口需要替换掉老的接口，但是又不能一次性全部替换掉，只能采用折中的技术进行处理，暂时兼容

实现

Java

```
interface IRead {
    void read();
}
```

假设有这样的一个接口，IRead，所有可读设备，例如磁盘、移动硬盘等都需要实现该接口

系统中通过"依赖于抽象，而不依赖于实现"将各种可读设备进行解耦与组合，能够应对大多数的设备数据读取

而陈旧光驱，即读取CD盘的接口并未实现该接口，实现的接口叫做CDRead接口，读取的方法名称也不叫read，而是叫readFromCD，那么现有的系统如何进行兼容?

```
interface CDRead {
    void readFromCD();
}


class CDReadImpl implements CDRead {
    @Override
    public void readFromCD() {/* 从CD中读取数据 */}
}
```

```
class NewCDReadImpl implements IRead {
    private CDReadImpl cdRead;

    public NewCDReadImpl(CDReadImpl cdRead) {
        this.cdRead = cdRead;
    }

    @Override
    public void read() {
        // 其实就是个代理
        cdRead.readFromCD();
    }
}
```

再创建一个新的CD数据读取类，并实现IRead接口，而实际的read()调用则仍是调用readFromCD()方法

从代码结构上来看，和代理模式基本没什么区别，但是适配器模式的目的与代理模式不同