前面我们看完了最后一种创建型模式:原型模式。接下来我们要进入到设计模式的第二大类:**结构性模式**。需要注意的是，在创建型模 性模式中就不同了，其注重的是侧重于如何将类或对象组合在一起形成更大的结构，以实现系统的高效协作和更好的扩展性。它们主要关注如何组织和组合类或对象，处理不同类之间的关系和交互，使得系统更灵活、可维护。也就是说在这个模式中一个相对重要的原则是**开放封闭原则**。

还有一个也很重要的原则:合成复用原则

来看一下第一个要学的结构性模式：**适配器模式**。

首先来说一下适配器模式起的作用。人如其名，适配器存在的目的就是起到一个适配的作用。在显示生活中，如果我们离开中国大陆，我们需要使用当地的电的话，我们通常需要一个适配器，就比如香港等地，你是无法使用你现有的接口去使用当地的插口的。你需要去购买一个能够接通俩头的适配器去转换电压或者提供插口之类的。

适配器的作用也是如此。就比如我们的程序中有着多个独立的类，当我们想要在一个类中去使用另一个类时，我们会先去考虑我们要使用的这个类有没有接口可以提供别的类使用，有自然就简单，但如果没有呢。没有的话我们有多种做法，最直接的就是去给这个被使用的类添加一个可供目标类使用的接口。但是这违背了开放封闭原则，所以我们需要考虑一些别的方法，这时候适配器模式就出现了。

还是那个问题，只说还是很抽象，上代码。

我们重新来分析下适配器模式下需要的几个层次。



在适配器模式中，存在的一个非常核心的概念就是包装，我们期望中的一个适配器是一个能够将想要整合进现有代码的外部类包装成我们能够直接使用现有的统一接口的类。

就比如把，我们在设计中规定了我们所有的渲染函数都必须是draw名，无论你是渲染矩形，圆形还是什么。现在我们想要在现有代码中去引入一个外部已经实现好的类，但是这个类的接口名不是draw，如果我们不包装直接使用，那么我们就需要时刻去注意我们编写的代码，这是很麻烦的。所以，我们考虑使用一层包装将这个外部类包装成跟我们现有的类在外部看起来一样的，我们只举最简单的例子，就比如我们只是包装它的函数名。那么我们就需要在这个适配器类中去构建我们现在的外部库向我们现有代码中兼容的装换原则。在这里可以见到视作是在外部用一个统一的函数名来包装调用我们实际的函数名。

好了，接下来我们来看一下适配器类中我们需要了解的层次。最底层就是我们的源接口，就是我们现在想要引入程序的代码，这个接口是不符合我们用户端当前使用的接口规范的。

在往上就是一个适配器的层次，在这个层次中，其规定了我们的源接口项目标接口转换的方法。这里我们对前面的简单情况进行扩展，我们前面只是说了只用变换名字调用函数的情况。让我们复杂一点，在类的设计中，由于我们期望的设计都应该是遵守设计模式三原则的，所以我们引入的这个类也是一个应该遵守开放封闭原则的类，所以我们的适配方法不应该是去修改我们的源接口中的实现的。

而在一般情况下，我们的这个适配器接收的参数是由用户层提供的，也就是我们装换后的接口提供的，那么，我们想要实现这俩者之间的适配，我们就得考虑我们当前目标层的接口之间与源接口之间需要来进行怎么样的对接。这个其实就是我们的电压适配器中那个物理原理(那个电压转换原理)。而这个就是我们的适配器类中所要注意的核心了，通过一个适配器提供的转换方法，我们将一个用户层接口所能提供的信息给进行一系列的加工，转换成了能够调用出来源接口的一系列信息。然后我们的适配器会去调用我们的源接口，这个就是我们适配器模式的核心。

重新回到我们那个最简单的模式来看一下，假设我们俩段的接口调用都不用传递或处理任何参数，那么我们这里的这个适配器接口将不用处理任何的细节，只需要在现有的封装接口中去调用我们的源接口即可，无需考虑参数间的处理。

然后我们再往上去进行探索，再往上其实就是我们用户接触的接口了。其实这个接口一般在设计中都是有一套接口规范来定义的。就比如你的渲染模块的渲染接口应该是什么名字，应该传递什么参数等等。这个接口是用户是直接调用的接口，直接屏蔽了用户与底层的交互。

最上层就是我们的用户层了，这一层没什么好说的，其实就是根据接口规范去找到对应的接口并进行调用的层次。

由于截屏问题，有些代码无法看到，自行脑补

