外观模式

外观模式，就是为了提供更简单的接口。书上的定义是，提供了一个统一的接口，用来访问子系统中的一群接口。外观定义了一个高层的接口，让子系统更容易使用。
借用网上的一张图来说明一下这个问题。（个人理解，欢迎拍砖，互相进步撒）

大家看图上的这个用来扫街的货是不是很强悍？（图下面的解释，很不错，有木有？看到一张这种图就能想到设计模式和 00 原则之类，真的很不错。）其实我觉得，这里面也可以看成是利用了外观模式。

现在假象这些组件都还能组装起来（就是说那些扫把、车、接口之类的全部还是零散的），那我们每次要让它像图中那样子工作的时候，是不是都得先找来一辆车，n 个扫把，还有适配器（就是能把扫把固定在车上的东东，也就是图上说的适配器模式的应用了撒），然后把它们一个个组装起来，然后再开动那辆车。那这样子不是很麻烦吗？要是每次你都要执行这些操作才能扫街，你会不耐烦的吧？反正是我的话我死了算了。

其实嘛，外观模式就相当于。。。。好吧，写到这里，我发现这个不是个很好的例子。不过，呐，我还是得继续。外观模式就相当于你多了一个下属（恭喜你，你升级为主管了），你每次觉得街道脏了，就跟他说，你去把这条街扫干净。（嗤，很搞笑的样子）。然后他就去组装好组件，接着就工作了。而你，只要说一句话就好了。是不是很爽啊？你只要一句话（相当于只调用一个方法），就能相当于你之前的 n 个步骤（之前那些组装过程全是你自己做哦亲）。当然这些你也可以自己做啊。外观模式的意图只是让接口变得更加简单，他提供简化接口的同时，依然将系统完整的功能暴露出来，一共需要的人使用。（例如，你的上级领导来了，你可以亲自去做，至于理由你们懂的）

可以看出，实现一个外观，需要将子系统组合进外观中，然后将工作委托给子系统执行。

既然上面提到适配器模式就顺带说下，适配器模式是将一个接口转成另一个接口，而外观模式是让接口更简单。

下面来大概写一下实现吧，不再编译器中写完整代码了，就直接在这里写了。

上面说了，要扫街就先要执行一些任务

1.找来扫把

2.找来一辆车

3.找来其他需要用到工具

4.用绳子或者什么把扫把固定在车上

5.开动车

现在我们把这些任务写成类和方法的调用

```
Broom.find();

Vehicle.find();

Tool.find();

Tool.assemble();

Vehicle.drive();
```

就这样，就涉及到三个类了有木有？试想一下，如果这是个非常繁杂的系统，你每次都要从头到尾执行一次，什么感觉你自己想吧。

现在就其创建一个外观，于是我们创建一个名为 CallSbToWork（卧槽，这名字很那啥，我知道。其实我想的时候是 somebody 的，但是我一写缩写，连我自己都笑了。好了，忽略忽略。。。）的类，也就是外观类，它暴露了一个方法——cleanStreet()。这个外观类将诸多组件看成一个子系统，通过调用子系统来实现 cleanStreet()方法。那你现在就可以调用其外观实现的方法来做扫街这件事了。也就是说，你只要调用一个方法就行了。

注意，外观只是提供你更直接的操作，并没有将原来的子系统屏蔽。你还是可以使用原来的子系统的。

```
public ClassCallSbToWork{

   Broom br;          //组合，我们要用到的组件都在这了。


  Vehicle ve;

   Tool tool;

   punlicCallSbToWork(Broom br,Vehicle ve,Tool tool){

            this.br=br;
```

```
            this.ve=ve;
            this.tool=tool;

    }
  public voidcleanStreet() {
        Broom.find();

         Vehicle.find();
         Tool.find();
         Tool.assemble();
         Vehicle.drive();
  }

}
```

然后你就可以创建 CallSbToWork 对象来调用 cleanStreet()方法来实现了。

至此，完毕。。。

原文链接：http://itxcl.diandian.com/post/2012-03-29/waiguan