外观模式

概念

- 定义：外观模式（Facade Pattern）是一种结构型设计模式，它为子系统的一组接口提供一个一致的界面（即定义一个高层接口），从而简化客户端对这些子系统的使用
- 结构：外观模式通常包含以下几部分
  - 外观类（Facade）：提供一个高层接口，使得子系统更容易使用
  - 子系统类（Subsystem Classes）：实现子系统的功能，外观类会调用这些类来完成实际的工作
- 优点
  - 简化接口：通过提供一个简单的接口，减少了客户端与子系统之间的耦合
  - 提高灵活性：子系统的变化不会影响到客户端，只需修改外观类即可
  - 更好的分层：有助于将复杂系统分层，使得每一层只关注自己的职责
- 缺点
  - 外观模式在增加新的子系统功能时，可能需要修改外观类，这违背了开闭原则
  - 当子系统非常复杂且分散，或子系统之间交互方式频繁变化，外观模式可能会导致外观类变得庞大臃肿

实例

- 现有三个子系统

```
// 子系统A
public class SubSystemA
{
    public void MethodA()
    {
        Console.WriteLine("执行子系统A中的方法A");
    }
}

// 子系统B
public class SubSystemB
{
    public void MethodB()
    {
        Console.WriteLine("执行子系统B中的方法B");
```

```
    }
}

// 子系统C
public class SubSystemC
{
    public void MethodC()
    {
        Console.WriteLine("执行子系统C中的方法C");
    }
}
```

- 不使用外观模式

```
/// <summary>
/// 不使用外观模式的情况
/// 客户端与三个子系统都耦合，使得客户端程序依赖与子系统
/// </summary>
class Client
{
    static void Main(string[] args)
    {
        SubSystemA a = new SubSystemA();
        SubSystemB b = new SubSystemB();
        SubSystemC c = new SubSystemC();
        a.MethodA();
        b.MethodB();
        c.MethodC();
        Console.Read();
    }
}
```

- 使用外观模式

```
// 外观类
public class Facade
{
    private SubSystemA _subSystemA;
    private SubSystemB _subSystemB;
    private SubSystemC _subSystemC;

    // 构造函数中实例化子系统
    public Facade()
```

```
    {
        _subSystemA = new SubSystemA();
        _subSystemB = new SubSystemB();
        _subSystemC = new SubSystemC();
    }

    // 把子系统的方法封装在外观类中
    public void Method()
    {
        _subSystemA.MethodA();
        _subSystemB.MethodB();
        _subSystemC.MethodC();
    }
}

// 客户端代码
class Client
{
    static void Main(string[] args)
    {
        Facade facade = new Facade();
        facade.Method();
        Console.Read();
    }
}
```