5.结构型模式-外观模式

概述：
外观模式又叫做门面模式，在面向对象程序设计中，解耦是一种推崇的理念，但事实上由于某些系统中过于复杂(或者代码写的比较烂)，从而增加了客户端与子系统之间的耦合度，例如：假设有一组火警报警系统，由三个子元件构成，一个警报器 一个喷水器 一个自动拨打电话的装置，小火警发生时则自动调用报警器和喷水器，大火警发生时则自动调用报警器 喷水器 和打电话装置．这种情况下可以采用外观模式，即引入一个类对子系统进行包装，让客户端与其进行交互．

外观模式(Facade Pattern)：外部与一个子系统的通信必须通过一个统一的外观对象进行，为子系统中的一组接口提供一个一致的界面，外观模式定义了一个高层接口，这个接口使得这一子系统更加容易使用．

外观模式的目的是通过建立一个中间类，把调用目标类的代码都封装好，例如有时候目标类有很多个，逐一得去调用它们会很麻烦，这样通过中间类封装好的接口，client的调用就好很简单．

In [1]:
```
#示例代码：火警报警系统(未使用设计模式)

class fireAlert:
    def run(self):
        print('this is fireAlert')


class waterPush:
    def run(self):
        print('start push water')


class callPhone:
    def run(self):
        print('calling.. 119')

if __name__ == '__main__':
    fire = fireAlert()
    water = waterPush()
    call = callPhone()

    fireStatus = 'big'

    if fireStatus == 'small':
        fire.run()
        water.run()
    elif fireStatus == 'big':
        fire.run()
        water.run()
        call.run()
```

```
this is fireAlert
start push water
calling.. 119
```

以上代码虽然也完成了业务需求，但是明显的是客户的负担较重，客户端与子系统的耦合度太大．如果在多个业务场景中需要启动三个部件，复制粘贴当仍然可以解决，但是减少重复代码应该是我们在编码过程中持续关注的事情，所以我们需要将现有代码进行封装．在设计模式中被封装成的新对象叫做外观．

In [2]:
```
#示例代码：应用外观模式改造代码

class fireAlert:
    def run(self):
        print('this is fireAlert')

class waterPush:
    def run(self):
        print('start push water')

class callPhone:
    def run(self):
        print('calling.. 119')

class fadeMode:
    def __init__(self):
        self.water = waterPush()
        self.fire = fireAlert()
        self.call = callPhone()

    def small_fire(self):
        self.fire.run()
        self.water.run()
    def big_fire(self):
        self.small_fire()
        self.call.run()


if __name__ == '__main__':
    x = fadeMode()
    fireStatus = 'small'
    if fireStatus == 'small':
        x.small_fire()
    elif fireStatus == 'big':
        x.big_fire()
```

```
this is fireAlert
start push water
```

根据"单一职责原则"，在软件中将一个系统划分为若干个子系统有利于降低整个系统的复杂性，一个常见的设计目标是使子系统间的通信和相互依赖关系达到最小，(高内聚 低耦合)而达到该目标的途径之一就是引入一个外观对象，它为子系统的访问提供了一个简单而单一的入口．通过引入一个新的外观类可以降低原有系统的复杂度，同时降低客户类与子系统类的耦合度．

外观模式要求一个子系统的外部与其内部的通信通过一个统一的外观对象进行，外观类将客户端与子系统的内部复杂性分隔开，使得客户端只需要与外观对象打交道，而不需要与子系统内部的很多对象打交道．外观模式的目的在于降低系统的复杂程度，外观模式从很大程度上提高了客户端使用的便捷性，使得客户端无须关心子系统的工作细节，通过外观角色即可调用相关功能．