1 门面模式 (Facade)

门面模式: 外部与子系统的通信必需通过一个统一的门面（facade）对象进行. 门面模式是对象形式的结构型设计模式. 门面模式提供一个高层次的接口, 使得子系统更易于使用.

基于模块化的子系统划分:

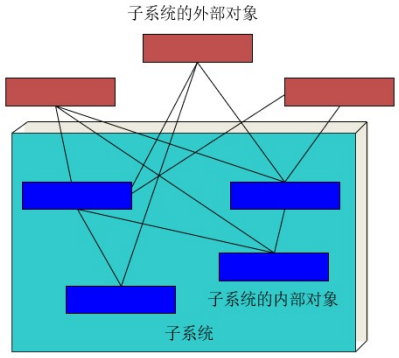
- 大型的软件系统都是比较复杂的, 软件设计的重要任务之一就是要合理控制软件复杂性.
- 处理复杂系统的一个常见方法便是将其 “分而治之”, 把一个系统划分为几个较小的子系统.
- 而使用一个子系统的客户端往往需要同时与子系统内部的许多对象交互.

什么情况下使用门面模式:

- **为一个复杂子系统提供一个简单接口**: 子系统往往因为不断演化而变得越来越复杂, 使用门面模式可以使得子系统更具有可复用性.Facade 模式可以提供一个简单的操作视图.
- **提高子系统的独立性**: 一般而言, 子系统和其他的子系统之间、客户端与实现层之间存在着很大的依赖性. 引入 Facade 模式将一个子系统与它的客户端以及其他的子系统分离, 可以提高子系统的独立性和可移植性.
- **形成层次化结构**: 在构建一个层次化的系统时, 可以使用 Facade 模式定义系统中每一层的入口. 如果层与层之间是相互依赖的, 则可以限定它们仅通过 Facade 进行通信, 从而简化了层与层之间的依赖关系.

迪米特法则: 迪米特法则说:“只与你直接的朋友们通信”. 迪米特法要求每一个对象与其他对象的相互作用均是短程的, 而不是长程的. 只要有可能, 朋友的数目越少越好. 即一个对象只应当知道它的直接合作者的接口. 门面模式创造出一个门面对象, 将客户端所涉及的属于一个子系统的协作伙伴的数目减到最少, 使得客户端与子系统内部的对象的相互作用被门面对象所取代.

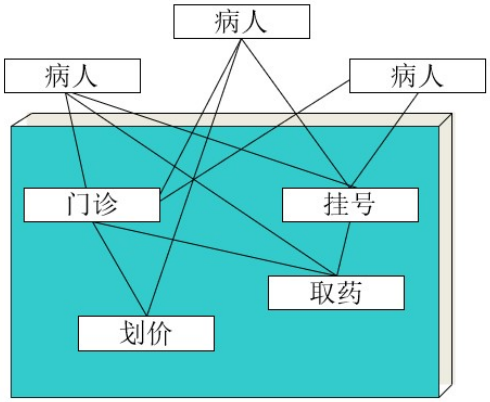
不同子系统间对象交互如图: 该图描述的是一个客户端必须与许多对象打交道才能完成一个功能, 图中的大方框代表一个子系统.



1.1 例子: 医院

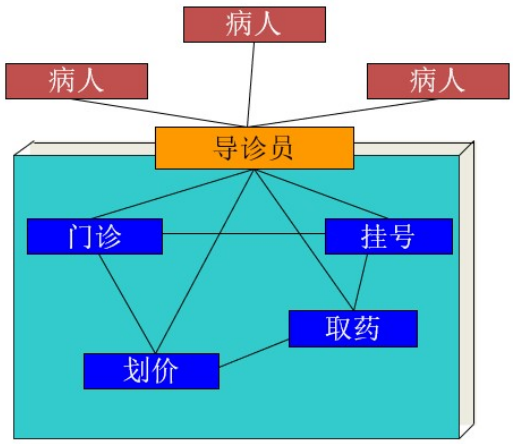
若把医院作为一个子系统, 按照部门职能, 这个系统可以划分为挂号、门诊、划价、化验、收费、取药等部门. 看病的病人要与这些部门打交道, 就如同一个子系统的客户端与一子系统的

各个类打交道一样, 不是一件容易的事情. 首先病人必须挂号, 然后门诊. 如果医生要求化验, 病人必须先划价, 然后缴款, 才能到化验部门做化验. 化验后, 再回到门诊室.



上图描述的是病人在医院里的体验，图中的方框代表医院。

解决这种不便的方法就是引进门面模式. 仍然通过医院的例子说明, 可以设置一个导诊员的位置, 由接待员负责代为挂号、划价、缴费、取药等. 这个接待员就是门面模式的体现, 病人只接触接待员, 由接待员负责与医院的各个部门打交道, 如图所示.

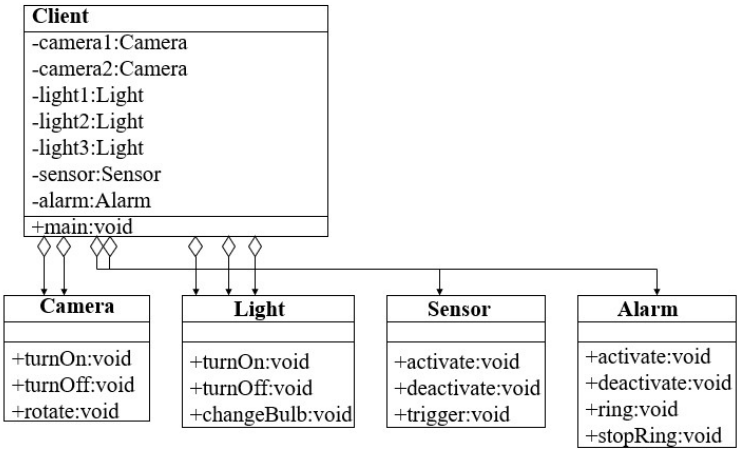


1.2 例子: 安全监控系统

一个安全监控系统由两个录像机、三个电灯、一个传感器和一个警报器组成. 安全监控系统的操作人员需要经常将这些仪器启动和关闭. 首先, 在不使用门面模式的情况下, 操作这个系统的操作员必须直接操作所有的这些设备.

1.2.1 不使用门面模式的设计

不使用门面模式的情况下, 系统的设计图如图所示, 可看出,Client 对象需要引用到所有的录象机 (Camera)、电灯 (Light)、感应器 (Sensor) 和警报器 (Alarm) 对象.Client 对象必须对安全监控系统全知全能, 这是一个高耦合度的做法.




```java
public class Camera {
  public void turnOn() { //打开录象机
    System.out.println("Turning on the camera.");
  }
  public void turnOff() { // 关闭录象机
    System.out.println("Turning off the camera.");
  }
  public void rotate(int degrees) { //转动录象机
    System.out.println("rotating the camera by " + degrees + " degrees");
  }
}
```

```java
public class Light {
  public void turnOn() { //打开灯
    System.out.println("Turning on the camera.");
  }
  public void turnOff() { //关闭灯
    System.out.println("Turning off the camera.");
  }
  public void changeBulb() { //换灯泡
    System.out.println("changing the light-bulb.");
  }
}
```

```java
public class Sensor {
  public void activate() { //启动感应器
    System.out.println("Activating the sensor.");
  }
```

```
  public void deactivate() { //关闭感应器
    System.out.println("Deactivating the sensor.");
  }
  public void trigger() { //触发感应器
    System.out.println("The sensor has been triggered.");
  }
}
```

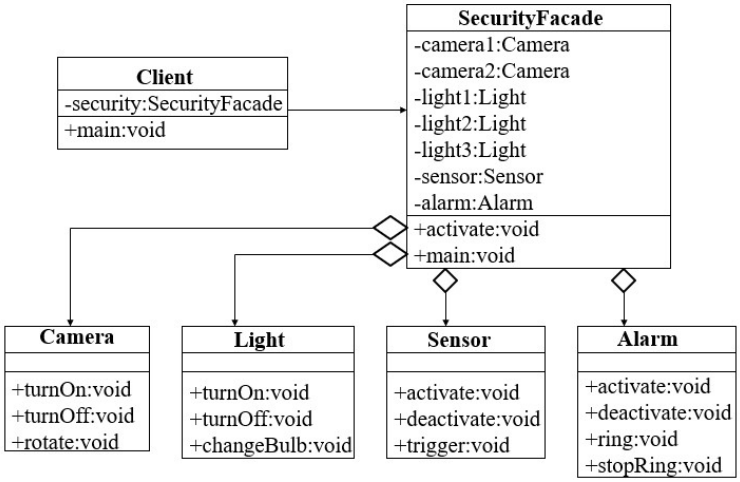
```
public class Alarm {
  public void activate() { //启动警报器
    System.out.println("Activating the alarm.");
  }
  public void deactivate() { //关闭警报器
    System.out.println("Deactivating the alarm.");
  }
  public void ring() { //拉响警报器
    System.out.println("Ringing the alarm.");
  }
}
```

```
public class Client {
  static private Camera camera1, camera2;
  static private Light light1, light2, light3;
  static private Sensor sensor;
  static private Alarm alarm;
  public static void main(String[] args) {
    camera1.turnOn();
    camera2.turnOn();
    light1.turnOn();
    light2.turnOn();
    light3.turnOn();
    sensor.activate();
    alarm.activate();
  }
}
```

1.2.2 使用门面模式的设计

一个合理的改进方案就是准备一个系统的控制台, 作为安全监控系统的用户界面. 用户只需要操作这个简化的界面就可以操控所有的内部设备, 这实际上就是门面模式的用意. 使用门面

模式对前面的设计方案进行改造后, 就得到了如下页图所示的设计图, 可以看出, 门面 Security-Facade 对象承担了与安全监控系统内部各个对象打交道的任务, 而客户对象只需要与门面对象打交道即可. 这是客户端与安全监控系统之间的一个门户, 它使二者之间的关系变得简单和易于管理.




```
1  public class SecurityFacade {
2    private Camera camera1, camera2;
3    private Light light1, light2, light3;
4    private Sensor sensor;
5    private Alarm alarm;
6    public void activate() {
7      camera1.turnOn();
8      camera2.turnOn();
9      light1.turnOn();
10     light2.turnOn();
11     light3.turnOn();
12     sensor.activate();
13     alarm.activate();
14   }
15   public void deactivate() {
16     camera1.turnOff();
17     camera2.turnOff();
18     light1.turnOff();
19     light2.turnOff();
20     light3.turnOff();
21     sensor.deactivate();
22     alarm.deactivate();
```

```
  }
}
```

```
public class Client {
  private static SecurityFacade security;
  public static void main(String[] args) { security.activate(); }
}
```

可以看出, 客户端程序大大简化了,Client 类只有一个对 SecurityFacade 类的引用.