



湖南理工学院  
Hunan Institute of Science and Technology

# 实验报告

课 程： 软件设计模式

班 级： 软件 21-2BF

姓 名： 陈琳雍

学 号： 14214801971

日 期： 2024/5/15

## 实验六 外观模式与享元模式实验

### 一. 实验目的

- 1) 初步了解和掌握外观模式 (Facade) 的类图结构, 以及主要的模式角色;
- 2) 理解外观模式的基本构造, 并通过掌握的编程语言, 完成实验要求的内容;
- 3) 充分理解和掌握装饰模式如何统一的外观对象进行, 为子系统中的一组接口提供一个统一的入口。
- 4) 初步了解和掌握享元模式 (Flyweight) 的类图结构, 以及主要的模式角色;
- 5) 理解享元模式的基本构造, 并通过掌握的编程语言, 完成实验要求的内容;
- 6) 充分理解和掌握装饰模式如何共享对象, 使得有效地支持大量的细粒度的对象。

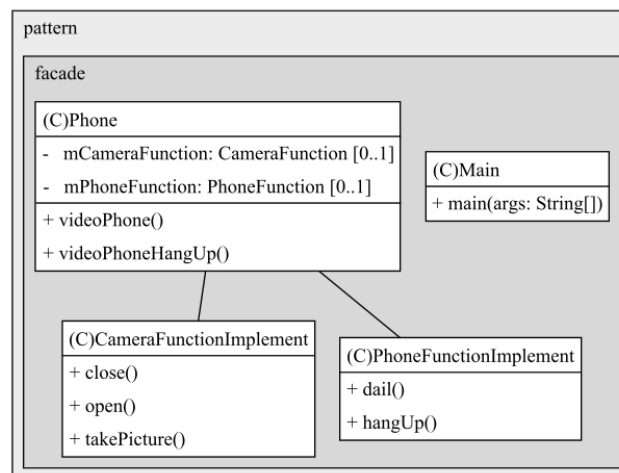
### 二. 实验内容

#### 1. 实验场景的设计如下:

生活中使用外观模式的例子非常多, 任何一个类似中央调度结构的组织都类似外观模式。手机就是一个外观模式的例子, 它集合了电话功能、短信功能、GPS、拍照等于一身, 通过手机你就可以完成各种功能, 而不是当你打电话时使用手机, 要拍照时非得用一个相机, 如果每使用一个功能你就必须操作特定的设备, 会使得整个过程很繁琐。而手机给了你一个统一的入口, 集电话、上网、拍照等功能于一身, 使用方便, 操作简单。

实验中, 结合摄像、电话功能, 扩展视频通话。

### 三. 实验过程



下面简单模拟一下手机的外观模式实现, 首先我们建立一个 Phone 类。代码大致如下:

```

public class Phone 2 个用法
{
    — private final PhoneFunction mPhoneFunction = new PhoneFunctionImplement(); 2
    — private final CameraFunction mCameraFunction = new CameraFunctionImplement();

    — public void videoPhone() 1 个用法
    — {
    —     mCameraFunction.open();
    —     mPhoneFunction.dail();
    — }

    — public void videoPhoneHangUp() 1 个用法
    — {
    —     mCameraFunction.close();
    —     mPhoneFunction.hangUp();
    — }
}

```

Phone 类中含有两个子系统，也就是拨号系统和拍照系统，Phone 将这两个系统封装起来，为用户提供一个统一的操作接口，也就是说用户只需要通过 Phone 这个类就可以操作打电话和拍照这两个功能。

用户不需要知道有 PhoneFunction 这个接口以及它的实现类，同样也不需要知道 Camera 相关的信息，通过 Phone 就可以包揽一切。而在 Phone 中也封装了两个子系统的交互，例如视频电话时需要先打开摄像头，然后再开始拨号，如果没有这一步的封装，每次用户实现视频通话功能时都需要手动打开摄像头、进行拨号，这样会增加用户的使用成本，外观模式使得这些操作更加简单、易用。

我们来看看 PhoneFunction 接口和 PhoneFunctionImplement。

```

public interface PhoneFunction
{
    — void dail(); 1 个用法 1 个多
    — void hangUp(); 1 个用法 1 个多
}

```

```

package design.pattern.facade;

public class PhoneFunctionImplement implements PhoneFunction
{
    — @Override 1 个用法
    — public void dail()
    — {
    —     System.out.println("Make a phone call.");
    — }

    — @Override 1 个用法
    — public void hangUp()
    — {
    —     System.out.println("Hang up the phone.");
    — }
}

```

Camera 也是类似的实现，具体代码如下：

```

public interface CameraFunction
{
—— void open(); 1 个用法 1 个实
—— void takePicture(); 0 个用法
—— void close(); 1 个用法 1 个实
}

public class CameraFunctionImplement implements CameraFunction 1 个用法
{
—— @Override 1 个用法
—— public void open() { System.out.println("Turn on the camera."); }
—— @Override 0 个用法
—— public void takePicture()
—— {
——     System.out.println("Take a picture.");
—— }
—— @Override 1 个用法
—— public void close() { System.out.println("Turn off the camera."); }
}

```

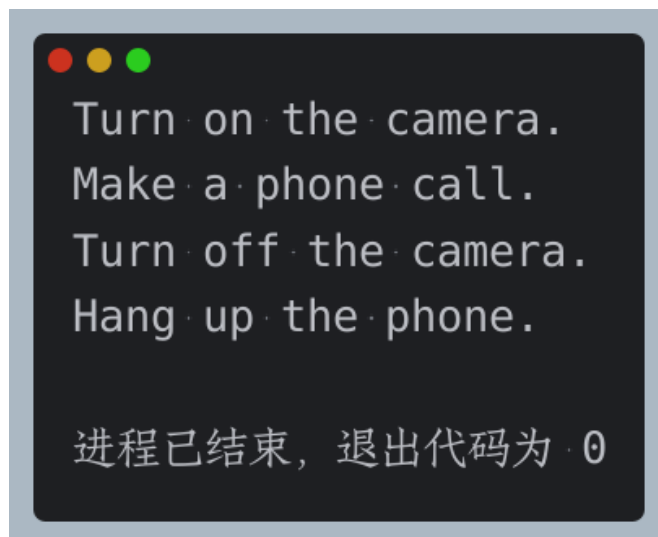
#### 四. 调试和运行结果

```

public class Main
{
—— /**
——  * 外观模式:
——  * 统一封装接口, 隐藏具体实现的子系统逻辑, 以隔离变化, 提供一个高层次接口, 使系统更加灵活、易用。
——  *
——  * @param args null
——  */
—— public static void main(String[] args)
—— {
——     Phone phone = new Phone();
——     phone.videoPhone();
——     phone.videoPhoneHangUp();
—— }
}

```

结果:



```

Turn on the camera.
Make a phone call.
Turn off the camera.
Hang up the phone.

进程已结束, 退出代码为 0

```

#### 五. 实验总结

在实验中, 我发现, 外观模式就是统一接口封装。将子系统的逻辑、交互隐藏起来, 为用户提供一个高层次的接口, 使得系统更加易用, 同时也对外隐藏了具体的实现, 这样即使具体的子系统发生了变化, 用户也不会感知到, 因为用户使用的是 **Facade** 高层接口, 内部的变化对于用户来说并不可见。这样一来就

将变化隔离开来，使得系统也更为灵活。