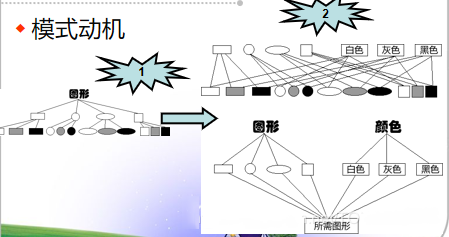
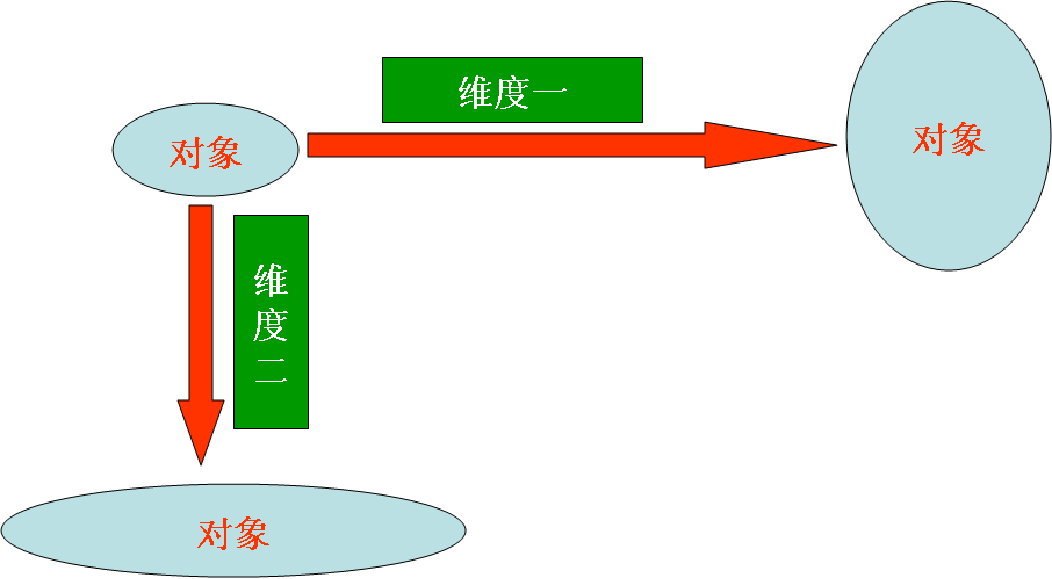
### **桥接模式**

#### **1、模式动机**

·设想如果要绘制矩形、圆形、椭圆、正方形，我们至少需要4个形状类，但是如果绘制的图形需要具有不同的颜色，如红色、绿色、蓝色等，此时至少有如下两种设计方案：  
第一种设计方案是为每一种形状都提供一套各种颜色的版本。  
第二种设计方案是根据实际需要对形状和颜色进行组合



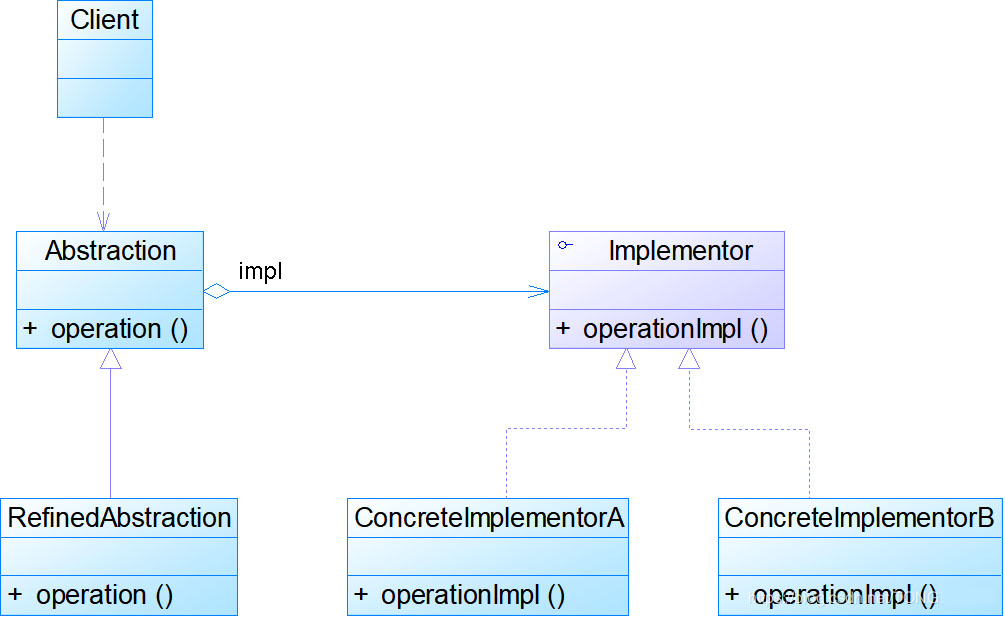
对于有两个变化维度（即两个变化的原因）的系统，采用方案二来进行设计系统中类的个数更少，且系统扩展更为方便。设计方案二即是桥接模式的应用。桥接模式将继承关系转换为关联关系，从而降低了类与类之间的耦合，减少了代码编写量。



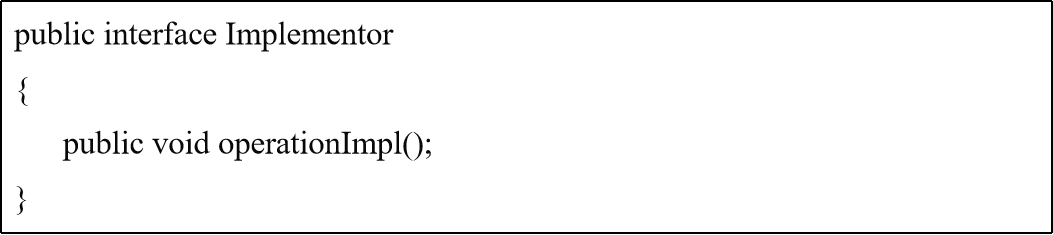
#### **2、模式定义**

* 桥接模式(Bridge Pattern)：将抽象部分与它的实现部分分离，使它们都可以独立地变化。它是一种对象结构型模式，又称为柄体(Handle and Body)模式或接口(Interface)模式。

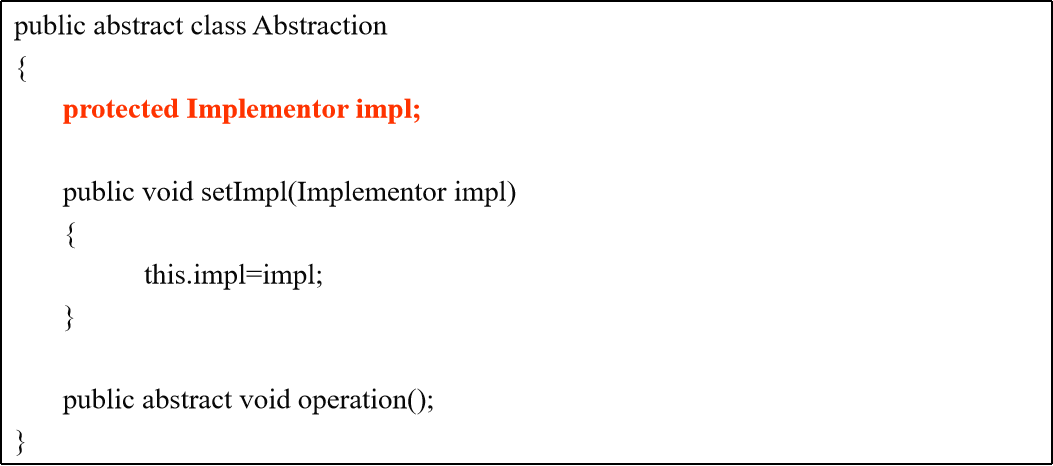
#### **3、模式结构**



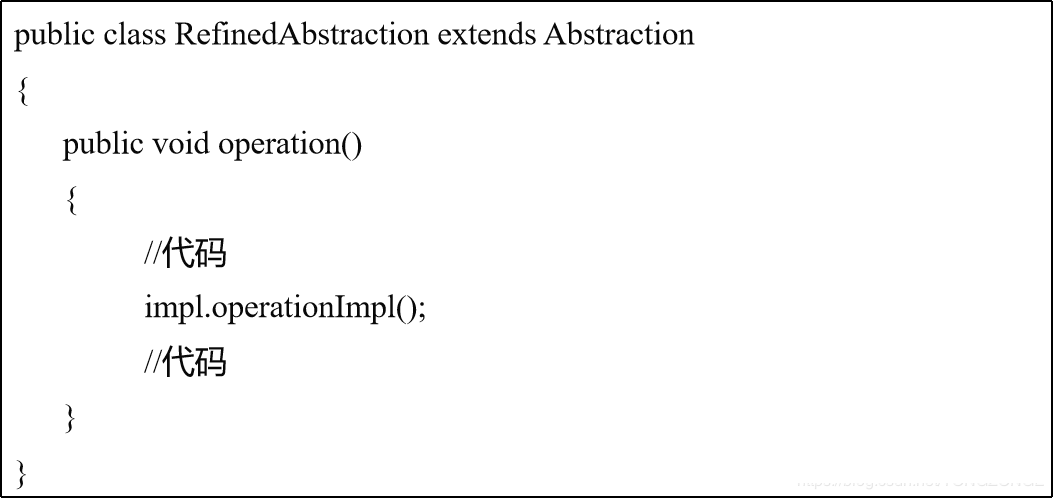
典型的实现类接口代码：



典型的抽象类代码：



典型的扩充抽象类代码：



4、模式分析

理解桥接模式，重点需要理解如何将抽象化(Abstraction)与实现化(Implementation)脱耦，使得二者可以独立地变化。

抽象化：抽象化就是忽略一些信息，把不同的实体当作同样的实体对待。在面向对象中，将对象的共同性质抽取出来形成类的过程即为抽象化的过程。

实现化：针对抽象化给出的具体实现，就是实现化，抽象化与实现化是一对互逆的概念，实现化产生的对象比抽象化更具体，是对抽象化事物的进一步具体化的产物。

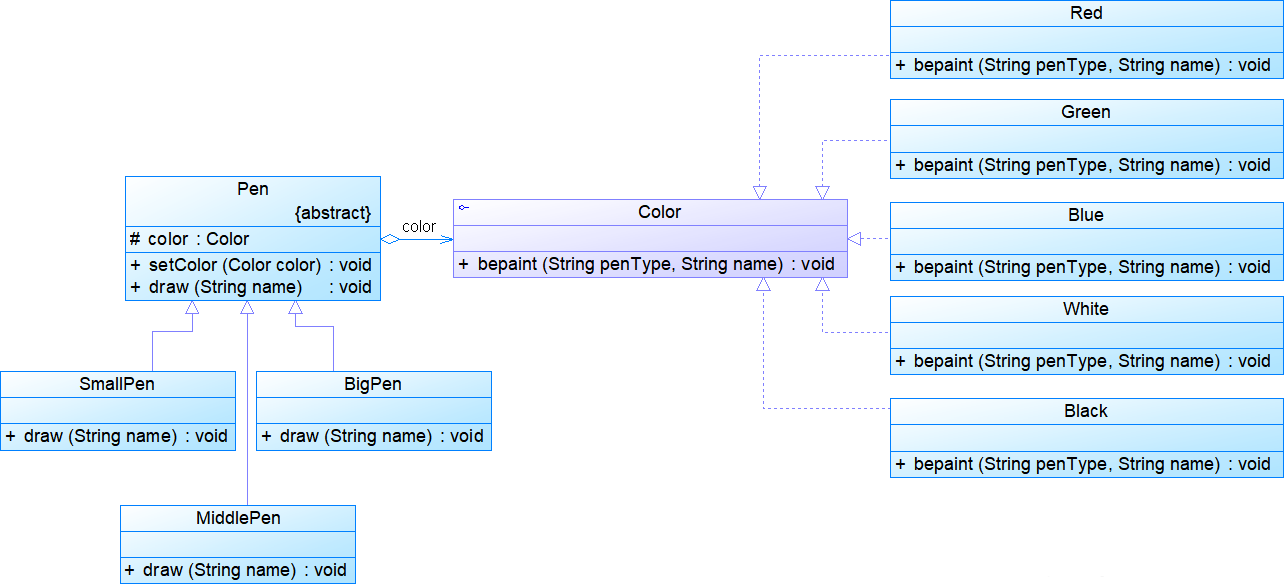
脱耦：脱耦就是将抽象化和实现化之间的耦合解脱开，或者说是将它们之间的强关联改换成弱关联，将两个角色之间的继承关系改为关联关系。桥接模式中的所谓脱耦，就是指在一个软件系统的抽象化和实现化之间使用关联关系（组合或者聚合关系）而不是继承关系，从而使两者可以相对独立地变化，这就是桥接模式的用意。

5、模式用例与分析

（1）实验毛笔

现需要提供大中小3种型号的画笔，能够绘制5种不同颜色，如果使用蜡笔，我们需要准备3\*5=15支蜡笔，也就是说必须准备15个具体的蜡笔类。而如果使用毛笔的话，只需要3种型号的毛笔，外加5个颜料盒，用3+5=8个类就可以实现15支蜡笔的功能。本实例使用桥接模式来模拟毛笔的使用过程。

uml图



* 代码实现：

package 桥接模式2;

//实现类接口

public interface Color {

void bepaint(String PenType,String name);

}

package 桥接模式2;

//具体实现类

public class Blue implements Color

{

@Override

public void bepaint(String PenType, String name) {

System.out.println(PenType + "蓝色的" + name + ".");

}

}

package 桥接模式2;

//具体实现类

public class Green implements Color {

@Override

public void bepaint(String PenType, String name) {

System.out.println(PenType + "绿色的" + name + ".");

}

}

package 桥接模式2;

//具体实现类

public class Red implements Color {

@Override

public void bepaint(String PenType, String name) {

System.out.println(PenType + "红色的" + name + ".");

}

}

package 桥接模式2;

//抽象类

public abstract class Pen {

protected Color color;

public void setColor(Color color) {

this.color = color;

}

public abstract void draw(String name);

}

package 桥接模式2;

//抽象类

public abstract class Pen {

protected Color color;

public void setColor(Color color) {

this.color = color;

}

public abstract void draw(String name);

}

package 桥接模式2;

//扩充抽象类

public class SmailPen extends Pen {

@Override

public void draw(String name) {

String penType = "小号毛笔绘制";

this.color.bepaint(penType,name);

}

}

package 桥接模式2;

//扩充抽象类

public class MiddlePen extends Pen {

@Override

public void draw(String name) {

String penType = "中号毛笔绘制";

this.color.bepaint(penType,name);

}

}

package 桥接模式2;

//扩充抽象类

public class BigPen extends Pen {

@Override

public void draw(String name) {

String penType = "大号毛笔绘制";

this.color.bepaint(penType,name);

}

}

package 桥接模式2;

import 工厂方法模式2.XMLUtil;

//客户端代码1

public class Client {

public static void main(String[] args) {

Color color ;

SmailPen s = new SmailPen();

color = (Color) XMLUtil.getBean();

s.setColor(color);

s.draw("鲜花");

}

}

* config.xml配置

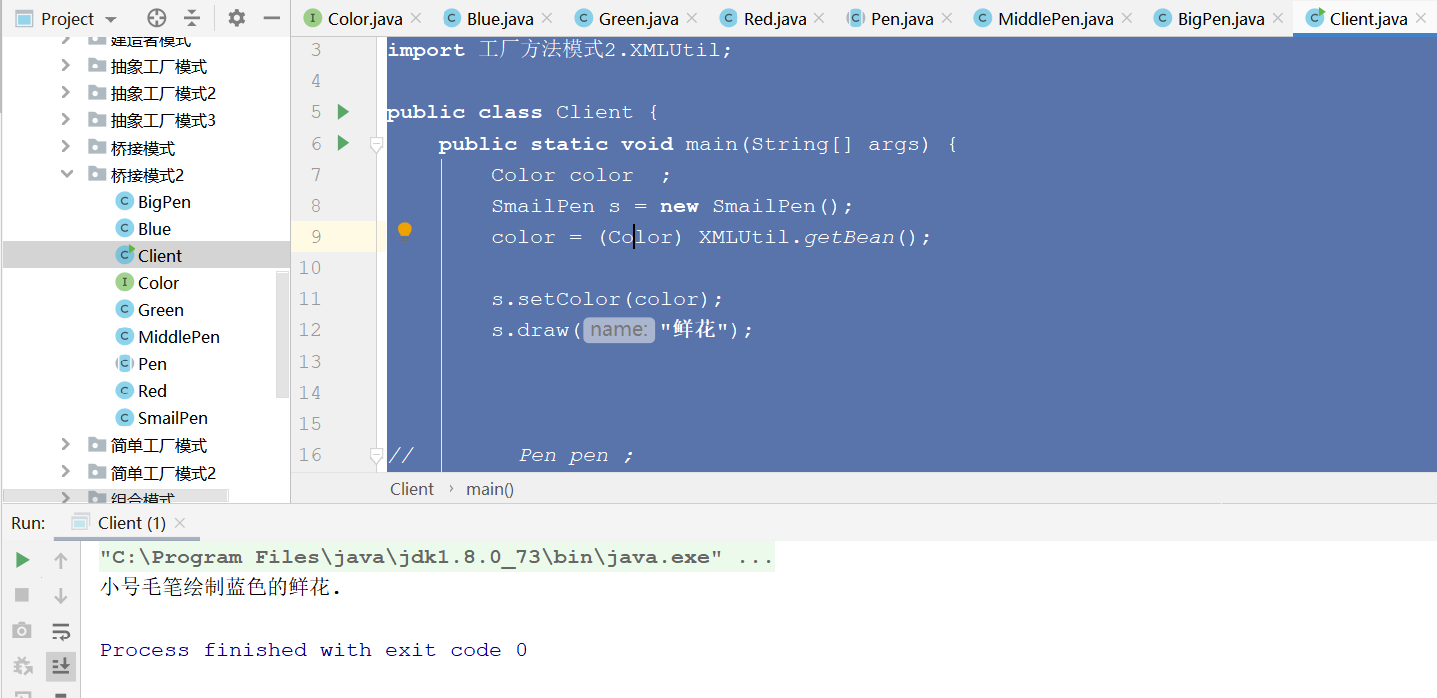
<?xml version="1.0"?>

<config>

<className>桥接模式2.Blue</className>

</config>

结果展示：



package 桥接模式2;

import 工厂方法模式2.XMLUtil;

//客户端代码2

public class Client {

public static void main(String[] args) {

Pen pen ;

Blue blue = new Blue();

pen = (Pen) XMLUtil.getBean();

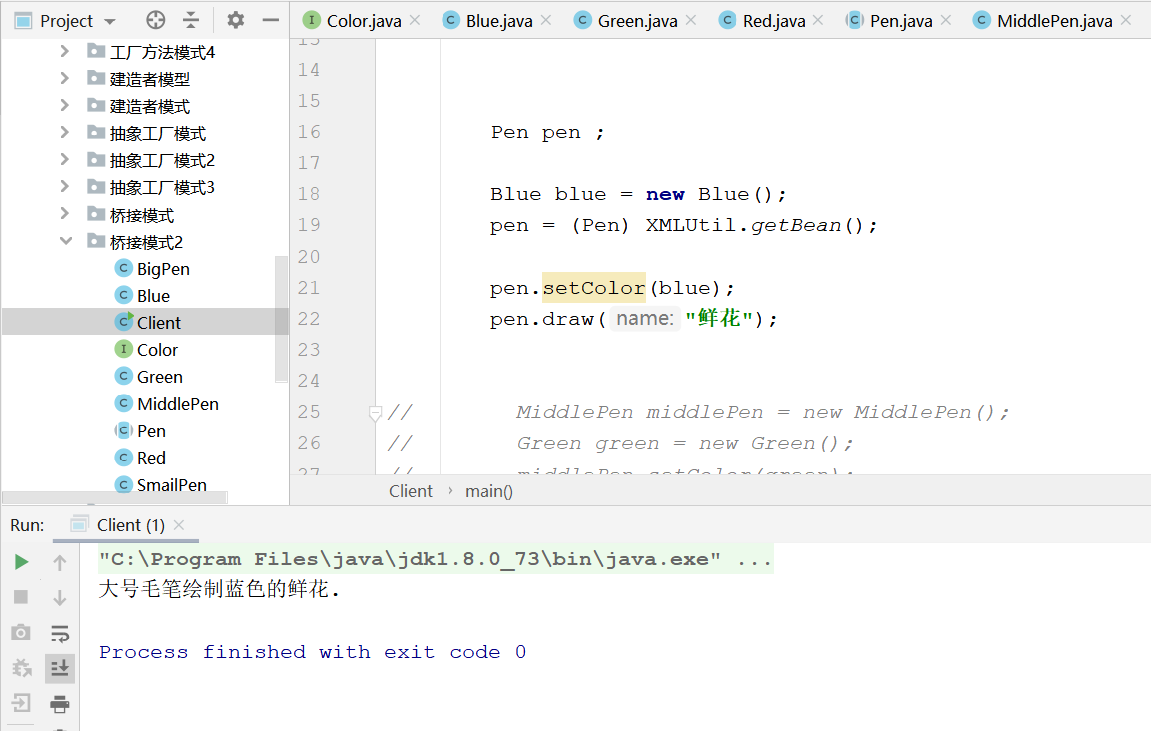
pen.setColor(blue);

pen.draw("鲜花");

}

}

结果展示：



package 桥接模式2;

客户端代码3

public class Client {

public static void main(String[] args) {

MiddlePen middlePen = new MiddlePen();

Green green = new Green();

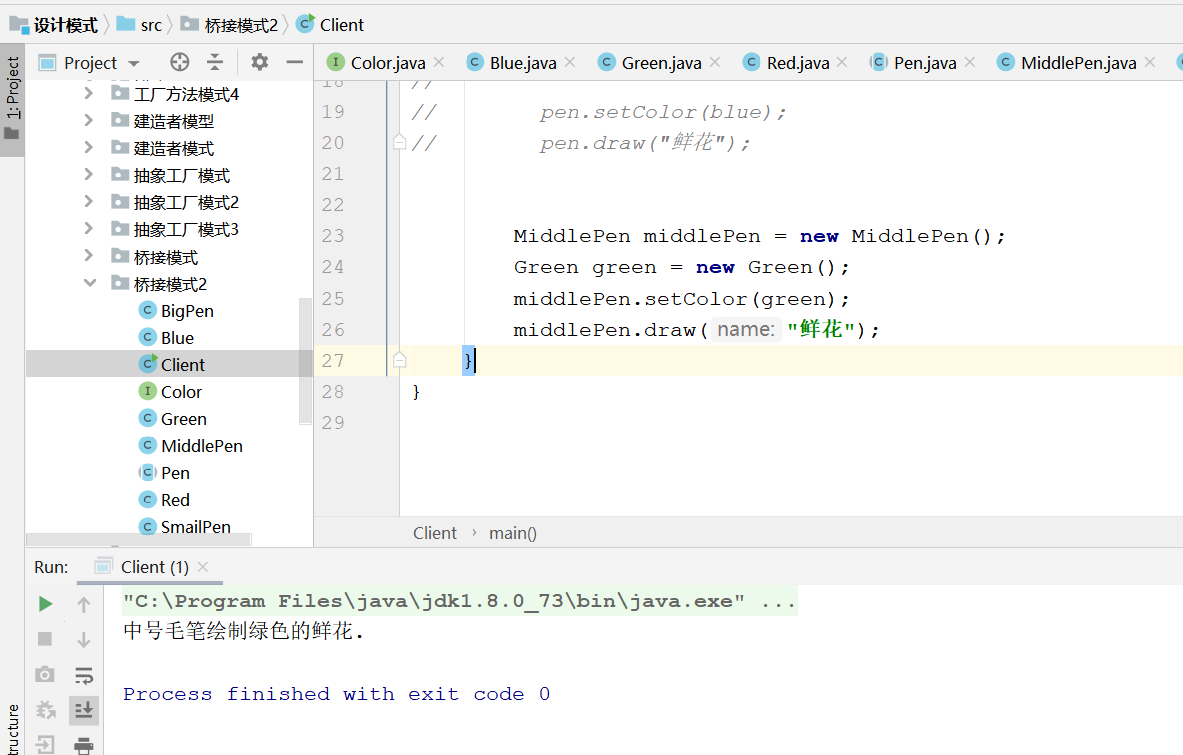
middlePen.setColor(green);

middlePen.draw("鲜花");

}

}

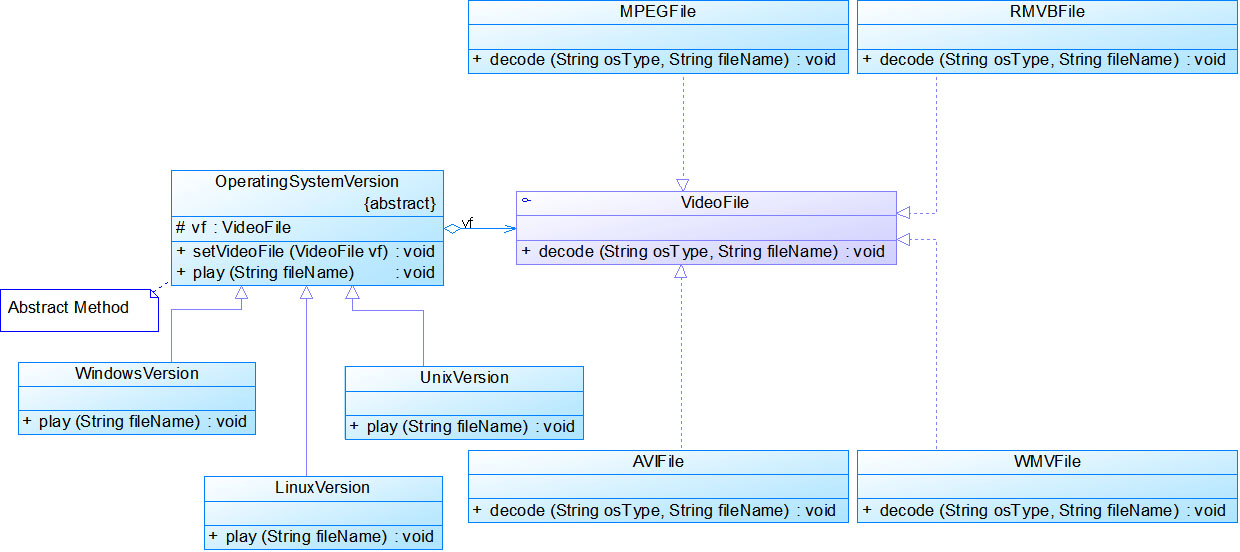
结果展示：



##### **（2）跨平台视频播放器**

* 如果需要开发一个跨平台视频播放器，可以在不同操作系统平台（如Windows、Linux、Unix等）上播放多种格式的视频文件，常见的视频格式包括MPEG、RMVB、AVI、WMV等。现使用桥接模式设计该播放器。

uml图



* 实现类接口

package 桥接模式3;

//抽象视频文件类 :

public interface VideoFile {

// decode方法

public void decode(String Type,String fileName)

}

* 具体实现类

package 桥接模式3;

//RMVB格式视频文件类 ：

public class RMVBFile implements VideoFile {

@Override

public void decode(String Type, String fileName) {

System.out.println("RMVB格式视频" + fileName + "在" + Type + "平台中解码播放");

}

}

package 桥接模式3;

//MPEG格式视频文件类

public class MPEGFile implements VideoFile {

@Override

public void decode(String Type, String fileName) {

System.out.println("MPEG格式视频" + fileName + "在" +Type +"平台中解码播放");

}

}

package 桥接模式3;

//AVI格式视频文件类 ：具体实现类

public class AVIFile implements VideoFile {

@Override

public void decode(String Type, String fileName) {

System.out.println("AVI格式视频" + fileName +"在" + Type + "平台中解码播放");

}

}

package 桥接模式3;

//WMV格式视频文件类 ：具体实现类

public class WMVFile implements VideoFile {

@Override

public void decode(String Type, String fileName) {

System.out.println("WMV格式视频" + fileName + "在" + Type + "平台中解码播放");

}

}

* 抽象类

package 桥接模式3;

//抽象类

public abstract class OperatingSystemVersion {

protected VideoFile vf;

//setVideoFile方法

public void setVideoFile(VideoFile vf) {

this.vf = vf;

}

//抽象play方法

public abstract void play(String fileName);

}

* 扩充抽象类

package 桥接模式3;

//Linux视频播放器： 扩充抽象类

public class LinuxVersion extends OperatingSystemVersion {

@Override

public void play(String fileName) {

vf.decode("Linux",fileName);

}

}

package 桥接模式3;

//Unix视频播放器： 扩充抽象类

public class UnixVersion extends OperatingSystemVersion {

@Override

public void play(String fileName) {

vf.decode("Unix",fileName);

}

}

package 桥接模式3;

//Windows视频播放器： 扩充抽象类

public class WindowsVersion extends OperatingSystemVersion {

@Override

public void play(String fileName) {

vf.decode("Windows",fileName);

}

}

客户端检测代码：

package 桥接模式3;

import 工厂方法模式2.XMLUtil;

//客户端测试类

public class client {

public static void main(String[] args) {

VideoFile file;

OperatingSystemVersion version;

file = new MPEGFile();

version = (OperatingSystemVersion) XMLUtil.getBean();

version.setVideoFile(file);

version.play("《大鱼海棠》");

* config.xml

<?xml version="1.0"?>

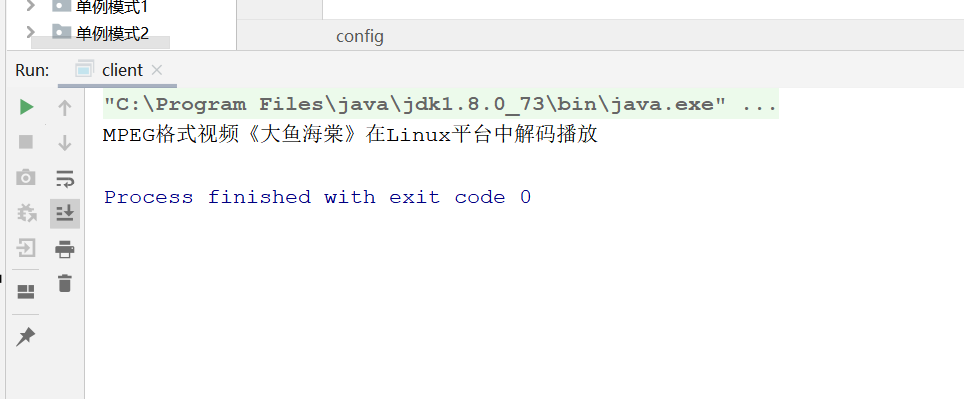
<config>

<className>桥接模式3.LinuxVersion</className>

<!--<className>桥接模式3.MPEGFile</className>-->

</config>

结果展示：



6、模式优缺点

优点

1、分离抽象接口及其实现部分。

2、桥接模式有时类似于多继承方案，但是多继承方案违背了类的单一职责原则（即一个类只有一个变化的原因），复用性比较差，而且多继承结构中类的个数非常庞大，桥接模式是比多继承方案更好的解决方法。

3、桥接模式提高了系统的可扩充性，在两个变化维度中任意扩展一个维度，都不需要修改原有系统。

4、实现细节对客户透明，可以对用户隐藏实现细节。

缺点

1、桥接模式的引入会增加系统的理解与设计难度，由于聚合关联关系建立在抽象层，要求开发者针对抽象进行设计与编程。

2、桥接模式要求正确识别出系统中两个独立变化的维度，因此其使用范围具有一定的局限性。

7、模式的适用环境

如果一个系统需要在构件的抽象化角色和具体化角色之间增加更多的灵活性，避免在两个层次之间建立静态的继承联系，通过桥接模式可以使它们在抽象层建立一个关联关系。

抽象化角色和实现化角色可以以继承的方式独立扩展而互不影响，在程序运行时可以动态将一个抽象化子类的对象和一个实现化子类的对象进行组合，即系统需要对抽象化角色和实现化角色进行动态耦合。

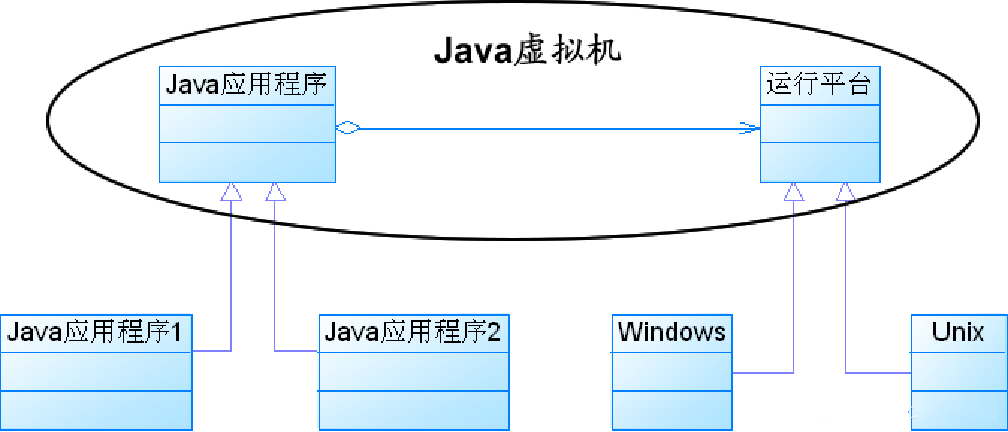
一个类存在两个独立变化的维度，且这两个维度都需要进行扩展。

虽然在系统中使用继承是没有问题的，但是由于抽象化角色和具体化角色需要独立变化，设计要求需要独立管理这两者。

对于那些不希望使用继承或因为多层次继承导致系统类的个数急剧增加的系统，桥接模式尤为适用。

#### **8、模式应用**

* Java语言通过Java虚拟机实现了平台的无关性。



适配器模式与桥接模式的联用

桥接模式和适配器模式用于设计的不同阶段，桥接模式用于系统的初步设计，对于存在两个独立变化维度的类可以将其分为抽象化和实现化两个角色，使它们可以分别进行变化；而在初步设计完成之后，当发现系统与已有类无法协同工作时，可以采用适配器模式。但有时候在设计初期也需要考虑适配器模式，特别是那些涉及到大量第三方应用接口的情况。

适配器模式与桥接模式的联用

