实验报告

实验设计名称： 桥接模式

专 业： 软 件 技术

班 级： 2018

姓 名： 林志龙

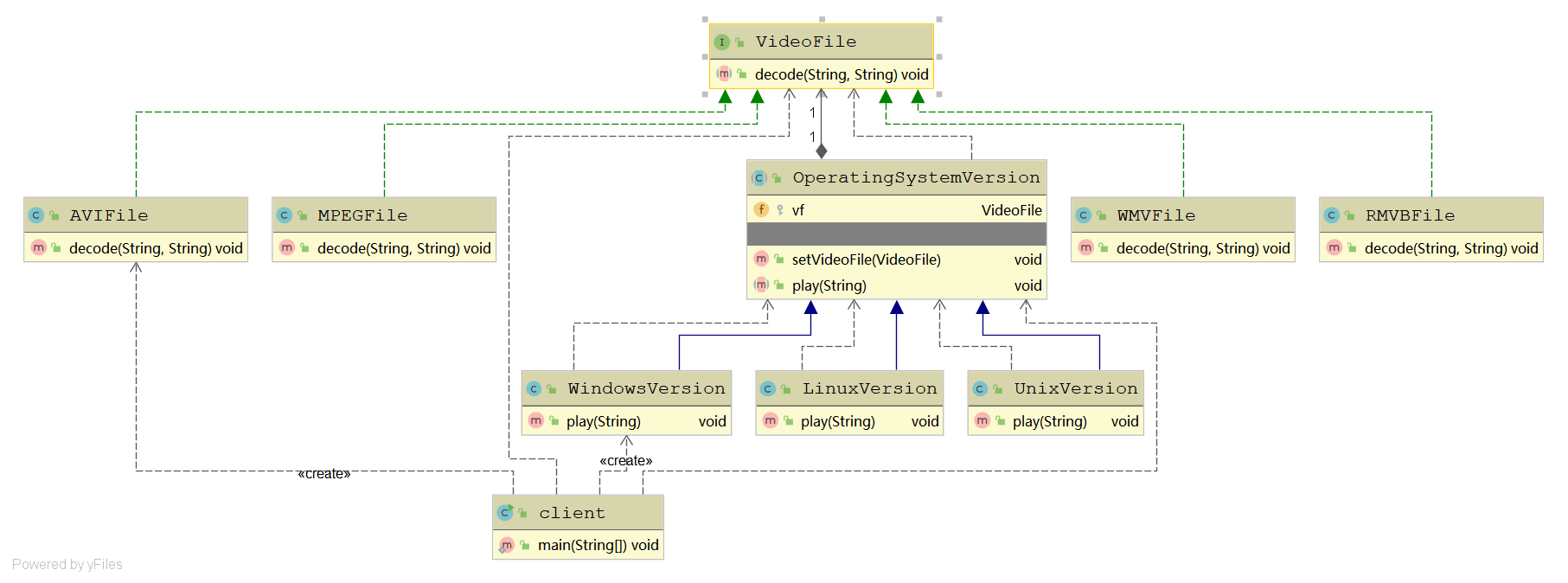
学 号： 1817700302

2020 年 4 月 15 日

1. 实验内容

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。

1. 实验设计类图(UML)



1. 实验代码

桥接模式包含如下角色：①Abstraction：抽象类；②RefinedAbstraction：扩充抽象类；③Implementor：实现类接口；④ConcreteImplementor：具体实现类 。

1. 抽象类代码实现：

**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);  
}

1. 扩充抽象类代码实现：

**package** 桥接模式3;  
*//Linux视频播放器：*

**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：***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：***public class** WindowsVersion **extends** OperatingSystemVersion {  
 @Override  
 **public void** play(String fileName) {  
 **vf**.decode(**"Windows"**,fileName);  
 }  
}

1. 实现类接口代码实现：

**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);  
  
}

1. 具体实现类代码实现：

**package** 桥接模式3;  
*//AVI格式视频文件类 ：***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;  
*//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;  
*//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;  
*//WMV格式视频文件类 ：***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"**);  
  
 }  
}

1. 客户端测试类：

**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》"**);  
  
*// file = (VideoFile) XMLUtil.getBean();  
// version = new UnixVersion();  
// version.setVideoFile(file);  
// version.play("《大黄蜂》");  
  
// file = new AVIFile();  
// version = new WindowsVersion();  
// version.setVideoFile(file);  
// version.play("《末代皇帝》");* }  
}

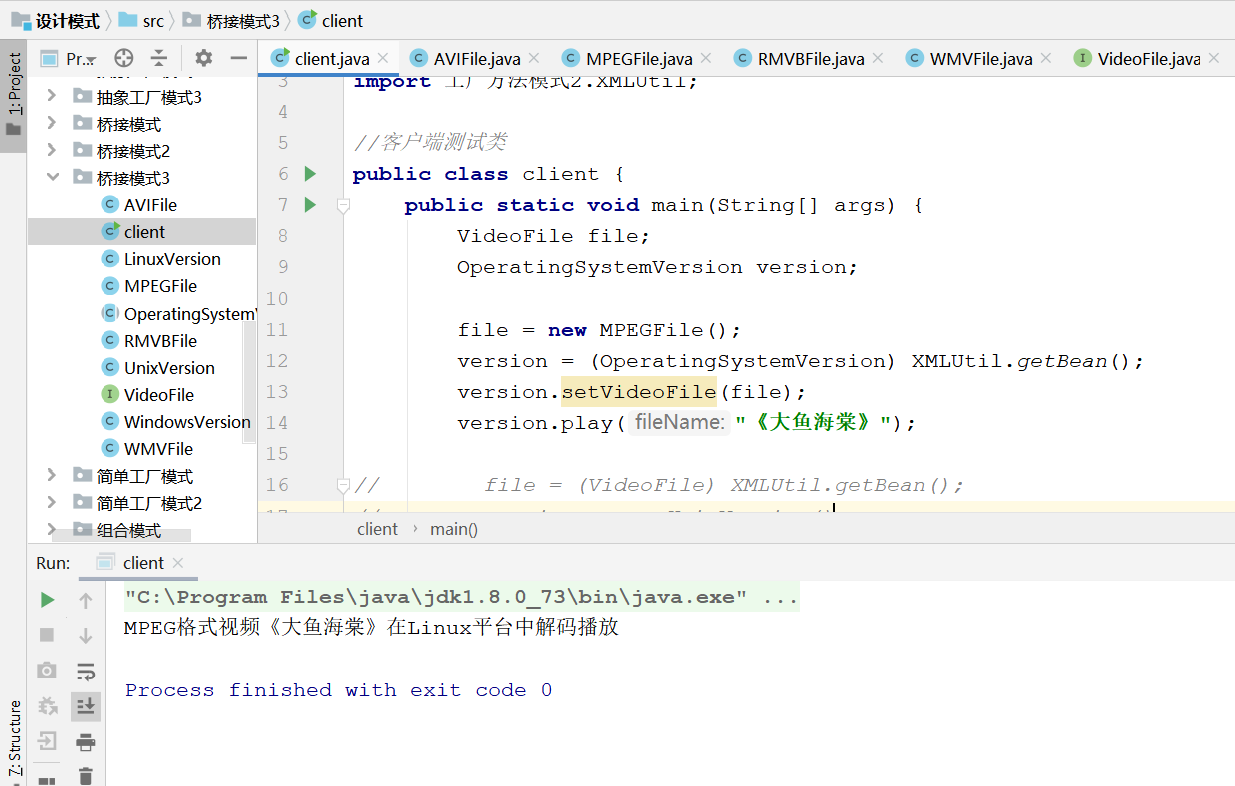
1. Config.xml文件配置：

XMLUtil工具类代码在实验一中已提出，这种不在提出，具体配置如下：

*<?***xml version="1.0"***?>*<**config**>  
<**className**>桥接模式3.LinuxVersion</**className**>  
</**config**>

1. 实验结果

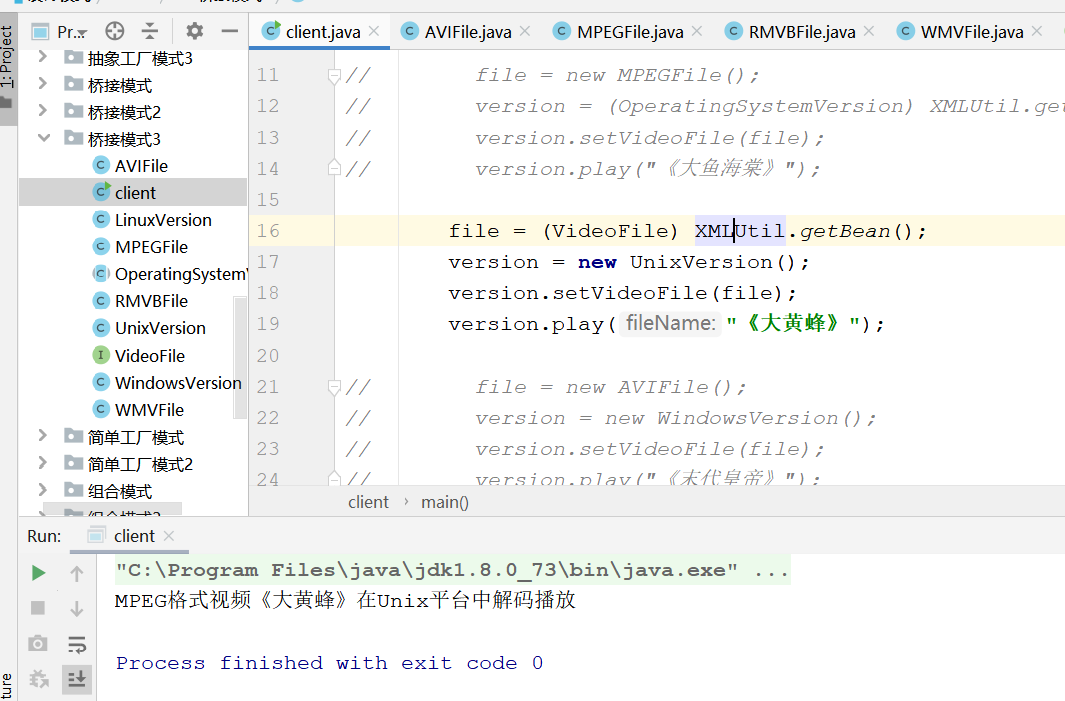
运行结果截图：



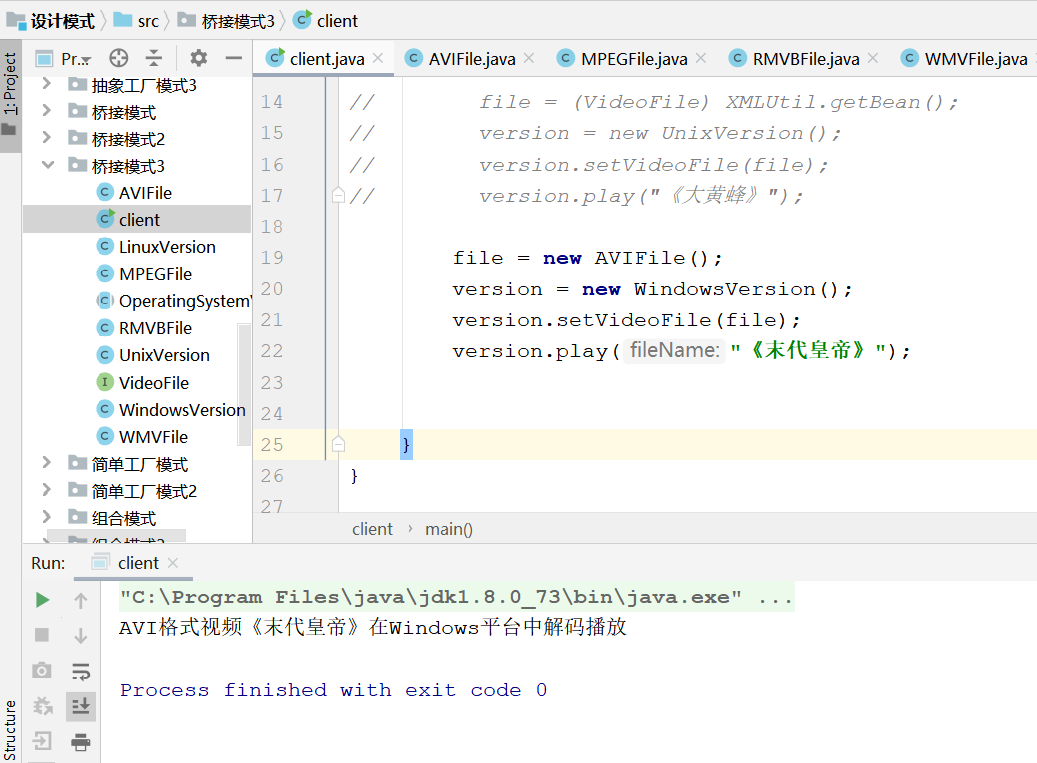
当VideoFile通过 XMLUtil.*getBean*()得到，此时config.xml配置如下：

*<?***xml version="1.0"***?>*<**config**>  
<**className**>桥接模式3.MPEGFile</**className**>  
</**config**>

运行结果截图为：



同样，也可以直接通过new方法创建对象并实例化。结果相同，展示结果如下：



1. 实验分析与总结

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。

桥接模式的优点有：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。

桥接模式的缺点有：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。

桥接模式的使用环境：①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；

②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；③一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；④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。