深圳大学
Shenzhen University

第10-1讲

桥接模式

软件体系结构与设计模式

Software Architecture & Design Pattern

深圳大学计算机与软件学院

主要内容

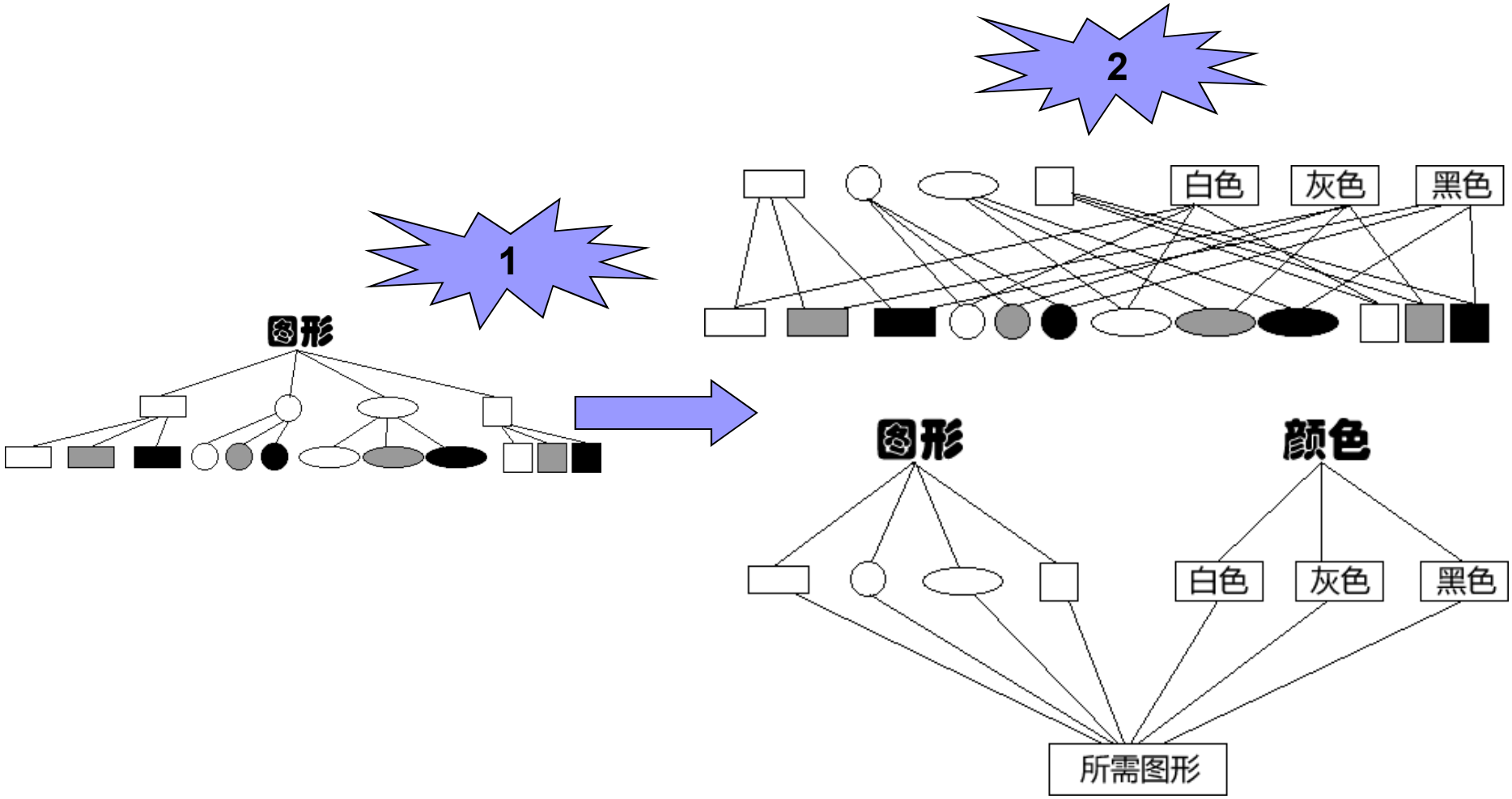
- 桥接模式
  - 桥接模式动机与定义
  - 桥接模式结构与分析
  - 桥接模式实例与解析
  - 桥接模式效果与应用

桥接模式

桥接模式动机

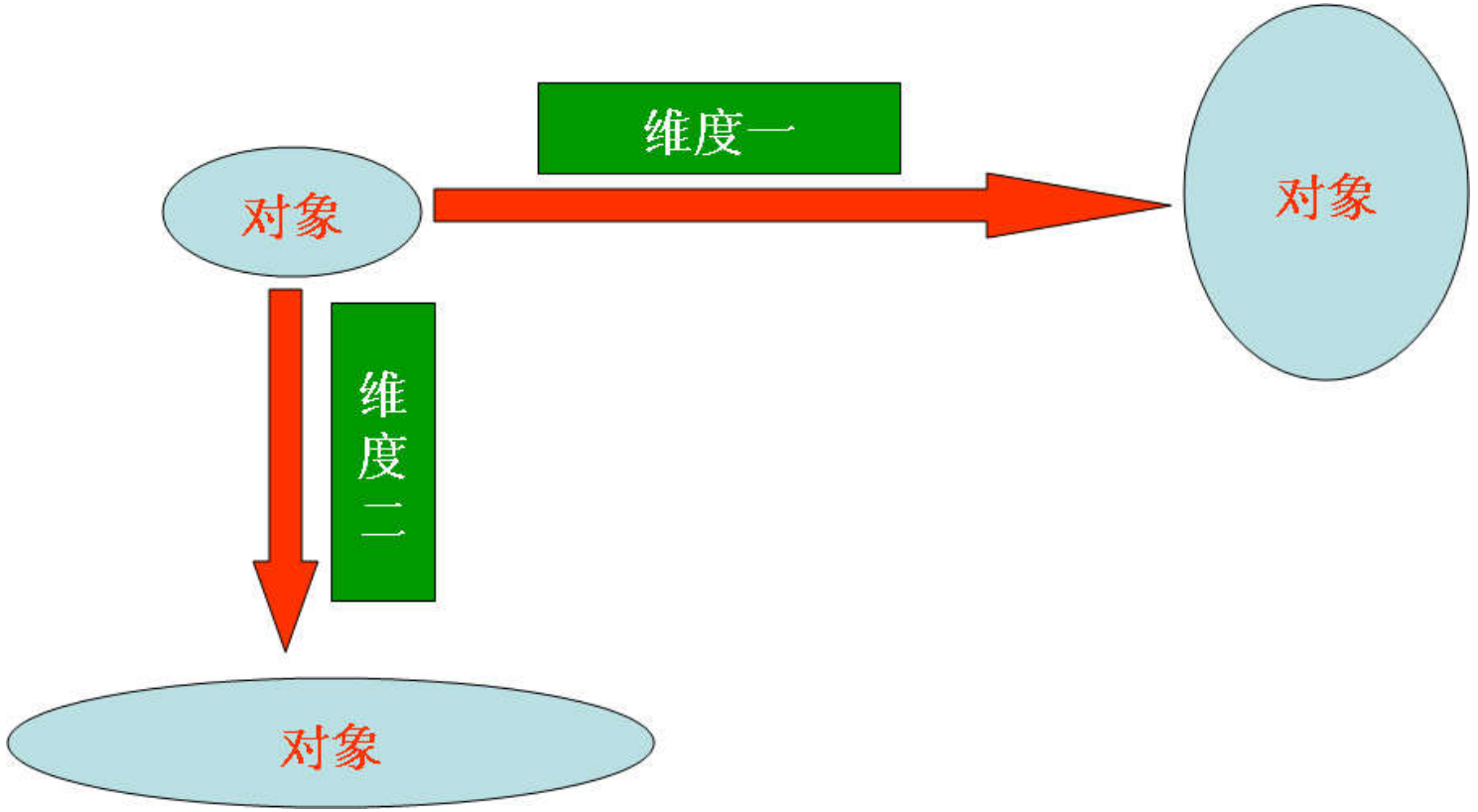
- 设想如果要绘制矩形、圆形、椭圆、正方形，我们至少需要4个形状类，但是如果绘制的图形需要具有不同的颜色，如红色、绿色、蓝色等，此时至少有如下两种设计方案：
  - 第一种设计方案是为每一种形状都提供一套各种颜色的版本。
  - 第二种设计方案是根据实际需要对形状和颜色进行组合。

桥接模式动机



桥接模式动机

- 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。



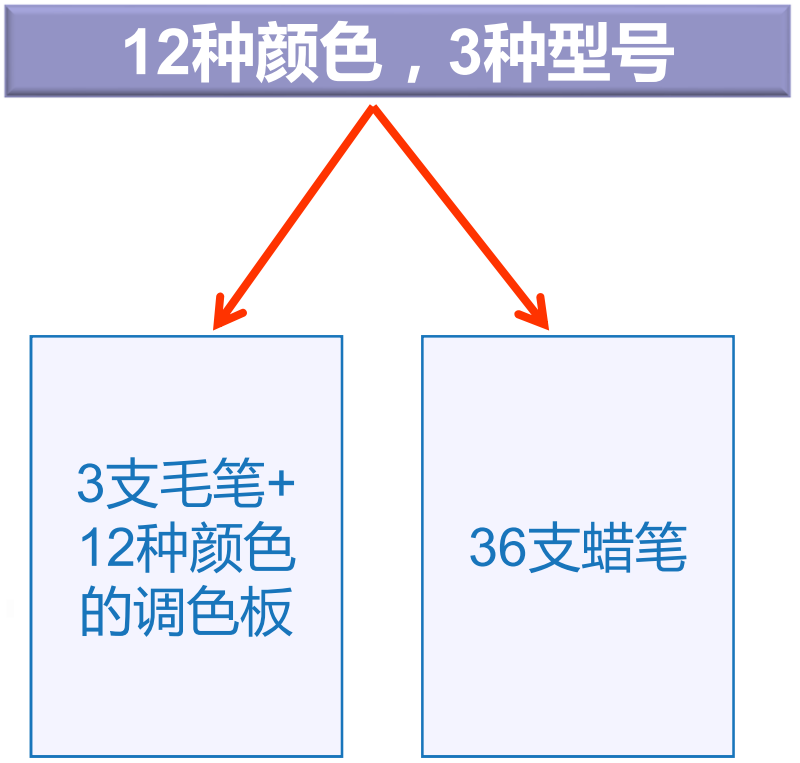
11.1 桥接模式

桥接模式动机

毛笔与蜡笔的“故事”

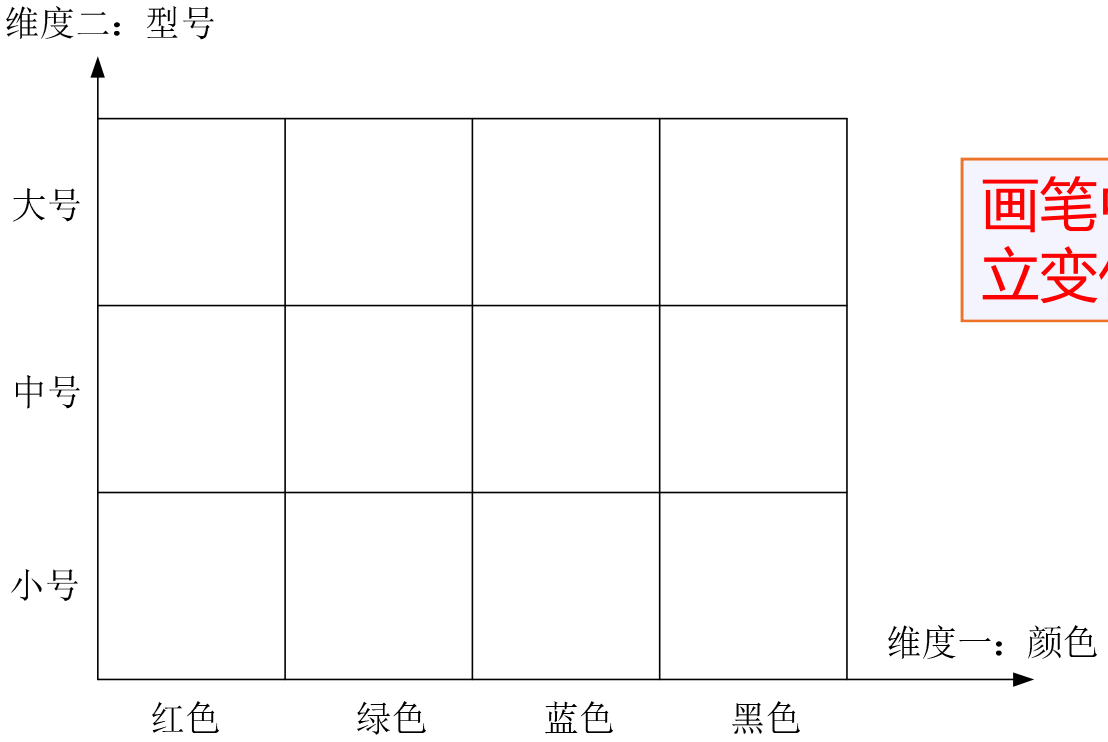
毛笔与调色板

不同型号的蜡笔



桥接模式动机

- **分析**
  - 蜡笔：颜色和型号两个不同的变化维度（即两个不同的变化原因）耦合在一起，无论是对颜色进行扩展还是对型号进行扩展都势必会影响另一个维度
  - 毛笔：颜色和型号实现了分离，增加新的颜色或者型号对另一方没有任何影响



画笔中存在的两个独立变化维度示意图

桥接模式动机

- **在软件开发中如何将多个变化维度分离？**

桥接模式定义

- 对象结构型模式

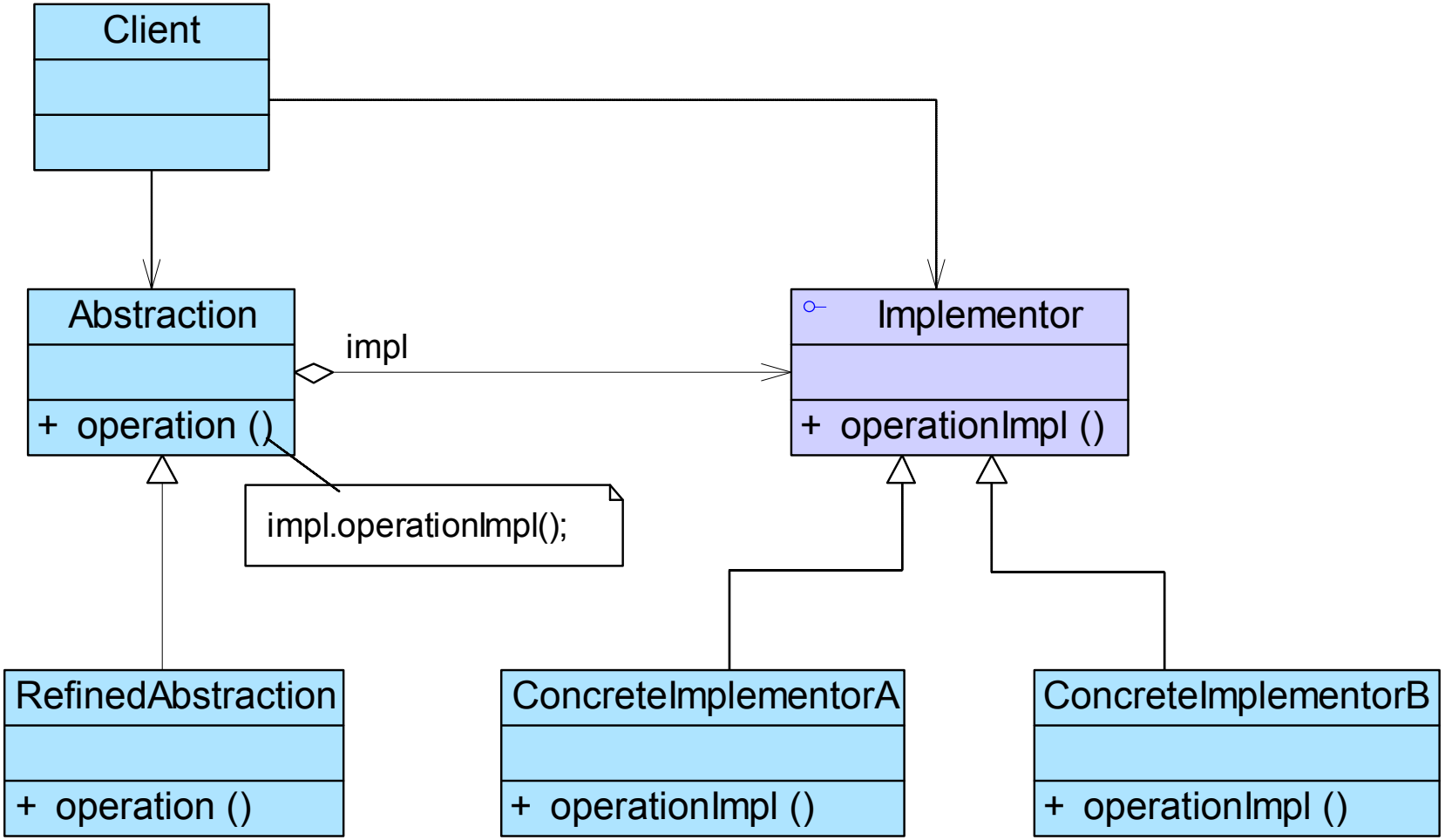
桥接模式：将抽象部分与它的实现部分解耦，使得两者都能够独立变化。

Bridge Pattern: Decouple an abstraction from its implementation so that the two can vary independently.

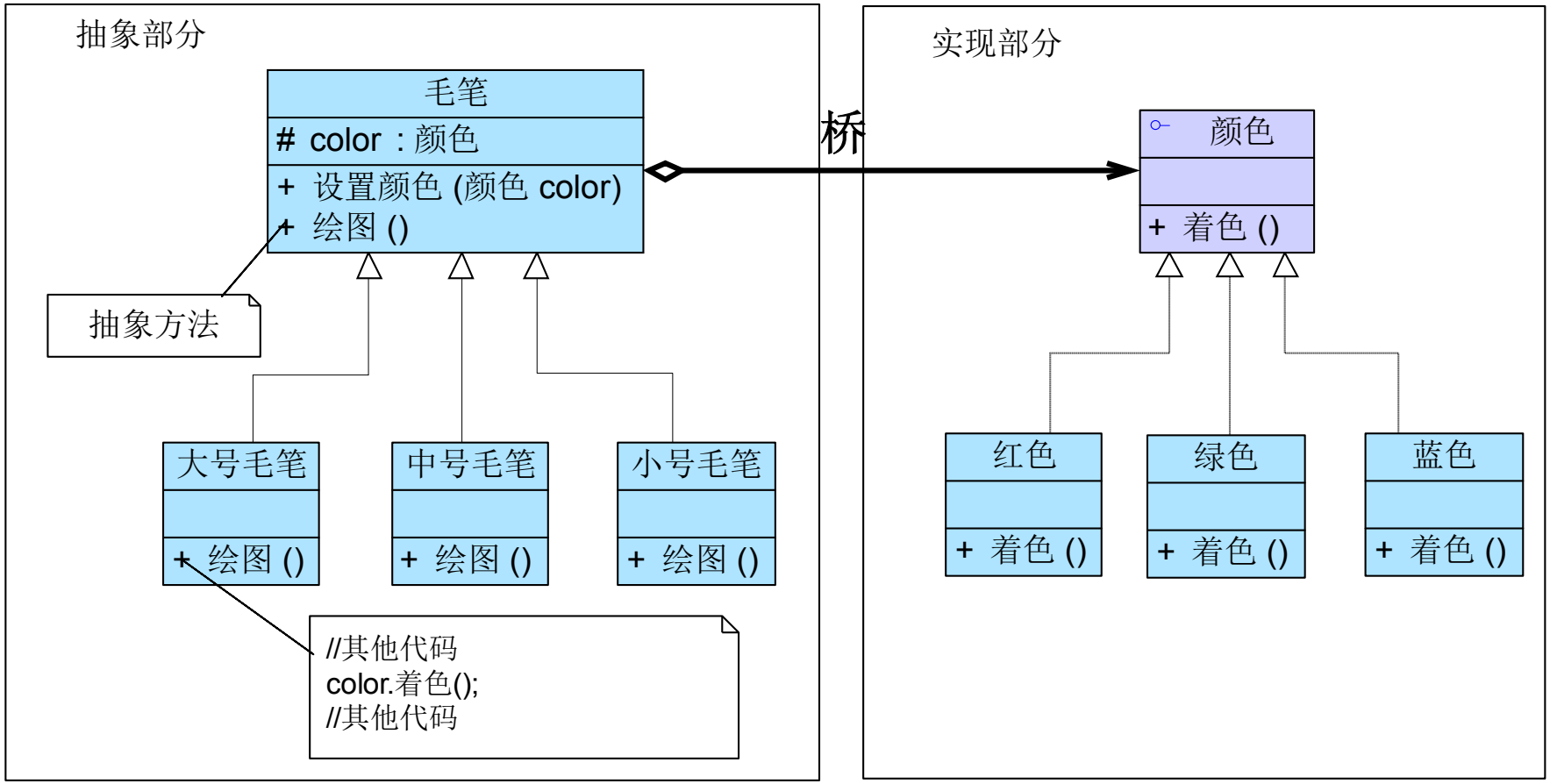
- 又被称为柄体(Handle and Body)模式或接口(Interface)模式
- 用抽象关联取代了传统的多层继承
- 将类之间的静态继承关系转换为动态的对象组合关系

桥接模式结构

- **桥接模式包含如下角色：**Abstraction：抽象类，RefinedAbstraction：扩充抽象类，Implementor：实现类接口，ConcreteImplementor：具体实现类



桥接模式结构分析



毛笔结构示意图

桥接模式结构分析

- 典型的实现类接口代码：

```
public interface Implementor
{
    public void operationImpl();
}
```

桥接模式结构分析

- 典型的抽象类代码：

```
public abstract class Abstraction
{
    protected Implementor impl;

    public void setImpl(Implementor impl)
    {
        this.impl=impl;
    }

    public abstract void operation();
}
```

桥接模式结构分析

- 典型的扩充抽象类代码：

```
public class RefinedAbstraction extends Abstraction
{
    public void operation()
    {
        //代码
        impl.operationImpl();
        //代码
    }
}
```

桥接模式实例

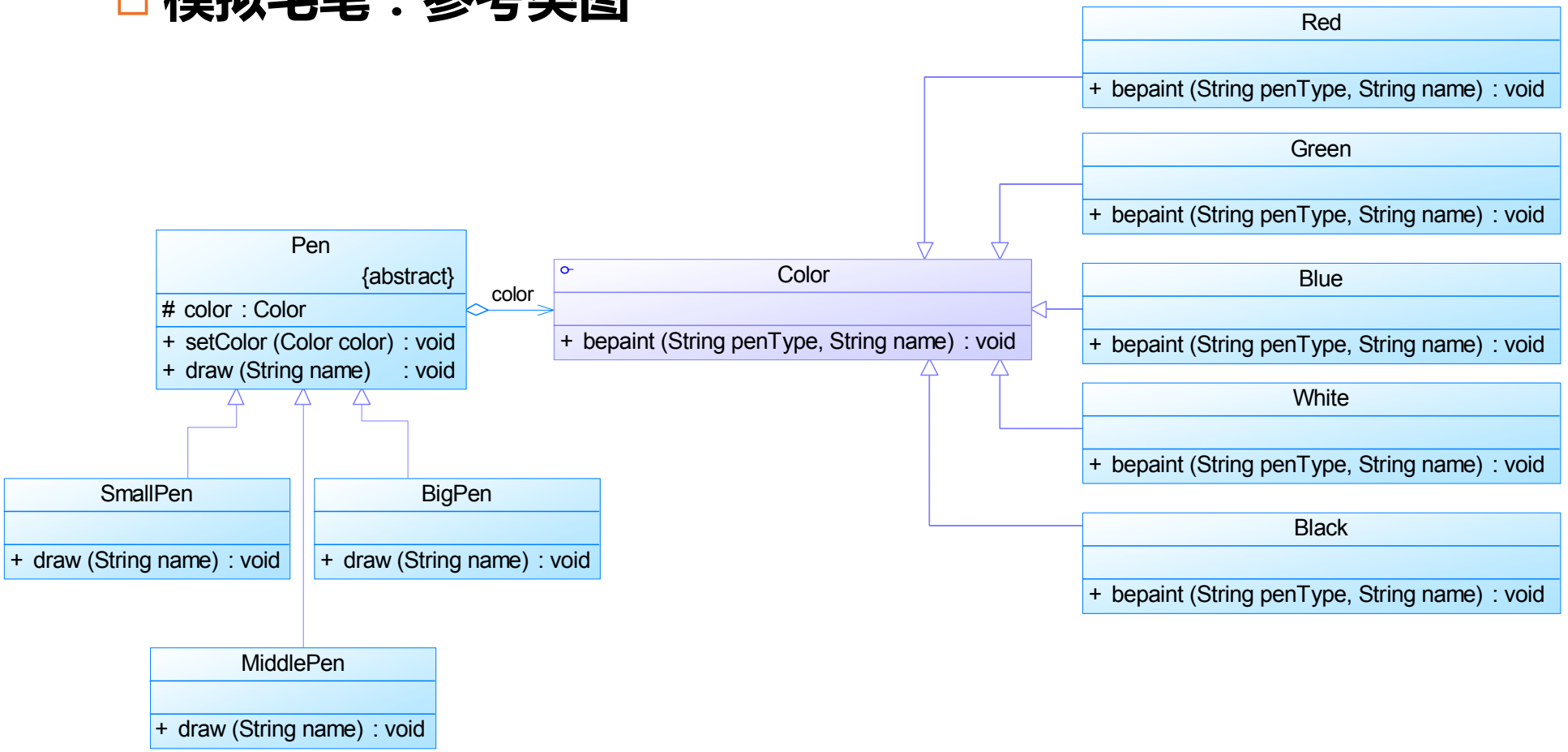
模拟毛笔：实例说明

- 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。

桥接模式实例与解析

桥接模式实例

模拟毛笔：参考类图



桥接模式实例与解析

- **桥接模式实例**
  - **模拟毛笔：参考代码**
- DesignPatterns之bridge包

Pen.java

```
1  package bridge;
2
3  public abstract class Pen
4  {
5      protected Color color;
6      public void setColor(Color color)
7      {
8          this.color=color;
9      }
10     public abstract void draw(String name);
11 }
```

SmallPen.java

```java
package bridge;

public class SmallPen extends Pen
{
    public void draw(String name)
    {
        String penType="小号毛笔绘制";
        this.color.bepaint(penType,name);
    }
}
```

BigPen.java

```java
package bridge;

public class BigPen extends Pen
{
    public void draw(String name)
    {
        String penType="大号毛笔绘制";
        this.color.bepaint(penType,name);
    }
}
```

MiddlePen.java

```java
package bridge;

public class MiddlePen extends Pen
{
    public void draw(String name)
    {
        String penType="中号毛笔绘制";
        this.color.bepaint(penType,name);
    }
}
```

Color.java

```java
package bridge;

public interface Color
{
    void bepaint(String penType,String name);
}
```

Red.java

```java
public class Red implements Color
{
    public void bepaint(String penType,String name)
    {
        System.out.println(penType + "红色的"+ name + ".");
    }
}
```

Green.java

```java
public class Green implements Color
{
    public void bepaint(String penType,String name)
    {
        System.out.println(penType + "绿色的"+ name + ".");
    }
}
```

Blue.java

```
3 public class Blue implements Color
4 {
5     public void bepaint(String penType,String name)
6     {
7         System.out.println(penType + "蓝色的"+ name + ".");
8     }
9 }
```

White.java

```
3 public class White implements Color
4 {
5     public void bepaint(String penType,String name)
6     {
7         System.out.println(penType + "白色的"+ name + ".");
8     }
9 }
```

Black.java

```
3 public class Black implements Color
4 {
5     public void bepaint(String penType,String name)
6     {
7         System.out.println(penType + "黑色的"+ name + ".");
8     }
9 }
```

XMLUtilPen.java

```java
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.SAXException;
import java.io.*;
public class XMLUtilPen
{
//该方法用于从XML配置文件中提取具体类类名，并返回一个实例对象
    public static Object getBean(String args)
    {
        try
        {
            //创建文档对象
            DocumentBuilderFactory dFactory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = dFactory.newDocumentBuilder();
            Document doc;
            doc = builder.parse(new File("BridgeconfigPen.xml"));
            NodeList nl=null;
            Node classNode=null;
            String cName=null;
            nl = doc.getElementsByTagName("className");

            if(args.equals("color"))
            {
                //获取包含类名的文本节点
                classNode=nl.item(0).getFirstChild();

            }
```

XMLUtilPen.java

```
30              else if(args.equals("pen"))
31              {
32                  //获取包含类名的文本节点
33                   classNode=nl.item(1).getFirstChild();
34              }
35
36               cName=classNode.getNodeValue();
37               //通过类名生成实例对象并将其返回
38               Class c=Class.forName(cName);
39               Object obj=c.newInstance();
40               return obj;
41             }
42              catch(Exception e)
43              {
44                  e.printStackTrace();
45                  return null;
46              }
47          }
48  }
```

BridgeconfigPen.xml

```
1  <?xml version="1.0"?>
2  <config>
3      <className>bridge.Blue</className>
4      <className>bridge.SmallPen</className>
5  </config>
```

Client.java

```java
package bridge;

public class Client
{
    public static void main(String a[])
    {
        Color color;
        Pen pen;

        color=(Color)XMLUtilPen.getBean("color");
        pen=(Pen)XMLUtilPen.getBean("pen");

        pen.setColor(color);
        pen.draw("鲜花");
    }
}
```

桥接模式实例

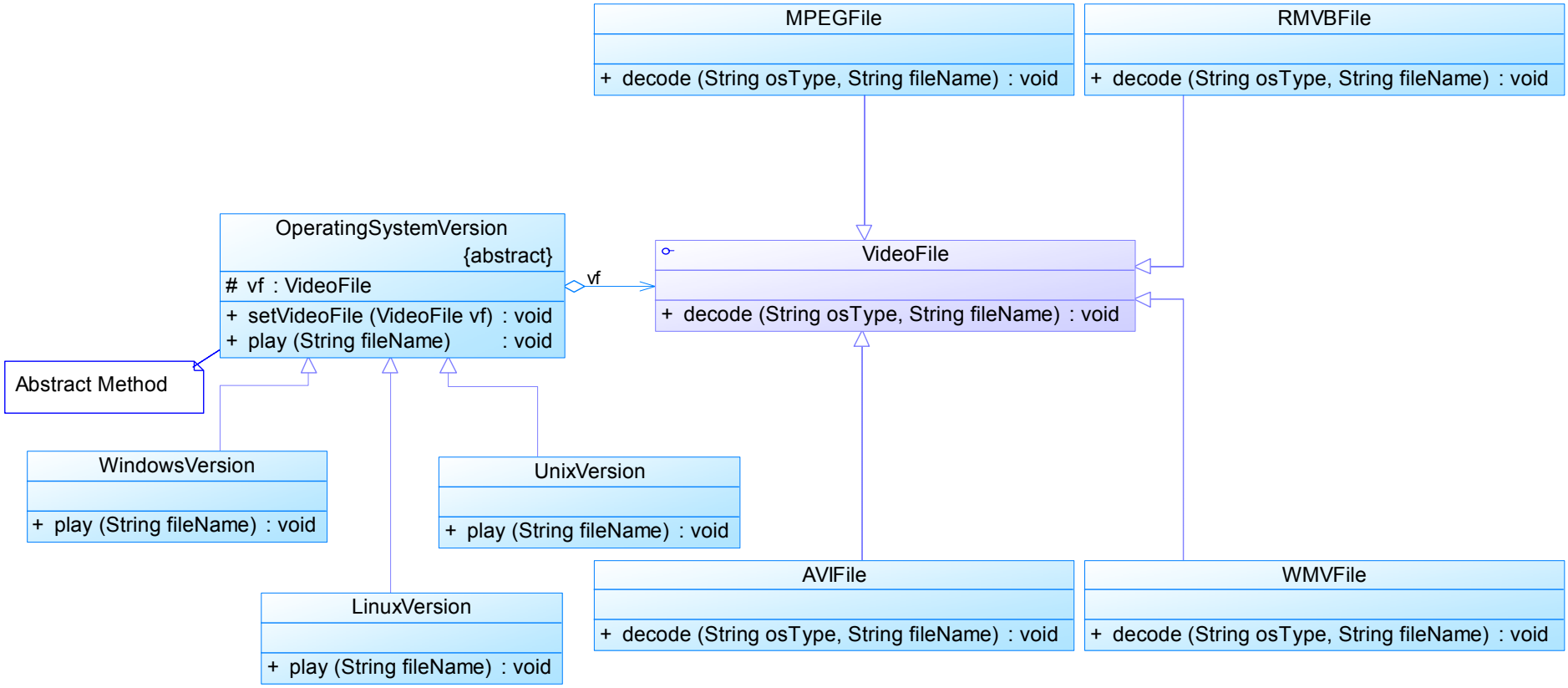
跨平台视频播放器：实例说明

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。

桥接模式实例与解析

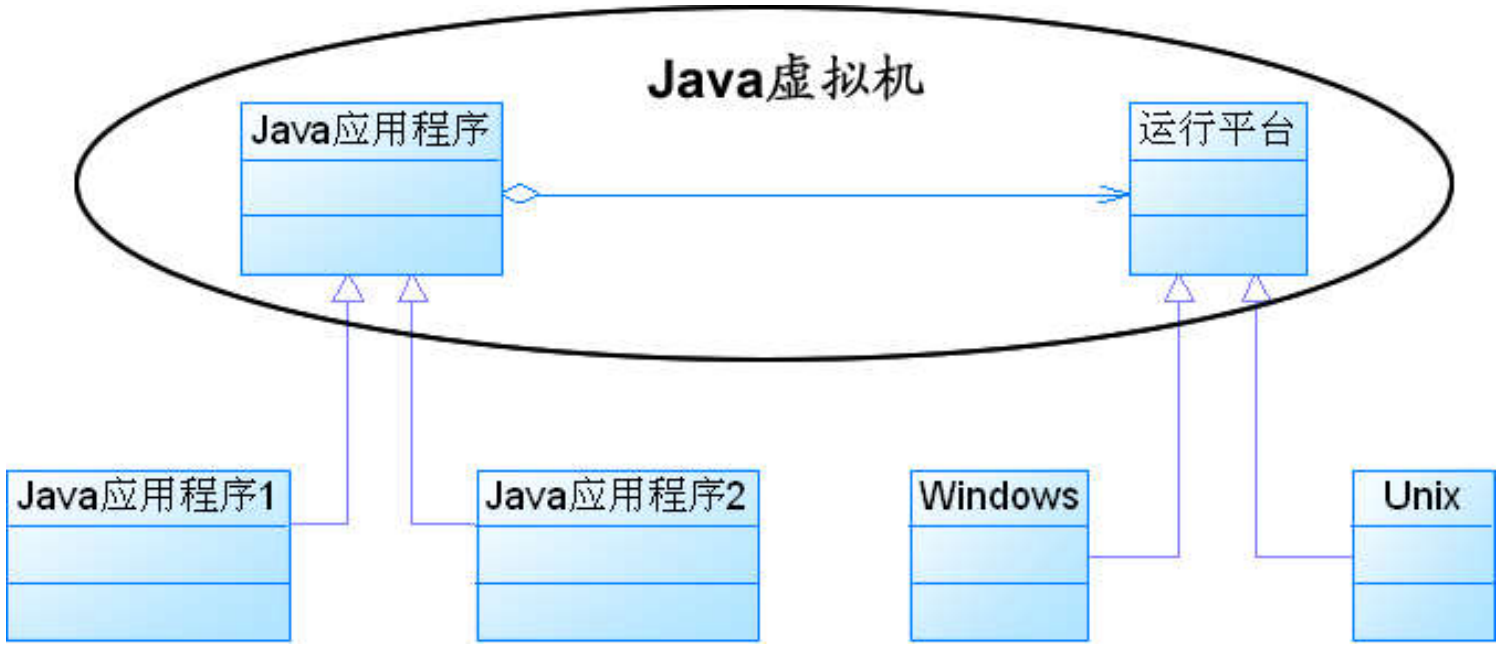
桥接模式实例

跨平台视频播放器：参考类图



桥接模式应用

- Java语言通过Java虚拟机实现了平台的无关性。



Java虚拟机示意图

桥接模式应用

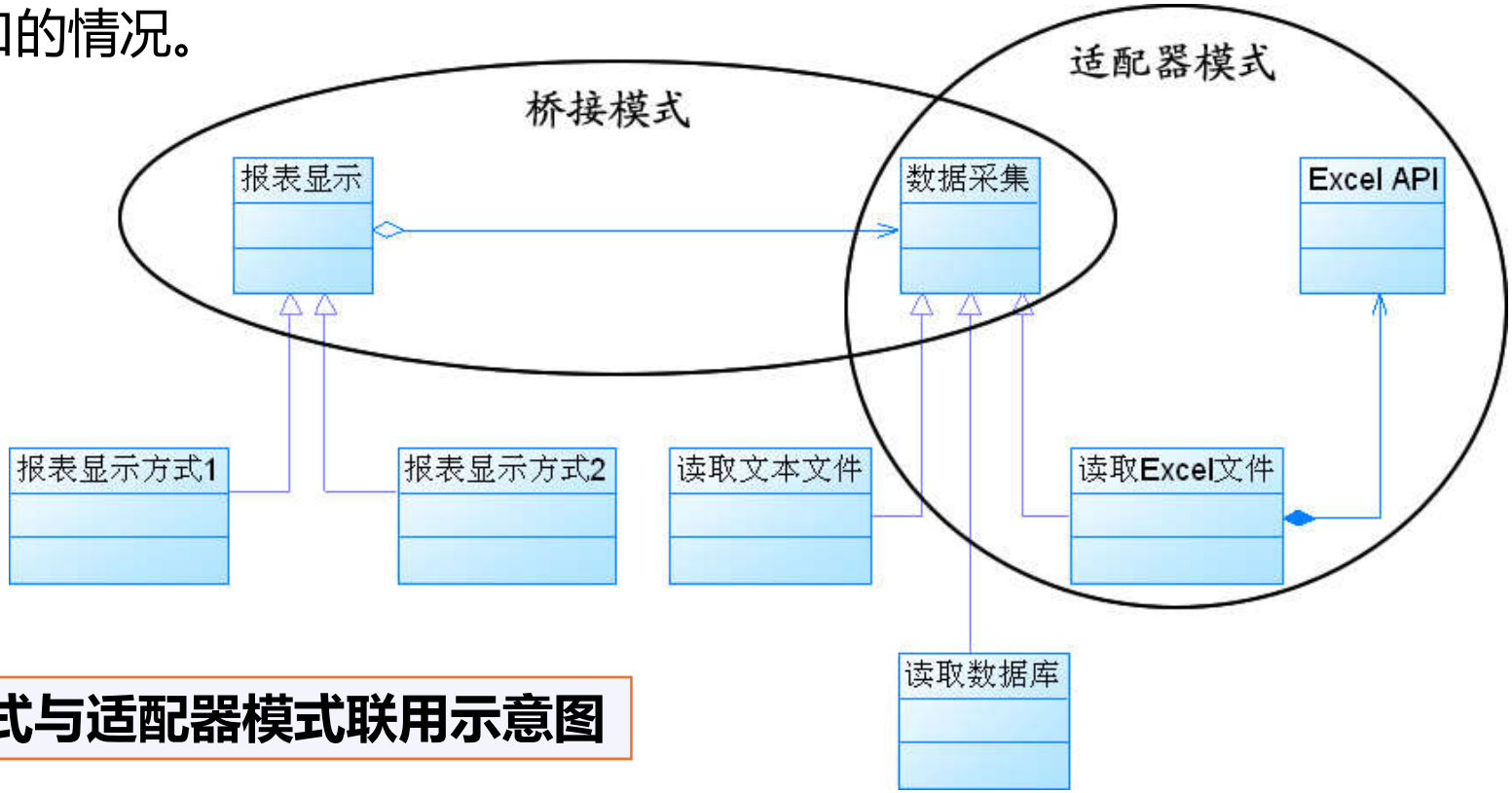
- 一个 Java桌面软件总是带有所在操作系统的视感(LookAndFeel)，如果一个Java软件是在Unix系统上开发的，那么开发人员看到的是Motif用户界面的视感；在Windows上面使用这个系统的用户看到的是Windows用户界面的视感；而一个在Macintosh上面使用的用户看到的则是Macintosh用户界面的视感，
- Java语言是通过所谓的Peer架构做到这一点的。
- Java为AWT中的每一个GUI构件都提供了一个Peer构件，在AWT中的Peer架构就使用了桥接模式。

桥接模式应用

- JDBC驱动程序也是桥接模式的应用之一。
- 使用JDBC驱动程序的应用系统就是抽象角色，而所使用的数据库是实现角色。
- 一个JDBC驱动程序可以动态地将一个特定类型的数据库与一个Java应用程序绑定在一起，从而实现抽象角色与实现角色的动态耦合。

桥接模式与适配器模式的联用

- **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
- **适配器模式：**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，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。



桥接模式与适配器模式联用示意图

桥接模式效果与应用

- **桥接模式优点：**
  - 分离抽象接口及其实现部分
  - 可以取代多层继承方案，极大地减少了子类的个数
  - 提高了系统的可扩展性，在两个变化维度中任意扩展一个维度，不需要修改原有系统，符合开闭原则

桥接模式效果与应用

- **桥接模式缺点：**
  - 会增加系统的理解与设计难度，由于关联关系建立在抽象层，要求开发者一开始就要针对抽象层进行设计与编程
  - 正确识别出系统中两个独立变化的维度并不是一件容易的事情

桥接模式效果与应用

- **在以下情况下可以使用桥接模式：**
  - 需要在抽象化和具体化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关系
  - 抽象部分和实现部分可以以继承的方式独立扩展而互不影响
  - 一个类存在两个（或多个）独立变化的维度，且这两个（或多个）维度都需要独立地进行扩展
  - 不希望使用继承或因为多层继承导致系统类的个数急剧增加的系统

课后思考

- 如果系统中存在两个以上的变化维度，是否可以使用桥接模式进行处理？如果可以，系统该如何设计？

谢谢