# 桥接模式简介

## 概述

1. 如果系统中的某个类存在两个独立变化的维度，通过桥接模式可以将这两个维度分离出来，使两者可以独立扩展。
2. 桥接模式可以处理多层继承存在的问题，用抽象关联来取代传统的多层继承，将类之间的静态继承关系转换为动态的对象组合关系，使得系统更加灵活，并易于扩展，同时有效地控制了系统中类的个数。
3. 在桥接模式中将两个独立变化的维度设计为两个独立的继承等级结构，而不是将二者耦合在一起形成多层继承结构。桥接模式在抽象层建立起一个抽象关联，该关联关系类似一条连接两个独立继承结构的桥，故名桥接模式。
4. 桥接模式的定义如下：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，任意扩充任一维度而对另一维度不会造成任何影响。
5. 桥接模式（Bridge Pattern）是一种对象结构型模式，它又被称为接口模式；通过桥接模式可以处理多层继承的问题，用抽象关系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，使得系统更加灵活，并易于扩展，同时可以有效控制系统中类的个数。

## 结构

桥接模式结构如下：



桥接模式包含4个角色。

1. Abstraction（抽象类）：用于定义抽象部分的接口，通常将其定义为抽象类而不是接口；在该类中定义了一个Implementor（实现类接口）类型的对象并维护该对象（它的访问权限为protected），它与Implementor之间具有关联关系；它既可以包含抽象业务方法（抽象方法），也可以包含具体业务方法（普通方法）。
2. RefinedAbstraction（扩充抽象类）：是Abstraction（抽象类）的子类，对抽象类进行扩充，通常情况下定义为具体类而不是抽象类，实现了在Abstraction中声明的抽象业务方法；在RefinedAbstraction中可以调用Implementor中定义的业务方法。
3. Implementor（实现类接口）：用于定义实现部分的接口；这个接口不一定要与Abstraction的接口完全一致，一般而言，Implementor接口仅提供基本操作，而Abstraction定义的接口可能会做更多更复杂的操作。Implementor接口对这些基本操作进行了声明（定义抽象方法），而具体实现将交给其子类（让子类实现它并实现其中的抽象方法）；通过关联关系，在Abstraction中不仅拥有它自己的方法，还可以调用到Implementor中定义的方法，使用关联关系来替代继承关系。
4. ConcreteImplementor（具体实现类）：它实现了Implementor接口，是Implementor的子类；在程序运行时ConcreteImplementor对象将替换其父类对象，提供给抽象类具体的业务操作方法。

# 实现

1. 在使用桥接模式时首先应该识别出一个类所具有的两个独立变化的维度，将它们设计为两个独立的继承等级结构，为两个维度都提供抽象层，并建立抽象耦合。
2. 通常情况下，将具有两个独立变化维度的类的一些普通业务方法和与之关系最密切的维度设计为“抽象类”层次结构（抽象部分），而将另一个维度设计为“实现类”层次结构（实现部分）。

## 实例

实例说明：某软件公司要开发一个跨平台的图像浏览系统，要求该系统能够显示BMP、JPG、GIF、PNG等多种格式的文件，并且能够在Windows、Linux、UNIX等多个操作系统上运行。系统首先将各种格式的文件解析为像素矩阵（Matrix），然后将像素矩阵显示在屏幕上，在不同的操作系统中可以调用不同的绘制函数来绘制像素矩阵。系统需具有较好的扩展性，以便在将来支持新的文件格式和操作系统。

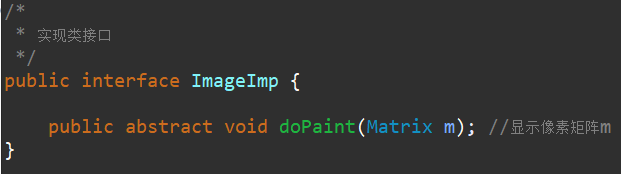


使用桥接模式设计该跨平台图像浏览系统：Image充当抽象类，其子类JPGImage、PNGImage、BMPImage、GIFImage充当扩充抽象类；ImageImp充当实现类接口，其子类WindowsImp、LinuxImp、UnixImp充当具体实现类；

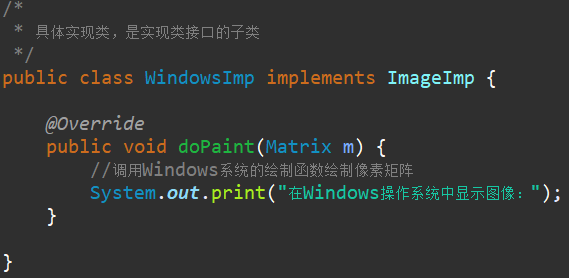
1. Matrix：像素矩阵类，它是一个辅助类，各种格式的图像文件最终对被转化为像素矩阵，不同的操作系统提供不同的方式显示像素矩阵。



1. ImageImp：抽象操作系统实现类，充当实现类接口



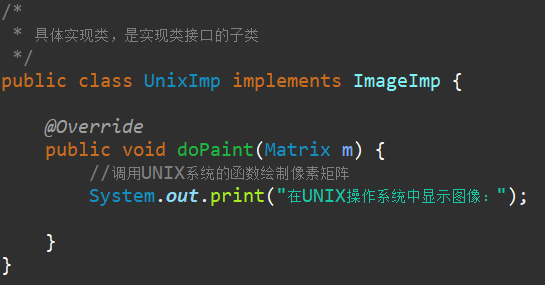
1. WindowsImp：Windows操作系统实现类，充当具体实现类



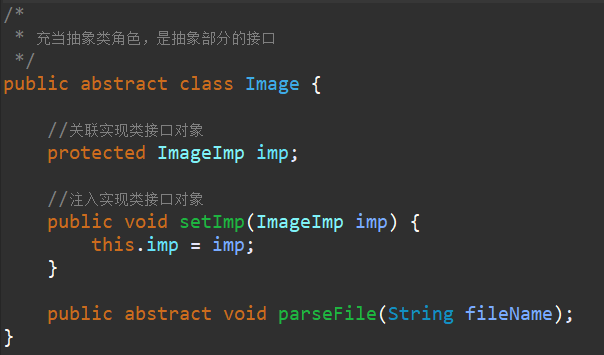
1. LinuxImp：Linux操作系统实现类，充当具体实现类



1. UnixImpl：UNIX操作系统实现类，充当具体实现类



1. Image：抽象图像类，充当抽象类；在该类内部定义实现类接口对象作为属性存在，将抽象类和实现类接口进行关联，并对其进行维护，且它的访问权限是protected（这是桥接模式的核心细节之一）。



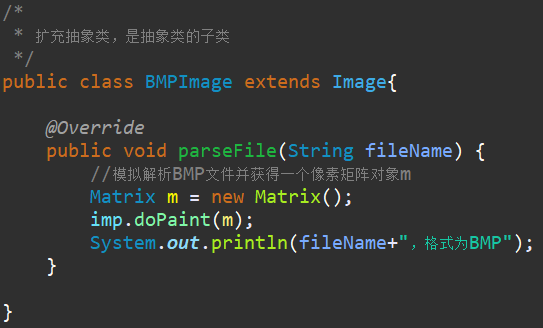
1. JPGImage：JPG格式图像类，充当扩充抽象类



1. PNGImage：PNG格式图像类，充当扩充抽象类



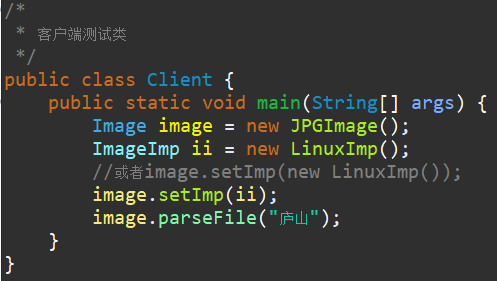
1. BMPImage：BMP格式图像类，充当扩充抽象类



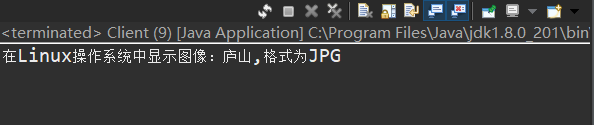
1. GIFImage：GIF格式图像类，充当扩充抽象类



1. Client：客户端测试类



运行客户端查看结果：



## 与适配器模式联用

在软件开发中，适配器模式通常与桥接模式联合使用。

1. 适配器模式可以解决两个已有接口间的不兼容的问题，在这种情况下被适配的类（即适配者类）往往是一个黑盒子（无法查看源代码或只知道每个方法的功能），有时候不想或者不能改变这个被适配的类，也不能控制其扩展。适配器模式通常用于现有系统与第三方产品功能的集成，采用增加适配器的方式将第三方类集成到系统中。而桥接模式则不同，用户可以通过接口继承或类继承的方式对系统进行扩展。
2. 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考虑适配器模式，特别是那些设计大量第三方应用接口的情况。

# 优缺点及适用环境

## 优点

桥接模式的优点主要如下：

1. 分离抽象接口及其实现部分。桥接模式使用“对象间的关联关系”解耦了抽象和实现之间固有的绑定关系，使得抽象和实现可以沿着各自的维度来变化。所谓抽象和实现沿着各自维度的变化，也就是说抽象和实现不再同一个继承层次结构中，而是“子类化”它们，是它们各自具有自己的子类，以便任意组合子类，从而获得多维度组合对象。
2. 在很多情况下，桥接模式可以取代多层继承方案，多层继承方案违背了单一职责原则，复用性差，并且类的个数非常多，桥接模式是比多层继承方案更好的解决方法，它极大地减少了子类的个数。
3. 桥接模式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都不需要修改原有系统，符合开闭原则。

## 缺点

桥接模式的缺点主要如下：

1. 桥接模式的使用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针对抽象层进行设计与编程。
2. 桥接模式要求正确地识别系统中的两个独立变化的维度，因此其使用范围具有一定的局限性，如何正确识别两个独立维度也需要一定的经验积累。

## 适用环境

在以下情况下可以考虑使用桥接模式：

1. 如果一个系统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，通过桥接模式可以使它们在抽象层建立一个关联关系。
2. 抽象部分和实现部分可以用继承的方式独立扩展而互不影响，在程序运行时（使用时）可以动态地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（或多个）独立变化的维度，且这两个（或多个）维度都需要独立进行扩展。
4. 对于那些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，桥接模式尤为适用。