**桥接（Bridge**）是用于**把抽象化与实现化解耦**，**使得二者可以独立变化**。这种类型的设计模式属于结构型模式，它通过提供抽象化和实现化之间的桥接结构，来实现二者的解耦。

这种模式涉及到一个作为桥接的接口，使得实体类的功能独立于接口实现类。这两种类型的类可被结构化改变而互不影响。

实例：使用相同的抽象类方法但是不同的桥接实现类，来画出不同颜色的圆。

**介绍**

**意图： 将抽象部分与实现部分分离，使它们都可以独立的变化。**

**主要解决： 在有多种可能会变化的情况下，用继承会造成类爆炸问题，扩展起来不灵活。**

**何时使用：** 实现系统可能有多个角度分类，每一种角度都可能变化。

**如何解决：** 把这种多角度分类分离出来，让它们独立变化，减少它们之间耦合。

**关键代码：** 抽象类依赖实现类。

**应用实例：** 墙上的开关，可以看到的开关是抽象的，不用管里面具体怎么实现的。

**优点：**  1、抽象和实现的分离。 2、优秀的扩展能力。 3、实现细节对客户透明。

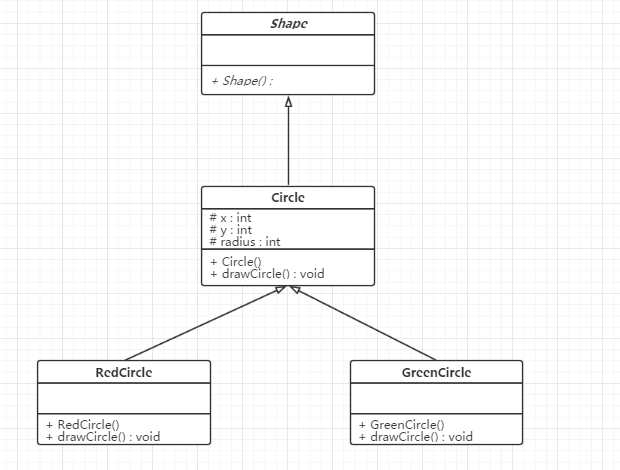
**缺点：** 桥接模式的引入会增加系统的理解与设计难度，由于聚合关联关系建立在抽象层，要求开发者针对抽象进行设计与编程。

**使用场景：**  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(使用“对象间的组合关系”解耦了抽象和实现之间固有的绑定关系)**。 2、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。 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。

**注意事项：**对于两个独立变化的维度，使用桥接模式再适合不过了。

**桥接模式与装饰的区别:  
装饰模式:**  
      这两个模式在一定程度上都是为了减少子类的数目，避免出现复杂的继承关系。但是它们解决的方法却各有不同，**装饰模式把子类中比基类中多出来的部分放到单独的类里面，以适应新功能增加的需要**，当我们把描述新功能的类封装到基类的对象里面时，就得到了所需要的子类对象，这些描述新功能的类通过组合可以实现很多的功能组合 .  
**桥接模式：**  
          **桥接模式则把原来的基类的实现化细节抽象出来(draw())，在构造到一个实现化的结构中(DrawAPI)，然后再把原来的基类(RedCircle,GreenCircle)改造成一个抽象化的等级结构(DrawAPI<-RedCircle/GreenCircle)，这样就可以实现系统在多个维度上的独立变化 。**

**原始的设计：**



使用桥接模式后的设计见桥接模式.png