# 基本概念

桥接模式是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(Interfce)模式。

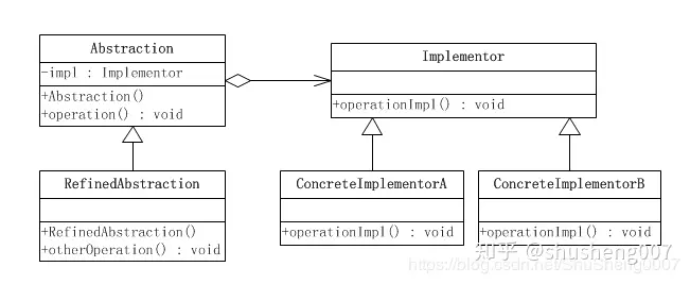
# 使用场景

1、“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，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。

2、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。

3、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。

# UML图



## 角色

抽象化(Abstraction)角色：抽象化给出的定义，并保存一个对实现化对象的引用。

修正抽象化(RefinedAbstraction)角色：扩展抽象化角色，改变和修正父类对抽象化的定义。

实现化(Implementor)角色：这个角色给出实现化角色的接口，但不给出具体的实现。必须指出的是，这个接口不一定和抽象化角色的接口定义相同，实际上，这两个接口可以非常不一样。实现化角色应当只给出底层操作，而抽象化角色应当只给出基于底层操作的更高一层的操作。

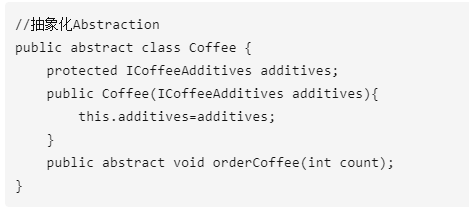
具体实现化(ConcreteImplementor)角色：这个角色给出实现化角色接口的具体实现。

抽象化角色就像是一个水杯的手柄，而实现化角色和具体实现化角色就像是水杯的杯身。手柄控制杯身，这就是此模式别名“柄体”的来源。

# 代码示例

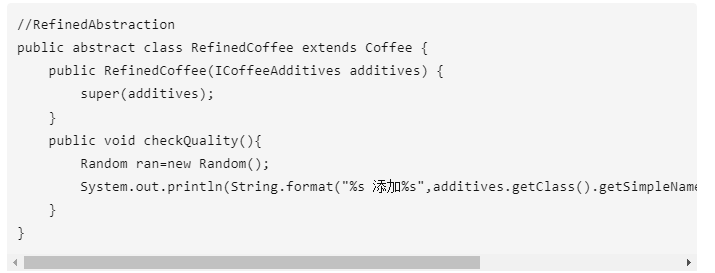
将咖啡的容量作为抽象化Abstraction，而咖啡口味为实现化Implementor

## 1、创建抽象化部分

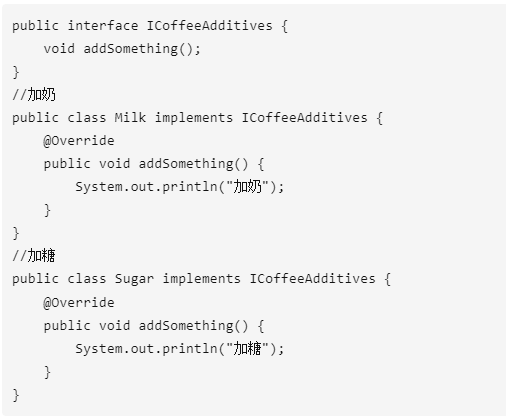


我们可以看到，Coffee持有了ICoffeeAdditives 引用，ICoffeeAdditives 的实例是通过构造函数注入的，这个过程就是我们所说的桥接过程。我们通过这个引用就可以调用ICoffeeAdditives的方法，进而将Coffee的行为与ICoffeeAdditives的行为通过orderCoffee()方法而组合起来。

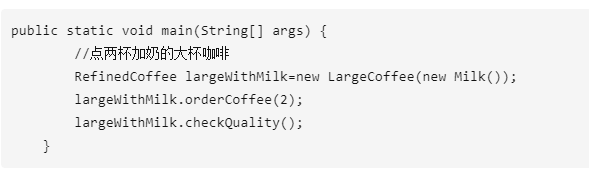
下面是一个对抽象化修正的一个类,里面增加了一个品控的方法



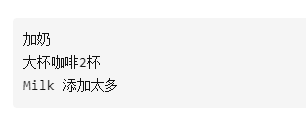
## 2、创建实现化部分



## 3、客户端调用



输出结果：



通过使用桥接模式，就使得咖啡的容量和口味这两个维度可以独立变化，互不干扰。

# 优点

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在同一个继承层次结构中，而是“子类化”它们，使它们各自都具有自己的子类，以便任何组合子类，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”，复用性较差，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

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，如何正确识别两个独立维度也需要一定的经验积累。