## 1.设计模式：Builder

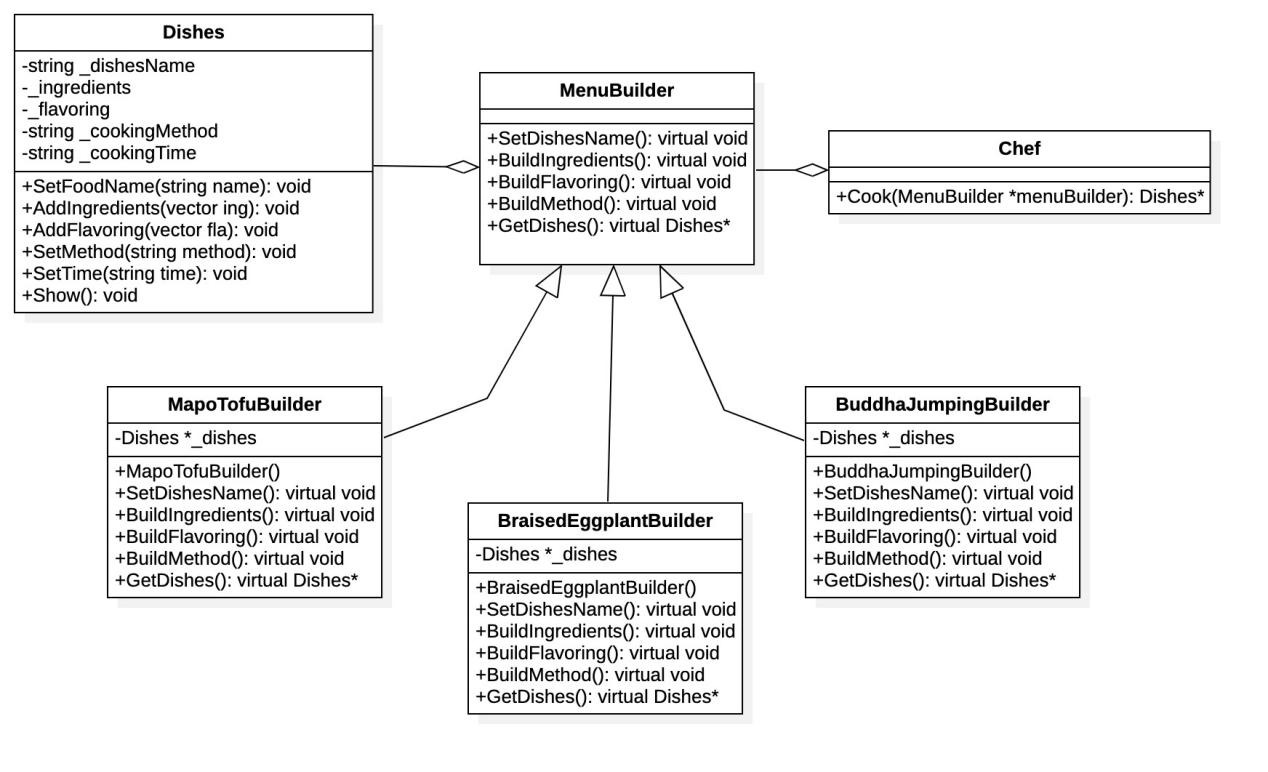
1. 实现API描述：

Builder设计模式是指:在一类复杂的产品的实现中，将他的制造工序中的每一步一种具体产品中分离出来，使得按照相同的制作步骤可以制造出不同的产品。在本场景中，动物运动会的食堂大厨在为运动员们准备食物时，烹饪的步骤是相同的，但是却要制作出不同类型的菜肴，因此用builder来实现：

在MenuBuilder中规定好制作一道菜肴的步骤，再由具体的子类如MapoTofuBuilder来制定每一个步骤具体的实现，最终将一个菜肴builder（菜谱）传给Cook类即可具体实现菜肴的制作，将制作好的菜肴提取出来。

本设计模式的测试接口写在了一个名为Builder的类中，只需实例化一个Builder类型的对象即可调用其公共函数Test（）即可测试。

1. 类图：



## 设计模式：Bridge

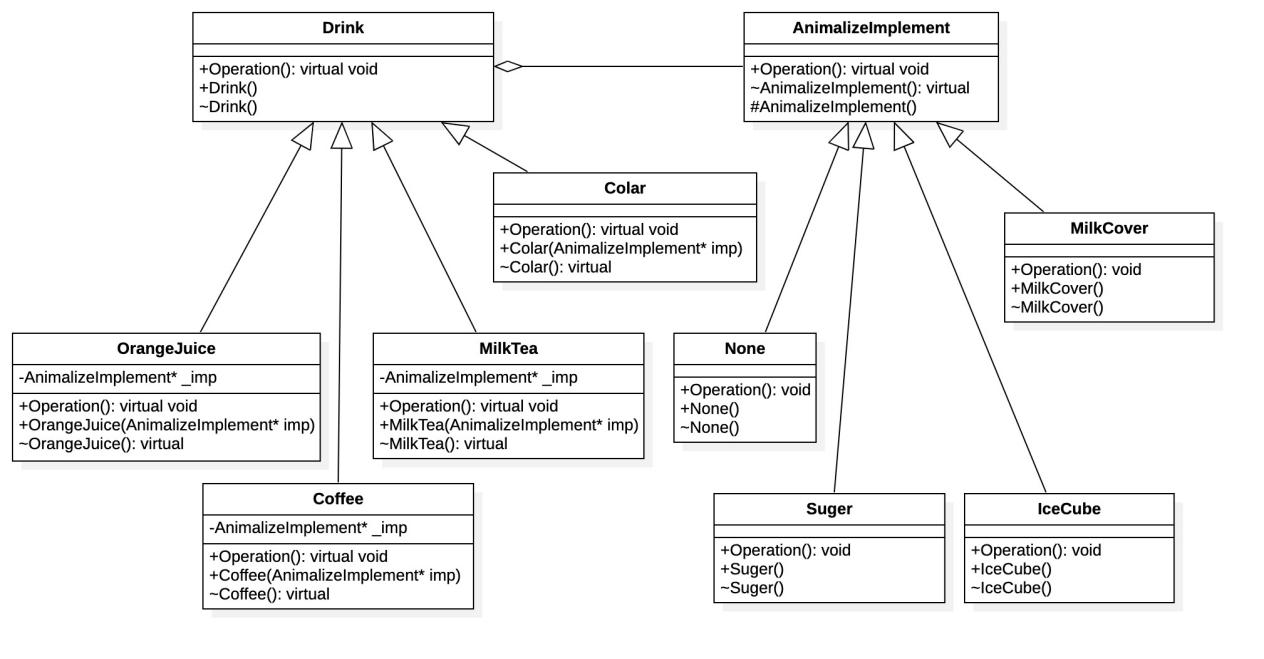
1. 实现API描述:

Bridge设计模式是指将抽象部份与它的实现部分分离，使它们都可以独立地变化，简单粗暴的说,就是抽象对外提供调用的接口；对外隐瞒实现部分在抽象中引用实现部分，从而实现抽象对实现部分的调用，而抽象中引用的" 实现部分可以在今后的开发过程中，切换成别的实现部分。

对于运动场上的饮料自助制作机来说，饮料的种类会不断的增加变化（抽象发生变化），而根据小动物自己的喜爱，想要往饮料中添加的伴侣也会发生变化（实现部分发生变化），因此选用Bridge来实现：

用一个Drink类作为所有饮料的父类,AnimalizeImplement作为一些实现操作的父类，在Drink中存放一个指向AnimalizeImplement类型的指针，以及一个用以实现具体操作的接口Operation，即实现了从抽象部分到实现部分的桥接，在实例化Drink的抽象子类比如OrangeJuice时，即可指定一个某一AnimalizeImplement操作比如加冰（IceCube），来构造一杯加冰的橙汁。

本设计模式的测试接口写在了一个名为Bridge的类中，只需实例化一个Bridge类型的对象即可调用其公共函数Test（）即可测试。

2.类图：

## 设计模式：Multiton

1. 实现APL描述：

多例模式Multiton是单例模式singelton的扩展,多例类可以有多个自己的实例,多例类必须自己创建，管理自己的实例，每个实例可以有不同的属性,多例类能够向外界提供自己的实例。一般的多例模式都默认为有上限的，叫做上限多例模式。

本场景为动物运动会闭幕之前主办方要进行一次抽奖，其形式为抽出5个上面数字随机的球，如果这五个上的数字与某个小动物运动员的编号相同，则中奖。这5个球几个视为从一个多例类中提取出来的五个实例，他们都可以有不同的属性（数字不同，为0～9中随机整数），本多例类LotteryBall(抽奖求）,创建管理自己实例是完全封装且随机的，因此可完美的模拟抽奖。

本设计模式的测试接口写在了一个名为Multiton的类中，只需实例化一个Multiton类型的对象即可调用其公共函数Test（）即可测试。

2.类图：

