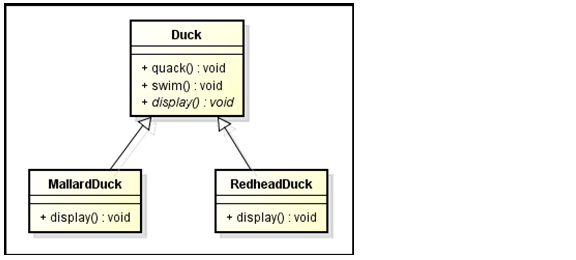
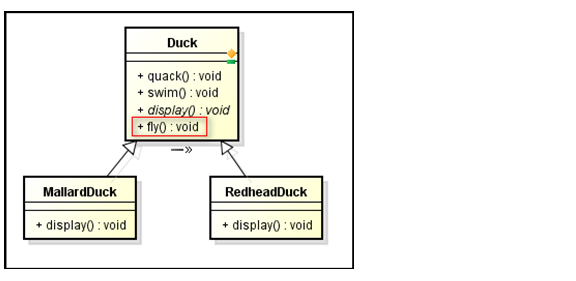
Design Pattern

1. Strategy
   1. background
      1. 

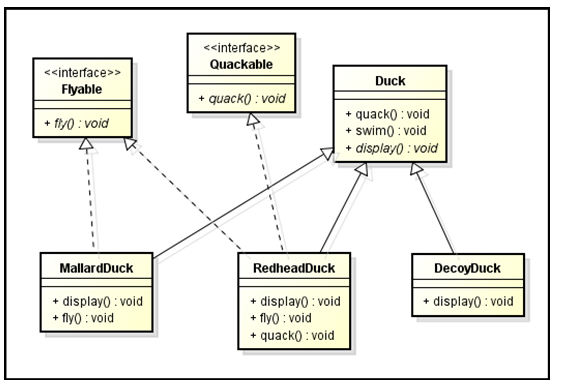
已有的系统如上所示，因为不同的鸭子有不同的颜色，所以display是一个abstract方法，有许多的子类继承自Duck类。

* 1. 问题的提出
     1. 现在想为游戏提供“飞”的功能，同时将来也要增加“叫”的功能
  2. 错误的解决方案
     1. 1



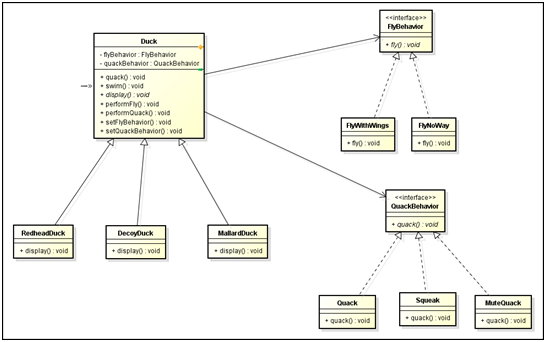
这种方案会污染所有的子类，即不应该能飞的鸭子也有这样的行为

* + 1. 2



增加两个接口，Flyable和Quackable，需要“飞”的鸭子实现这个接口，并且实现自己的“飞”逻辑，对于“叫”接口也类似，对于不需要“飞”或“叫"鸭子就不实现任何接口。

缺点：每个能”飞“的鸭子的业务都封装在各处的类内，这样无法实现代码重用，如果将来有一个很小的针对”飞“的改动，将导致要每一个实现了”飞“接口的子类都必须作修改。

* 1. 正确的解决方法
     1. 
  2. 设计原则
     1. 找出应用中可能需要变化之处，把它们独立出来，不要和那些不需要变化的代码混在一起（把会变化的部分取出并”封装“起来，好让其它部分不会受到影响）
     2. 针对接口编程，而不是针对实现编程

父类不要负责Fly和Quack的接口，反而是由我们制造的一组其它类专门实现FlyBehavior和QuackBehavior，这就称为”行为“类。由行为类而不是Duck类来实现行为接口

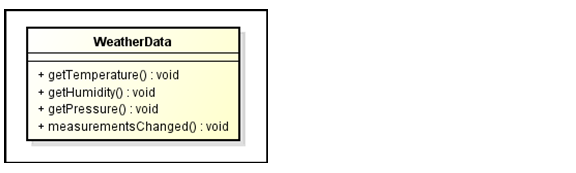
在我们的新设计中，鸭子的子类将使用接口，所表示的行为，所以实现的”实现“不会被绑死在鸭子的子类中

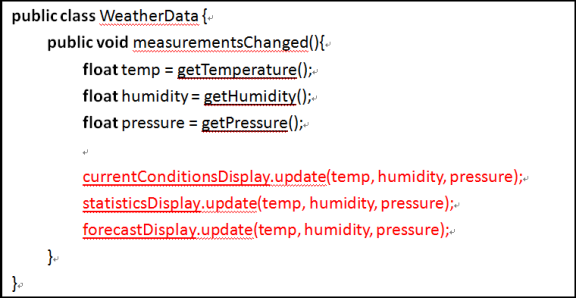
* + 1. 多用组合，少用继承

正如例子中所见，使用组合建立系统具有很大的弹性，不仅可将算法封装成类，更可以”在运行时动态地改变行为“，只要组合的行为对象符合正确的接口标准即可

* 1. 定义
     1. 策略模式定义了算法族，分别封装起来，让它们之间可以互相替换，此模式让算法的变化独立于使用算法的客户

1. Observer Pattern
   1. Background
      1. 需要在不同的布告板（“目前状况”布告板，“气象统计”布告板和“天气预报”布告析）上显示天气的信息
      2. 天气站发布的信息类



* 1. 错误的示范
     1. 
     2. 针对具体实现编程，会导致我们以后在增加或删除布告板时必须修改程序
     3. · update(temp, humidity,pressure)这一部分至少看起来就是一个接口，方法名和参数都是一样的
     4. 红色的部分就是改变的地方，需要封装起来