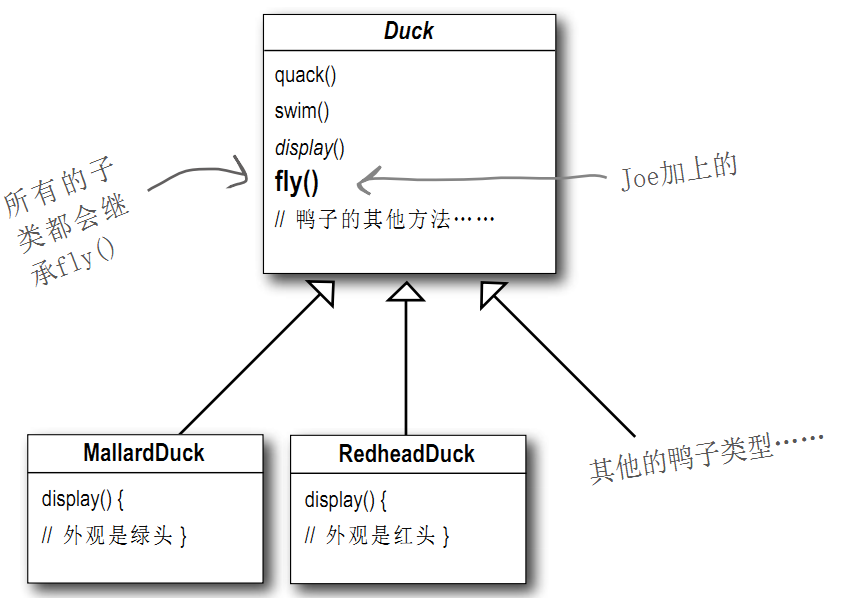
 最近在读《深入浅出设计模式》这一本书，收获颇多，打算写一个系列的blog，并用**[Python](http://lib.csdn.net/base/python" \o "Python知识库" \t "_blank)**实现其中的每一个设计模式，以加深理解。

       下面将要介绍的就是书中的第一个设计模式：**策略模式**

书中以模拟鸭子应用问题作为实例，一款游戏应用中要求有各种各样的鸭子。

       先用继承来实现了这一个应用，其设计如下：

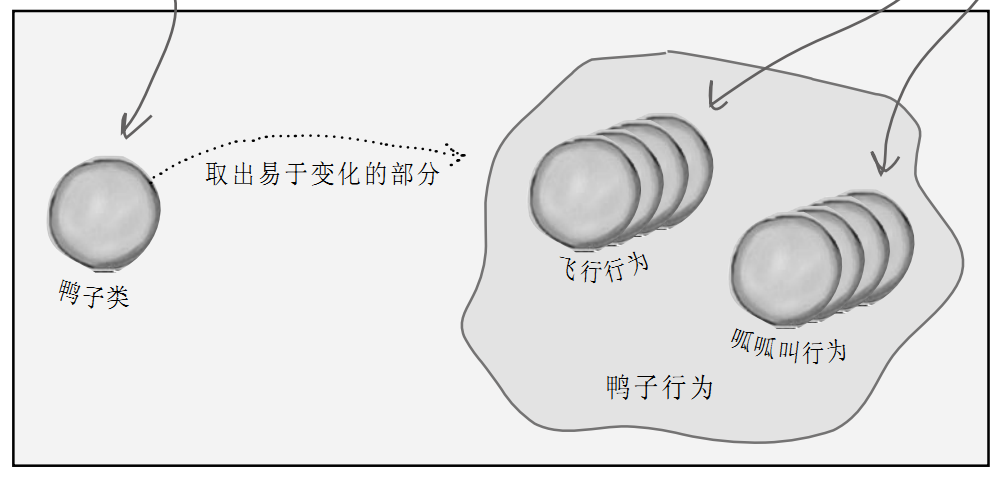


        这个设计主要是以Duck类作为基类，后面所有的鸭子类均以此类派生而来，刚开始在应用还不是很复杂的时候，似乎还没有什么问题，但当派生出来的类（鸭子的类型）越来越多时，问题就出现了。并不是所有的鸭子都会飞的，比如像皮鸭子就不会飞。也不是所有的鸭子都会呱呱叫，也有鸭子吱吱叫。也许你会说，我可以在派生类中重写基类的方法，从而达到应用的要求，实现不同的鸭子有不同的叫法，不同的飞行方式。但是有一个问题不能解决，重写函数并不应该改变原有函数的行为，比如fly()这个函数，不能飞的鸭子就不应该有fly()这个函数存在。而如果我们只是重写基类的fly()函数，看起来是不合适的。并且quack()函数用于发出“呱呱叫”，而基类中定义了quack()就意味着所有的鸭子中都有quck()，如果现在要求鸭子“吱吱叫”，怎么办呢？在基类中添加一个"吱吱叫“的函数？那这样又会影响到其它的子类。

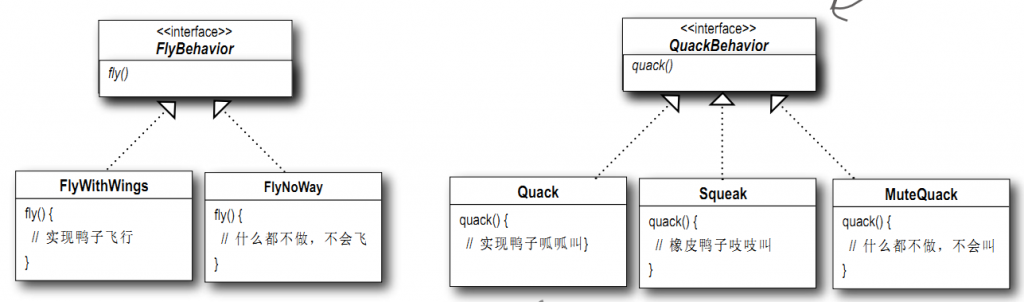
        如此看来，派生并不解决问题最好的办法，或者说不能只用派生来解问题。

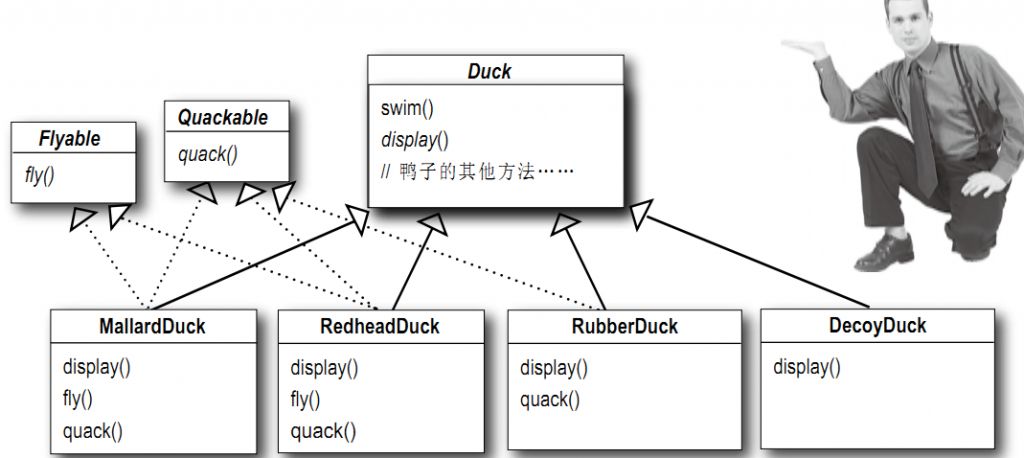
        分析一下，得到以下设计原则：

* **针对接口编程，而不是实现编程**
* **分离应用中经常变化的部分**



最终，我们分开了易于变化的部分，飞行行为和呱呱叫行为，设计出来的类图如下：





我用**[python](http://lib.csdn.net/base/python" \o "Python知识库" \t "_blank)**代码实现如下：

**[python]** [view plain](http://blog.csdn.net/huangkangying/article/details/7853418) [copy](http://blog.csdn.net/huangkangying/article/details/7853418)

1. '''''
2. The first Design Pattern:
3. Strategy Pattern.
4. KeyNote:
5. Identify the aspects of your application that vary and separate them
6. from what stays the same.
7. '''
9. **class** FlyBehavior:
10. ''''' Interface class: FlyBehavior '''
11. **def** fly(self):
12. **return**
14. **class** FlyWithWing(FlyBehavior):
15. **def** fly(self):
16. **print** 'I am flying with wings!'
18. **class** FlyNoWay(FlyBehavior):
19. **def** fly(self):
20. **print** 'I cannot fly!'
22. **class** QuackBehavior:
23. ''''' Interface Behavior: QuackBehavior '''
24. **def** quack(self):
25. **return**
26. **class** Quack(QuackBehavior):
27. **def** quack(self):
28. **print** 'Quack!'
30. **class** Squeak(QuackBehavior):
31. **def** quack(self):
32. **print** 'Squeak'
34. **class** MuteQuack(QuackBehavior):
35. **def** quack(self):
36. **print** 'MuteQuack'
38. **class** Duck:
39. '''''Base class: Duck. All ducks are inherent from this class'''
40. **def** \_\_init\_\_(self, flyParam, quackParam):
41. self.flyBehavior = flyParam
42. self.quackBehavior = quackParam
43. **def** performFly(self):
44. self.flyBehavior.fly()
45. **def** performQuack(self):
46. self.quackBehavior.quack()
47. **def** swim(self):
48. **print** 'All ducks can swim...'
49. **return**
50. **def** display(self):
51. **return**
53. **class** RedDuck(Duck):
54. **def** \_\_init\_\_(self, flyParam=FlyWithWing(), quackParam=MuteQuack()):
55. Duck.\_\_init\_\_(self, flyParam, quackParam)
56. **def** display(self):
57. **print** 'I am a red duck!'
58. **return**
60. **class** RubberDuck(Duck):
61. **def** \_\_init\_\_(self, flyParam=FlyNoWay(), quackParam=Quack()):
62. Duck.\_\_init\_\_(self, flyParam, quackParam)
63. **def** display(self):
64. **print** 'I am a rubber duck!'
66. duck = RedDuck()
67. duck.display()
68. duck.performFly()
69. duck.performQuack()
70. duck.swim()
72. duck = RubberDuck()
73. duck.display()
74. duck.performFly()
75. duck.performQuack()
76. duck.swim()