## 装饰者：

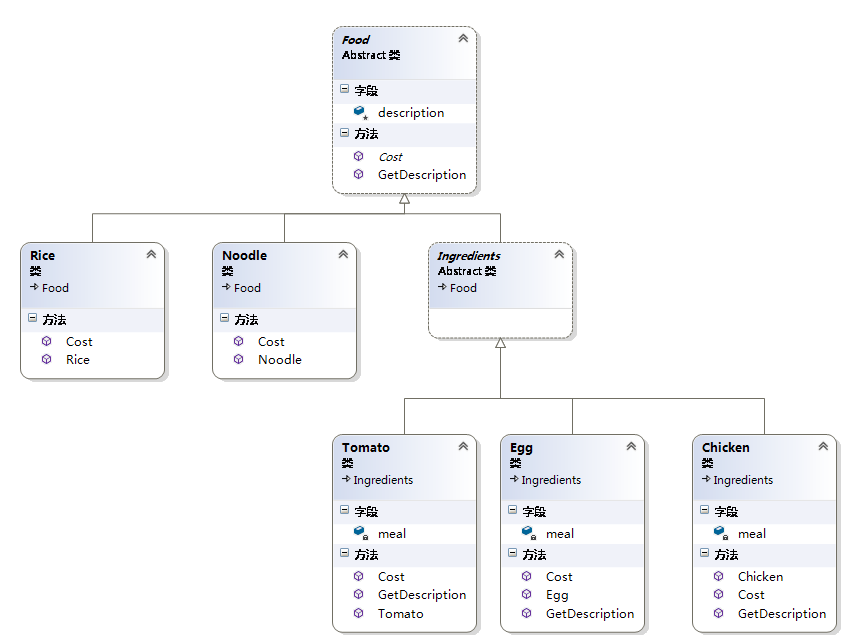
装饰者ingredients和被装饰对象rice, Noodle具有相同的超类型，Food

可以用一个或多个装饰者包装一个对象（西红柿鸡蛋面）

由于装饰者和被装饰对象有相同的超类型，所以在任何需要原始对象（被包装）的场合，都可以用装饰过的对象代替她

装饰者可以在所委托被装饰者的行为之前或之后，加上自己的行为，以达到特定的目的

对象可以在任何时候被装饰，所以可以在运行时动态的、不限量的用需要的装饰者来装饰对象



//创建被装饰的对象 noodle

Food f1 = new Noodle();

//用鸡蛋装饰

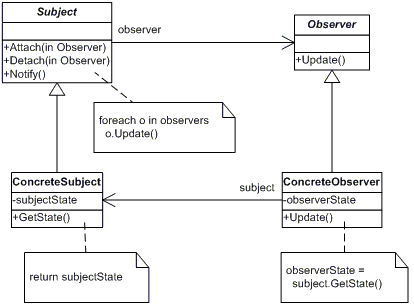
f1 = new Egg(f1);

//用西红柿装饰

f1 = new Tomato(f1);

## 观察者：

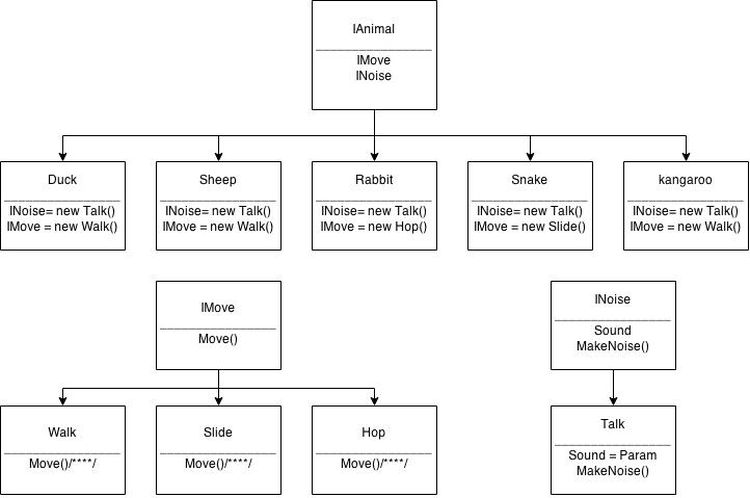
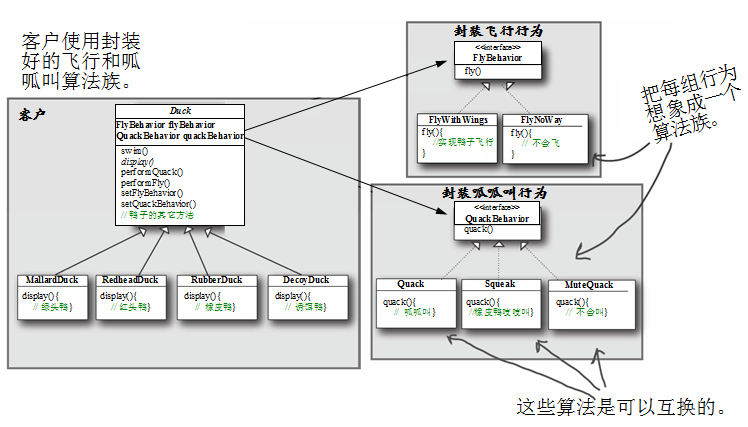
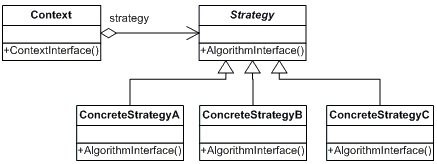
观察者模式（Observer Pattern）：定义了对象之间的一种一对多依赖关系，使得每当一个对象状态发生改变时，其相关依赖对象皆得到通知并被自动更新。观察者模式又被称为发布-订阅（Publish/Subscribe）模式



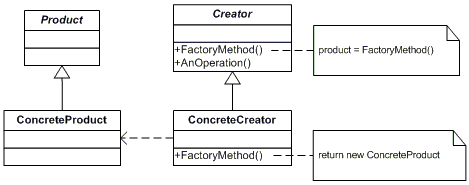
## 策略模式：

每一鸭子都有一个FlyBehavior且有一个QuackBehavior，让鸭子将飞行和呱呱叫委托它们代为处理。

如果将两个类结合起来使用(如同本例)，这就是组合(Composition)。这种作法和继承不同的地方在于：鸭子的行为不是继承而来，而是和适当的行为对象组合而来。



## 工厂模式：



  /// <summary>

  /// The 'Product' abstract class

  /// </summary>

  abstract class Page

  {

  }

  class SkillsPage : Page

  {

  }

  /// A 'ConcreteProduct' class

  class EducationPage : Page

  {

  }

  /// A 'ConcreteProduct' class

  class ExperiencePage : Page

  {

  }

1. abstract class Document
2. {
3. private List<Page> \_pages = new List<Page>();
5. // Constructor calls abstract Factory method
6. public Document()
7. {
8. this.CreatePages();
9. }
11. public List<Page> Pages
12. {
13. get { return \_pages; }
14. }
16. // Factory Method
17. public abstract void CreatePages();
18. }
19. /// <summary>
20. /// A 'ConcreteCreator' class
21. /// </summary>
22. class Resume : Document
23. {
24. // Factory Method implementation
25. public override void CreatePages()
26. {
27. Pages.Add(new SkillsPage());
28. Pages.Add(new EducationPage());
29. Pages.Add(new ExperiencePage());
30. }
31. }
33. /// <summary>
34. /// A 'ConcreteCreator' class
35. /// </summary>
36. class Report : Document
37. {
38. // Factory Method implementation
39. public override void CreatePages()
40. {
41. Pages.Add(new IntroductionPage());
42. Pages.Add(new ResultsPage());
43. Pages.Add(new ConclusionPage());
44. Pages.Add(new SummaryPage());
45. Pages.Add(new BibliographyPage());
46. }
47. }
48. }