**设计模式之Decorator(油漆工)**

装饰模式:Decorator常被翻译成"装饰",我觉得翻译成"油漆工"更形象点,油漆工(decorator)是用来刷油漆的,那么被刷油漆的对象我们称decoratee.这两种实体在Decorator模式中是必须的.

**Decorator定义**:  
动态给一个对象添加一些额外的职责,就象在墙上刷油漆.使用Decorator模式相比用生成子类方式达到功能的扩充显得更为灵活.

**为什么使用Decorator**?

我们通常可以使用继承来实现功能的拓展,如果这些需要拓展的功能的种类很繁多,那么势必生成很多子类,增加系统的复杂性,同时,使用继承实现功能拓展,我们必须可预见这些拓展功能,这些功能是编译时就确定了,是静态的.

使用Decorator的理由是:这些功能需要由用户动态决定加入的方式和时机.Decorator提供了"即插即用"的方法,在运行期间决定何时增加何种功能.

**如何使用**?  
举Adapter中的打桩示例,在Adapter中有两种类:方形桩 圆形桩,Adapter模式展示如何综合使用这两个类,在Decorator模式中,我们是要在打桩时增加一些额外功能,比如,挖坑 在桩上钉木板等,不关心如何使用两个不相关的类.

我们先建立一个接口:

|  |
| --- |
| public interface Work {  　　public void insert();  } |

接口Work有一个具体实现:插入方形桩或圆形桩,这两个区别对Decorator是无所谓.我们以插入方形桩为例:

|  |
| --- |
| public class SquarePeg implements Work{ 　　public void insert(){ 　　　　System.out.println("方形桩插入"); 　　}  } |

现在有一个应用:需要在桩打入前,挖坑,在打入后,在桩上钉木板,这些额外的功能是动态,可能随意增加调整修改,比如,可能又需要在打桩之后钉架子(只是比喻).

那么我们使用Decorator模式,这里方形桩SquarePeg是decoratee(被刷油漆者),我们需要在decoratee上刷些"油漆",这些油漆就是那些额外的功能.

|  |
| --- |
| public class Decorator implements Work{  　　private Work work; 　　//额外增加的功能被打包在这个List中 　　private ArrayList others = new ArrayList();  　　//在构造器中使用组合new方式,引入Work对象; 　　public Decorator(Work work) 　　{ 　　　　this.work=work; 　　  　　　　others.add("挖坑");  　　　　others.add("钉木板"); 　　}  　　public void insert(){  　　　　newMethod(); 　　}  　　 　　//在新方法中,我们在insert之前增加其他方法,这里次序先后是用户灵活指定的 　　  　　public void newMethod() 　　{ 　　　　otherMethod(); 　　　　work.insert();   　　}  　　public void otherMethod() 　　{ 　　　　ListIterator listIterator = others.listIterator(); 　　　　while (listIterator.hasNext()) 　　　　{ 　　　　　　System.out.println(((String)(listIterator.next())) + " 正在进行"); 　　　　}  　　}  } |

在上例中,我们把挖坑和钉木板都排在了打桩insert前面,这里只是举例说明额外功能次序可以任意安排.

好了,Decorator模式出来了,我们看如何调用:

Work squarePeg = new SquarePeg();   
Work decorator = new Decorator(squarePeg);  
decorator.insert();

Decorator模式至此完成.

如果你细心,会发现,上面调用类似我们读取文件时的调用:

FileReader fr = new FileReader(filename);  
BufferedReader br = new BufferedReader(fr);

实际上Java 的I/O API就是使用**[Decorator实现的](http://www-900.ibm.com/developerWorks/java/l-jdkdp/part3/index.shtml" \l "2" \t "_blank)**,I/O变种很多,如果都采取继承方法,将会产生很多子类,显然相当繁琐.

**Jive中的Decorator实现**  
在论坛系统中,有些特别的字是不能出现在论坛中如"打倒XXX",我们需要过滤这些"反动"的字体.不让他们出现或者高亮度显示.

在IBM Java专栏中专门**[谈Jive的文章](http://www-900.ibm.com/developerWorks/java/l-jivesrc/index.shtml" \l "5" \t "_blank)**中,有谈及Jive中ForumMessageFilter.java使用了Decorator模式,其实,该程序并没有真正使用Decorator,而是提示说:针对特别论坛可以设计额外增加的过滤功能,那么就可以重组ForumMessageFilter作为Decorator模式了.

所以,我们在分辨是否真正是Decorator模式,以及会真正使用Decorator模式,一定要把握好Decorator模式的定义,以及其中参与的角色(Decoratee 和Decorator).

# [装饰模式(Decorator Pattern)](http://www.cnblogs.com/lds85930/articles/643079.html)

## 一、 装饰（Decorator）模式

装饰（Decorator）模式又名包装（Wrapper）模式[GOF95]。装饰模式以对客户端透明的方式扩展对象的功能，是继承关系的一个替代方案。

**引言**

孙悟空有七十二般变化，他的每一种变化都给他带来一种附加的本领。他变成鱼儿时，就可以到水里游泳；他变成雀儿时，就可以在天上飞行。而不管悟空怎么变化，在二郎神眼里，他永远是那只猢狲。

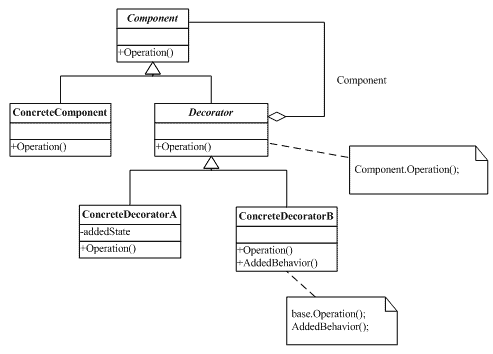
装饰模式以对客户透明的方式动态地给一个对象附加上更多的责任。换言之，客户端并不会觉得对象在装饰前和装饰后有什么不同。装饰模式可以在不使用创造更多子类的情况下，将对象的功能加以扩展。

## 二、 装饰模式的结构

装饰模式使用原来被装饰的类的一个子类的实例，把客户端的调用委派到被装饰类。装饰模式的关键在于这种扩展是完全透明的。

在孙猴子的例子里，老孙变成的鱼儿相当于老孙的子类，这条鱼儿与外界的互动要通过"委派"，交给老孙的本尊，由老孙本尊采取行动。

装饰模式的类图如下图所示：



在装饰模式中的各个角色有：

* 抽象构件（Component）角色：给出一个抽象接口，以规范准备接收附加责任的对象。
* 具体构件（Concrete Component）角色：定义一个将要接收附加责任的类。
* 装饰（Decorator）角色：持有一个构件（Component）对象的实例，并定义一个与抽象构件接口一致的接口。
* 具体装饰（Concrete Decorator）角色：负责给构件对象"贴上"附加的责任。

## 三、 装饰模式示例性代码

以下示例性代码实现了装饰模式：

http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// Decorator pattern -- Structural example    
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifusing System;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "Component"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifabstract class Component  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  abstract public void Operation();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "ConcreteComponent"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifclass ConcreteComponent : Component  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  override public void Operation()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine("ConcreteComponent.Operation()");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "Decorator"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifabstract class Decorator : Component  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Fields  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  protected Component component;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public void SetComponent( Component component )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    this.component = component;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  override public void Operation()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    if( component != null )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif      component.Operation();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "ConcreteDecoratorA"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifclass ConcreteDecoratorA : Decorator  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Fields  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private string addedState;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  override public void Operation()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    base.Operation();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    addedState = "new state";  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine("ConcreteDecoratorA.Operation()");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "ConcreteDecoratorB"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifclass ConcreteDecoratorB : Decorator  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  override public void Operation()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    base.Operation();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    AddedBehavior();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine("ConcreteDecoratorB.Operation()");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  void AddedBehavior()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif/// <summary>  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif/// Client test  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif/// </summary>  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifpublic class Client  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public static void Main( string[] args )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    // Create ConcreteComponent and two Decorators  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    ConcreteComponent c = new ConcreteComponent();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    ConcreteDecoratorA d1 = new ConcreteDecoratorA();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    ConcreteDecoratorB d2 = new ConcreteDecoratorB();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    // Link decorators  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    d1.SetComponent( c );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    d2.SetComponent( d1 );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    d2.Operation();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}

上面的代码在执行装饰时是通过SetComponent方法实现的，在实际应用中，也有通过构造函数实现的，一个典型的创建过程可能如下：

http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifnew Decorator1(  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif   new Decorator2(  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif      new Decorator3(  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif         new ConcreteComponent()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif         )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif      )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif   )

装饰模式常常被称为包裹模式，就是因为每一个具体装饰类都将下一个具体装饰类或者具体构件类包裹起来。

## 四、 装饰模式应当在什么情况下使用

在以下情况下应当使用装饰模式：

1. 需要扩展一个类的功能，或给一个类增加附加责任。
2. 需要动态地给一个对象增加功能，这些功能可以再动态地撤销。
3. 需要增加由一些基本功能的排列组合而产生的非常大量的功能，从而使继承关系变得不现实。

## 五、 装饰模式实际应用的例子

该例子演示了通过装饰模式为图书馆的图书与录像带添加"可借阅"装饰。

http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// Decorator pattern -- Real World example    
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifusing System;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifusing System.Collections;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "Component"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifabstract class LibraryItem  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Fields  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private int numCopies;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Properties  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public int NumCopies  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif    get{ return numCopies; }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif    set{ numCopies = value; }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public abstract void Display();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "ConcreteComponent"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifclass Book : LibraryItem  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Fields  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private string author;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private string title;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Constructors  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public Book(string author,string title,int numCopies)  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    this.author = author;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    this.title = title;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    this.NumCopies = numCopies;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public override void Display()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine( " Book ------ " );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine( " Author: {0}", author );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine( " Title: {0}", title );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine( " # Copies: {0}", NumCopies );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "ConcreteComponent"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifclass Video : LibraryItem  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Fields  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private string director;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private string title;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private int playTime;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Constructor  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public Video( string director, string title,  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    int numCopies, int playTime )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    this.director = director;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    this.title = title;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    this.NumCopies = numCopies;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    this.playTime = playTime;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public override void Display()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine( " Video ----- " );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine( " Director: {0}", director );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine( " Title: {0}", title );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine( " # Copies: {0}", NumCopies );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine( " Playtime: {0}", playTime );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "Decorator"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifabstract class Decorator : LibraryItem  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Fields  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  protected LibraryItem libraryItem;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Constructors  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public Decorator ( LibraryItem libraryItem )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  { this.libraryItem = libraryItem; }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public override void Display()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  { libraryItem.Display(); }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "ConcreteDecorator"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifclass Borrowable : Decorator  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Fields  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  protected ArrayList borrowers = new ArrayList();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Constructors  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public Borrowable( LibraryItem libraryItem )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif    : base( libraryItem ) {}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public void BorrowItem( string name )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    borrowers.Add( name );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    libraryItem.NumCopies--;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public void ReturnItem( string name )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    borrowers.Remove( name );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    libraryItem.NumCopies++;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public override void Display()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    base.Display();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    foreach( string borrower in borrowers )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif      Console.WriteLine( " borrower: {0}", borrower );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif   
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif/// <summary>  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif///  DecoratorApp test  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif/// </summary>  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifpublic class DecoratorApp  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public static void Main( string[] args )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    // Create book and video and display  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Book book = new Book( "Schnell", "My Home", 10 );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Video video = new Video( "Spielberg",  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif      "Schindler's list", 23, 60 );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    book.Display();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    video.Display();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    // Make video borrowable, then borrow and display  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine( " Video made borrowable:" );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Borrowable borrowvideo = new Borrowable( video );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    borrowvideo.BorrowItem( "Cindy Lopez" );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    borrowvideo.BorrowItem( "Samuel King" );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    borrowvideo.Display();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}

## 六、 使用装饰模式的优点和缺点

**使用装饰模式主要有以下的优点：**

1. 装饰模式与继承关系的目的都是要扩展对象的功能，但是装饰模式可以提供比继承更多的灵活性。
2. 通过使用不同的具体装饰类以及这些装饰类的排列组合，设计师可以创造出很多不同行为的组合。
3. 这种比继承更加灵活机动的特性，也同时意味着装饰模式比继承更加易于出错。

**使用装饰模式主要有以下的缺点：**

由于使用装饰模式，可以比使用继承关系需要较少数目的类。使用较少的类，当然使设计比较易于进行。但是，在另一方面，使用装饰模式会产生比使用继承关系更多的对象。更多的对象会使得查错变得困难，特别是这些对象看上去都很相像。

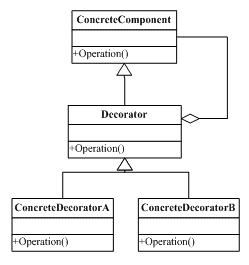
## 七、 模式实现的讨论

大多数情况下，装饰模式的实现都比上面定义中给出的示意性实现要简单。对模式进行简化时需要注意以下的情况：

（1）一个装饰类的接口必须与被装饰类的接口相容。

（2）尽量保持Component作为一个"轻"类，不要把太多的逻辑和状态放在Component类里。

（3）如果只有一个ConcreteComponent类而没有抽象的Component类（接口），那么Decorator类经常可以是ConcreteComponent的一个子类。如下图所示：



（4）如果只有一个ConcreteDecorator类，那么就没有必要建立一个单独的Decorator类，而可以把Decorator和ConcreteDecorator的责任合并成一个类。

## 八、 透明性的要求

**透明的装饰模式**

装饰模式通常要求针对抽象编程。装饰模式对客户端的透明性要求程序不要声明一个ConcreteDecorator类型的变量，而应当声明一个Component类型的变量。换言之，下面的做法是对的：

http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifComponent c = new ConcreteComponent()；  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifComponent c1 = new ConcreteDecorator1(c)；  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifComponent c2 = new ConcreteDecorator(c1)；

而下面的做法是不对的：

http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifConcreteComponent c = new ConcreteDecorator()；

这就是前面所说的，装饰模式对客户端是完全透明的含义。

用孙悟空的例子来说，必须永远把孙悟空的所有变化都当成孙悟空来对待，而如果把老孙变成的雀儿当成雀儿，而不是老孙，那就被老孙骗了，而这是不应当发生的。

下面的做法是不对的：

http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif大圣本尊 c = new 大圣本尊()；  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif雀儿 bird = new 雀儿 (c)；

**半透明的装饰模式**

然而，纯粹的装饰模式很难找到。装饰模式的用意是在不改变接口的前提下，增强所考虑的类的性能。在增强性能的时候，往往需要建立新的公开的方法。即便是在孙大圣的系统里，也需要新的方法。比如齐天大圣类并没有飞行的能力，而雀儿有。这就意味着雀儿应当有一个新的fly()方法。

这就导致了大多数的装饰模式的实现都是"半透明"（semi-transparent）的，而不是完全"透明"的。换言之，允许装饰模式改变接口，增加新的方法。即声明ConcreteDecorator类型的变量，从而可以调用ConcreteDecorator类中才有的方法：

http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif齐天大圣 c = new 大圣本尊()；  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif雀儿 bird = new 雀儿(c)；  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifbird.fly()；

齐天大圣接口根本没有fly()这个方法，而雀儿接口里有这个方法。