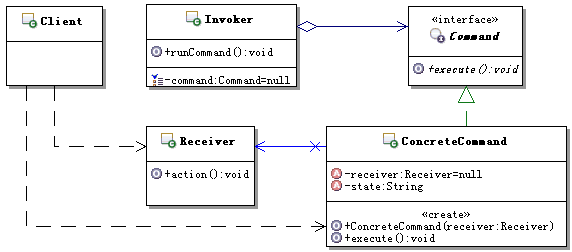
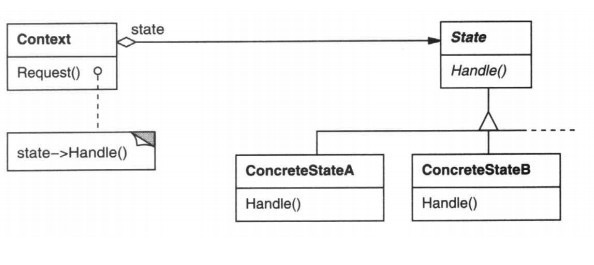
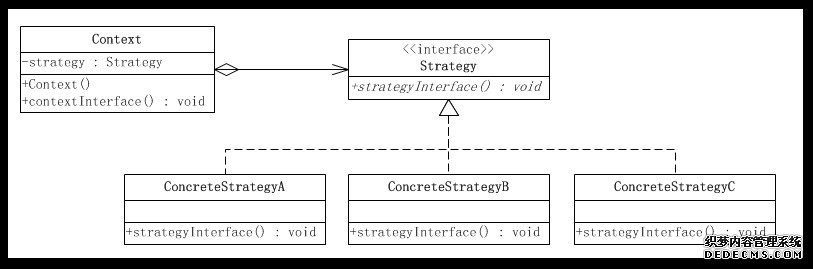
1.命令模式（command）将一个请求封装为一个对象，从而使你可用不同的请求对客户进行参数化；对请求排队或记录请求日志，以及支持可撤销的操作。命令模式的本质是对命令进行封装，将发出命令的责任和执行命令的责任分割开，实现二者之间的松耦合。－－类图：



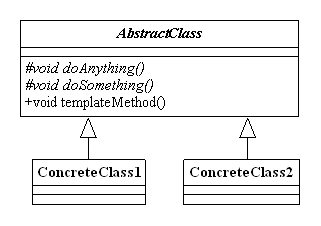
2.状态模式（state）允许对象在内部状态改变时改变它的行为，对象看起来好像修改了他的类。

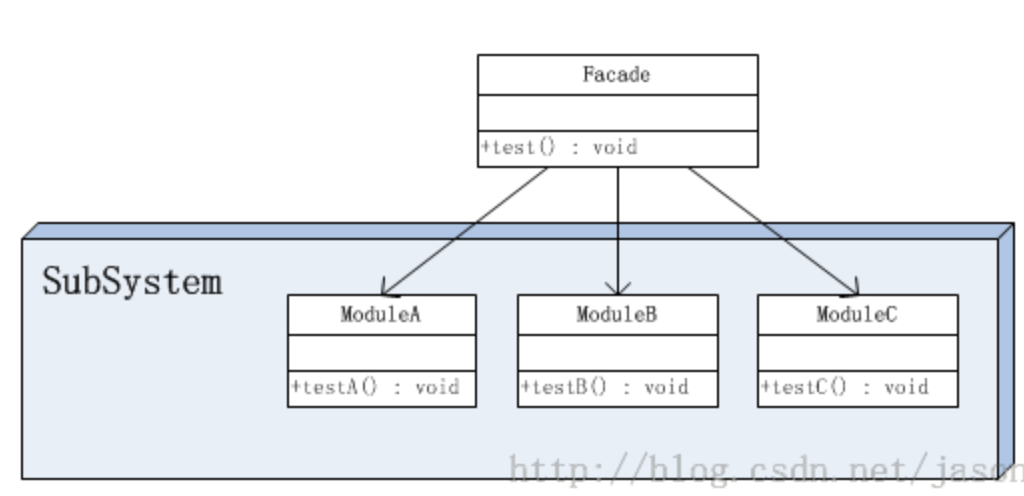
一个对象的行为取决于他的状态，并且它必须在运行时根据状态改变它的行为，一个操作中含有庞大的多分支结构，并且这些分支决定于对象的状态。———-类图：



3.策略模式（strategy）定义了一系列的算法，并将每一个算法封装起来，而且使他们还可以相互替换。策略模式让算法独立于是用它的客户而独立变化。多个类只区别在表现行为不同，可以使用strategy模式，在运行时动态选择具体要执行的行为。需要在不同情况下使用不同的略（算法），或者策略还可能在未来用其他方式来实现。对客户隐藏具体策略（算法）的实现细节，彼此完全独立

4.模板模式（template）定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。模板方法使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤.如果你只想掌握一种设计模式,那么它就是模板方法

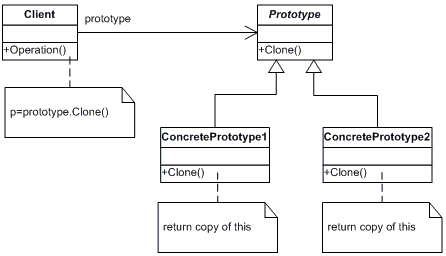


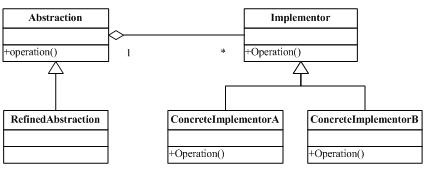
5.门面模式（facade）要求一个子系统的外部与其内部的通信必须通过一个统一的门面(Facade)对象进行。门面模式提供一个高层次的接口，使得子系统更易于使用。就如同医院的接待员一样，门面模式的门面类将客户端与子系统的内部复杂性分隔开，使得客户端只需要与门面对象打交道，而不需要与子系统内部的很多对象打交道。

6.原型模式(prototype)是一种创建型设计模式,它通过复制一个已经存在的实例来返回新的实例,而不是新建实例.被复制的实例就是我们所称的原型,这个原型是可定制的.原型模式多用于创建复杂的或者耗时的实例, 因为桥接种情况下,复制一个已经存在的实例可以使程序运行更高效,或者创建值相等,只是命名不一样的同类数据.原型模式中的拷贝分为"浅拷贝"和"深拷贝":

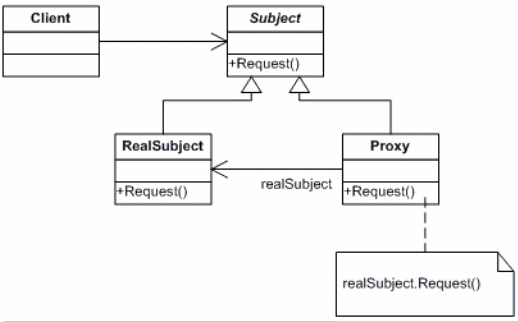
浅拷贝: 对值类型的成员变量进行值的复制,对引用类型的成员变量只复制引用,不复制引用的对象.

深拷贝: 对值类型的成员变量进行值的复制,对引用类型的成员变量也进行引用对象的复制.

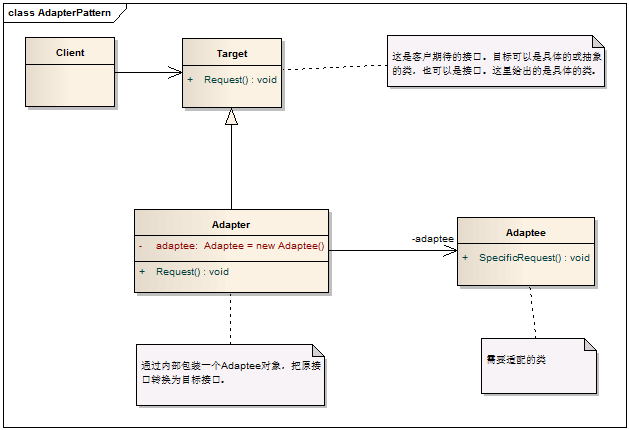


7桥接模式(bridge)就是把抽象部分和它的实现部分分离开来，让两者可独立变化。这里的抽象部分指的是一个概念层次上的东西，它的实现部分指的是实现这个东西的（功能）部分，分离就把实现部分从它要实现的抽象部分独立出来，自我封装成对象

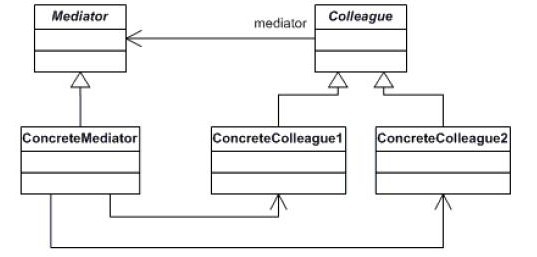
8代理模式(proxy)为另一个对象提供一个替身或占位符以控制对这个对象的访问，简而言之就是用一个对象来代表另一个对象。



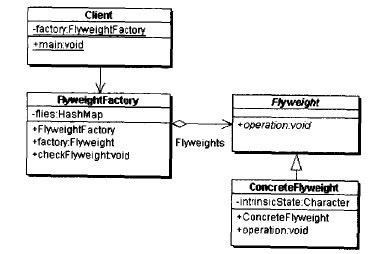
9适配器模式(adapter)将一类的接口转换成客户希望的另外一个接口，Adapter模式使得原本由于接口不兼容而不能一起工作那些类可以一起工作。使用的前提是：接口中规定了所有要实现的方法； 但一个要实现此接口的具体类，只用到了其中的几个方法，而其它的方法都是没有用的。实现方法：用一个抽象类实现已有的接口，并实现接口中所规定的所有方法，这些方法的实现可以都是空方法。原本要实现接口的子类，只实现上面的抽象类即可，并在其内部实现时，只对其感兴趣的方法进行实现；

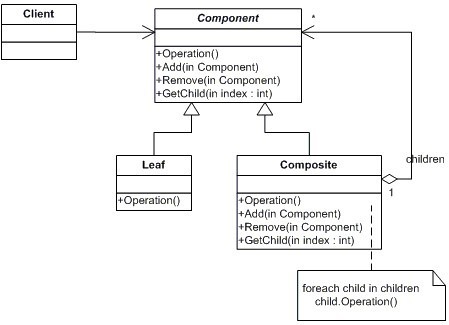


10.中介者模式(Mediator)是行为模式之一。定义一个中介对象来封装系列对象之间的交互。中介者使各个对象不需要显示地相互引用，从而使其耦合性松散，而且可以独立地改变他们之间的交互

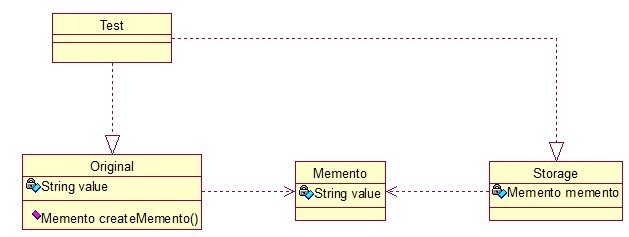
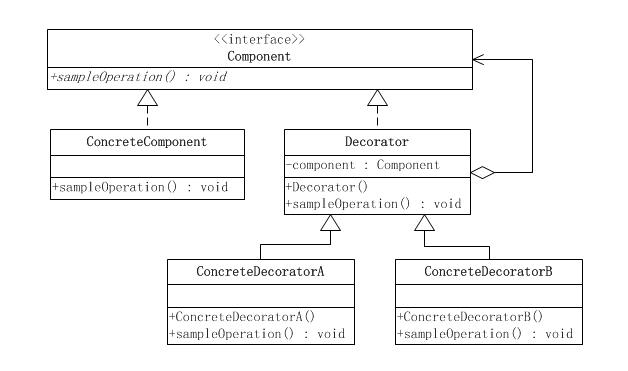


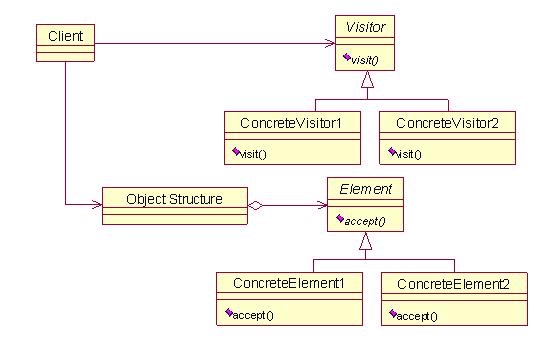
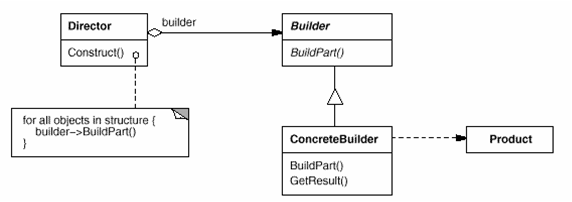
11享元模式(Flyweight)对象的结构模式。享元模式以共享的方式高效地支持大量的细粒度对象。在单纯的享元模式中，所有的享元对象都是可以共享的。在java语言中，String类型就是使用了享元模式。String对象是不变的对象，一旦创建出来就不能改变，如果要改变一个字符串的值，只能创建一个新的String对象。在JVM 内部，String类型是共享的，如果一个系统中有两个String对象所包含的字符串相同，JVM 实际上只会创建一个String对象提供给两个引用，从而实现了对String的共享。String的intern()方法给出这个字符串在共享池中的唯一实例



12组合模式(Composite)有时候又叫做部分-整体模式，它使我们树型结构的问题中，模糊了简单元素和复杂元素的概念，客户程序可以向处理简单元素一样来处理复杂元素,从而使得客户程序与复杂元素的内部结构解耦。组合模式让你可以优化处理递归或分级数据结构  


13观察者模式（observer）又叫做发布-订阅（Publish/Subscribe）模式、模型-视图（Model/View）模式、源-监听器（Source/Listener）模式或从属者（Dependents）模式。观察者模式定义了一种一对多的依赖关系，让多个观察者对象同时监听某一个主题对象。这个主题对象在状态上发生变化时，会通知所有观察者对象，使它们能够自动更新自己。一个软件系统常常要求在某一个对象的状态发生变化的时候，某些其它的对象做出相应的改变。做到这一点的设计方案有很多，但是为了使系统能够易于复用，应该选择低耦合度的设计方案。减少对象之间的耦合有利于系统的复用，但是同时设计师需要使这些低耦合度的对象之间能够维持行动的协调一致，保证高度的协作（Collaboration）。观察者模式是满足这一要求的各种设计方案中最重要的一种  


14备忘录模式(memento)又叫做快照模式(Snapshot Pattern)或Token模式，属于行为模式。它是在不破坏封闭的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态，这样以后就可将该对象恢复到原先保存的状态  
  
15装饰模式(decorator)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。  


16访问者模式(visitor)表示一个作用于某对象结构中的各元素的操作，它使你可以在不改变各元素类的前提下定义作用于这些元素的新操作。从定义可以看出结构对象是使用访问者模式的必备条件，而且这个结构对象必须存在遍历自身各个对象的方法，类似于Java中的Collection。访问者模式的目的是要把处理从数据结构中分离出来，如果系统有比较稳定的数据结构，又有易于变化的算法的话，使用访问者模式是个不错的选择，因为访问者模式使的算法操作的增加变得容易。相反，如果系统的数据结构不稳定，易于变化，则此系统就不适合使用访问者模式了  
  
17建造者模式(builder)将一个复杂的构建与其表示相分离，使得同样的构建过程可以创建不同的表示。 [构建与表示分离，同构建不同表示]。与抽象工厂的区别：在建造者模式里，有个指导者，由指导者来管理建造者，用户是与指导者联系的，指导者联系建造者最后得到产品。即建造模式可以强制实行一种分步骤进行的建造过程。建造模式是将复杂的内部创建封装在内部，对于外部调用的人来说，只需要传入建造者和建造工具，对于内部是如何建造成成品的，调用者无需关心。  
  
18.工厂模式(factory)  
 工厂模式主要用一下几种形态：

a：简单工厂（Simple）:又叫静态工厂，是工厂模式三中状态中结构最为简单的。主要有一个静态方法，用来接受参数，并根据参数来决定返回实现同一接口的不同类的实例。

b：工厂方法（Factory）:把不同的产品放在实现了工厂接口的不同工厂类里面，这样就算其中一个工厂类出了问题，其他工厂类也能正常工作，互相不受影响，以后增加新产品，也只需要新增一个实现工厂接口工厂类，就能达到，不用修改已有的代码。

c：抽象工厂（Abstract）:工厂方法用来创建一个产品，它没有分类的概念，而抽象工厂则用于创建一系列产品，所以产品分类成了抽象工厂的重点  
  
