GRASP

通用职责分配软件模式

信息专家（Information expert）

创建者（Creator）

高内聚（High Cohesion）

低耦合（Low Coupling）

控制器（Controller）

多态性（Polymorphism）

纯虚构（Pure Fabrication）

间接性（Indirection）

防止变异（Protected Variations）

专家模式（expert）

谁都负责自己的信息的统计工作，也就是，你只负责你本身拥有的信息的处理，只有对自己了解的信息，你才是专家，才能拥有职责。

Sale 负责总价钱

SalesLineItem 负责每一个Line的总价钱

ProductSpecification 负责每一个产品的价格

创建者模式（creator）

谁拥有创建的信息，谁包含被创建对象，谁统计被创建对象的信息，谁负责创建对象。

Sale负责创建SalesLineItem,需要数量作为参数，因为只有Sale知道需要多少个数量的Item。

低耦合（low Coupling）

一个类型A和类型B之间的耦合关系有如下几种形式：

B是A的属性

A中的一个方法里面使用到B类型，或者以B类型作为参数

B是A的子类，或者间接的子类

A是一个接口，而B实现了A

三个类

Payment ，Post ， Sale

Sale负责创建payment，因为sale知道总价钱。

高内聚（high Cohesion）

内聚种类：

非常低内聚：一个类负责不同功能区不同的事情

低内聚：一个类负责相同功能区里面不同的事情

高内聚：一个类负责相同功能区里面一些职责，并和其他类一起完成该功能。

控制器 (controller)

指的是事件控制器，对于一个非图形界面事件，最好能够有专门的控制器来针对它做出处理。

对于一个entereItem（）的事件来说，需要有一个专门的控制器，接受事件，然后处理事件。

多态性（Polymorphism）

多用多态来实现行为判断，最好不要用if语句。

纯虚构（Pure Fabrication）

指的是，要实现高内聚，必须将功能单一化，虽然有时候class满足专家模式，但是，对于一些大型的，需要多次重用的方法，需要单独提取出来作为一个类，这样能够增加重用性。

间接性（Indirection）

有时候需要用间接的耦合来代替直接的耦合，对于上一个例子，通过建立一个中间类别来解开sale和database之间的耦合，把直接耦合转化为间接耦合。

防止变异（Protected Variations）

面向接口编程

迭代器模式（Iterator）

通过一个迭代器来访问容器。这样可以做到多态，因为对于不同的容器，他们的迭代器是相同，所以，你可以指定一个迭代器的接口，即可以访问所有容器的数据而不需要具体知道是哪个容器。；

代理模式（Proxy）

如果访问真实的主题有危险，或者有某些禁忌的话，可以使用代理模式，它通过将真实的主题和代理的主题共同实现一个父类来达到目的，父类中只有一个方法，供给客户端使用。

在代理中可以使用preRequest和postRequest。

原型模式（Prototype）

通过给出一个原型对象来指明所要对象的类型，然后用复制这个原型对象的办法创建出更多相同类型对象。

依赖倒置——DIP模式（Dependency Inversion Principle）

高层模块不应该依赖于底层模块，二者都应该依赖于抽象

抽象不应该依赖于细节，细节应该依赖于抽象

调停着模式（Mediator）

同样是一个中介的意思，也就是说，对于多个对象之间的复杂联系，通过一个中介来将他们所有的行为联系起来。

侨界模式（Bridge）

桥接模式最大的特点在于将2个不同方面的组合通过继承和接口的方式将他们组合起来，也就是我们数学里面的排列组合；

例如：A B 两个方式写入数据，而C D两个方式写入不同的平在

而传统的方式在与通过单方面的技能来实现，而桥接模式则能够实现AC，BC，AD，BD之间的任意组合。

外观模式（Facade）也可以叫前端模式

就是在一个子系统或者系统中（可能包含了很多种方法），通过建立一个类，来将这些你需要的方法的使用顺序，调用顺序组合起来，也就是一个中间类别的意思，同样是功能的抽象化。

Facade模式主要适用于以下几种情况:

* 不需要使用一个复杂系统的所有功能，而且可以创建一个新的类，包含访问系统的所有规则。如果只需要使用系统的部分功能，那么你为新类所创建的API将比原系统的API简单的多。
* 希望封装或者隐藏系统原系统。
* 希望使用原系统的功能，而且还希望增加一些新的功能。
* 编写新类的成本小于所有人学会使用或者未来维护原系统上所需的成本

开闭原则（OCP——Open Close Priciple）

对拓展开放

对修改关闭

也就是说我们在设计一个类的时候，是不需要修改类本身的代码就可以进行拓展。

修饰者模式（Decorator）

最主要的特点是代替继承，实现类的拓展

还有就是动态的添加职责。

适配器模式（adapter）

对于类适配器：就是用一个类继承一个父类，同时实现一个接口

对于对象适配器：就是用一个类，他有上一句中父类的属性，同时实现一个接口。

策略模式（Strategy）

通过类拥有一个接口的属性，通过具体不同的实现该接口的类来做不同的事情。

共性与可变性

找出共性

判断共性中还有那些可变的地方

通过一些方法来解决可变性

方法如下：

If语句是最差的方法

策略模式比继承好

观察者模式（Observer）

通过一个存放观察者的列表，维护观察者。然后在需要调用观察者的时候在notify链表对应的观察者。

命令模式（Command）

通过将命令封装成类的，通过指定统一的接口来实现命令的统一。

工厂模式（Factory）

分为三种：

1、简单工厂模式，又叫做静态工厂方法（Static Factory Method）模式。   
2、工厂方法模式，又称为多态性工厂(Polymorphic Factory)模式   
3、抽象工厂模式，又称工具（Kit或ToolKit）模式