<http://c.biancheng.net/design_pattern/>

1. **面向对象设计原则：**
   1. 开闭原则：
   * 对扩展开放，对修改关闭
   * 在不修改源代码的前提下，实现模块的扩展
   1. 里氏替换原则：
   * 子类尽量不要重写父类的方法
   1. 依赖倒置原则：
   * 核心思想：面向接口编程
   * 高层模块不依赖底层模块，两者都应该依赖抽象类
   * 抽象不依赖细节，细节应该依赖抽象
   1. 单一职责原则：
   * 一个类或者方法应该只负责一种职责
   1. 接口隔离原则：
   * 一个类对另一个类的依赖应该建立在最小接口上
   1. 迪米特法则：
   * 只与“直接朋友”交谈，不和“陌生人”交谈
   * 要尽量先使用组合或者聚合等关联关系来实现，其次才考虑使用继承关系来实现
   * 如果要使用继承关系，则必须严格遵循里氏替换原则
   1. 总结：

开闭原则是总纲，它告诉我们要对扩展开放，对修改关闭；里氏替换原则告诉我们不要破坏继承体系；依赖倒置原则告诉我们要面向接口编程；单一职责原则告诉我们实现类要职责单一；接口隔离原则告诉我们在设计接口的时候要精简单一；迪米特法则告诉我们要降低耦合度；合成复用原则告诉我们要优先使用组合或者聚合关系复用，少用继承关系复用。

1. **设计模式：**

一共23种设计模式

按目的可分为 **创建型、结构性、行为型** 三种模式

1. 创建型模式：单例、原型、工厂、抽象工厂、建造者
2. 结构性模式：代理、适配器、桥接、装饰、外观、享元、组合
3. **创建型模式**
   * 1. 单例模式：

一个类只有一个实例对象，且自行创建

避免了资源浪费

* + 懒汉式：

类加载时不创建实例，在第一次调用时创建，通过同步来保证线程安全，因此每次访问都要同步，会影响性能

* + 饿汉式：

类加载时就创建实例，不会有线程安全问题

* 扩展：
  + 有限的多例模式（Multitcm）

生成有限个实例并保存在List中，客户端需要时随机读取



* + 1. 原型模式：

用一个已经创建的实例作为原型，通过克隆该原型来创建一个相同或相似的新例对象

* + 实现cloneable接口，调用Object类的clone方法进行克隆

**该方法是浅克隆**

* + 需要克隆的类实现Serializable接口，通过对象流的写入和写出进行克隆

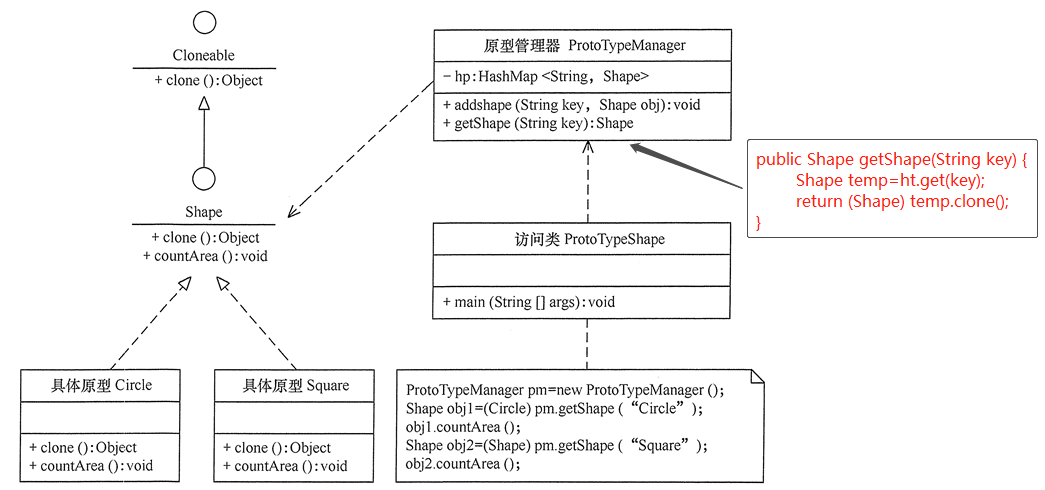
**该方法是深克隆**

|  |
| --- |
| // 对象写到流里面  ByteArrayOutputStream baos = new ByteArrayOutputStream();  ObjectOutputStream oos=new ObjectOutputStream(baos);  oos.writeObject(obj1); // 写入  // 将对象从流里读出  ByteArrayInputStream baoi = new ByteArrayInputStream(baos.toByteArray());  ObjectInputStream ois=new ObjectInputStream(baoi);  Realizetype obj2= (Realizetype) ois.readObject(); // 写出 |

* 扩展：
* 带原型管理的原型模式

增加一个原型管理器类，该类有个HashMap成员，保存多个原型

* + 通过add的方式添加原型到HashMap中
  + 通过get的方式复制HashMap中保存的原型



* + 1. 工厂模式：

定义一个创建产品对象的工厂接口，将产品对象的实际创建工作推迟到具体子工厂类当中

用户只需要知道具体工厂的名称就能得到产品，无须知道产品的创建过程

缺点：每增加一个产品就要增加一个具体产品类和对应的具体工厂类

分为简单工厂、工厂、抽象工厂三种

* + 普通工厂模式：

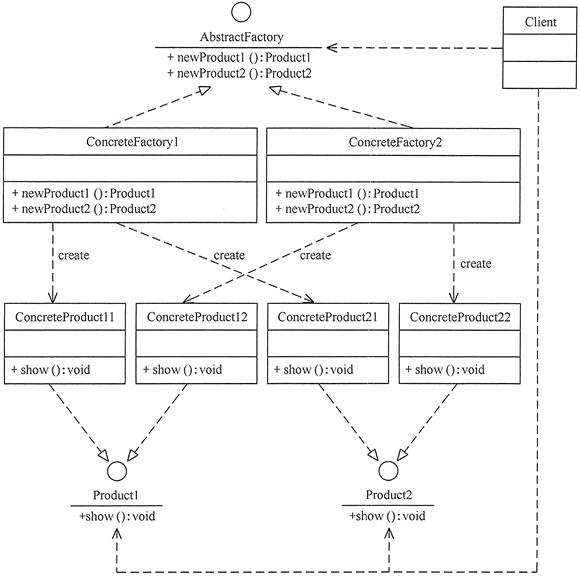


* + 抽象工厂模式：

普通工厂模式只能生产同等级的产品，如一个工厂只能生产手机却不能生产电视

抽象工厂能生产多个等级的产品，如小米工厂能生产小米手机和小米电视

缺点：当产品族里新增一个产品时，全是工厂都要修改



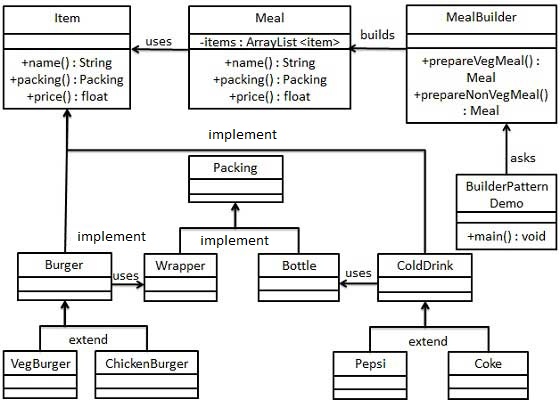
* + 1. 建造者模式：

指将一个复杂对象的构造与它的表示分离，使同样的构建过程可以创建不同的表示，这样的设计模式被称为建造者模式

它是将一个复杂的对象分解为多个简单的对象，然后一步一步构建而成。它将变与不变相分离，即**产品的组成部分是不变的，但每一部分是可以灵活选择的**

* 生产同一种产品，但有不同的属性

如：手机都由CPU+主板+系统组成，但是小米手机的CPU是高通，华为手机的CPU是麒麟，但它们具有一样的组成部分



1. **结构性模式**
   * 1. 代理模式：

为其他对象提供一种代理以控制对这个对象的访问

被代理的对象和代理对象均实现同一个接口

* 优点：
  1. 可扩展目标对象的功能（Spring Aop）
  2. 一定程度上降低了系统的耦合度
* 缺点：

1. 增加了一个中间层，处理速度变慢
2. 增加了系统的复杂度



**个人使用：**

1. 代理模式可以做一个任务的分发的模式，代理者根据传递的某个值，判断需要调用的具体任务处里者(可以利用反射创建处里者)，然后将任务分发给该处理者执行任务，代理者可以提前做一些值校验，校验不合法的数值
2. 这样的话我们只需关注下层的业务编写，无需关注上层如何调用，因为上层的调用已经写好，兼容各种任务
   * 1. 适配器模式：

将一个类的接口转换成客户希望的另外一个接口，使得原本由于接口不兼容而不能一起工作的那些类可以一起工作

一个类由于接口不兼容造成无法工作，可以通过一个适配器将类的接口转换为当前期望的接口，以此来进行工作

如：内存卡的无法直接插在电脑上读取，需要一个读卡器，把内存卡的接口转换为USB接口，因此能插入电脑进行读取

* 注意：

适配器模式不是在软件设计时添加的，而是为了解决正在服役的项目的问题



* 扩展：

双向适配器模式



代理模式和适配器模式的区别：

1. 代理模式提供一个中间层，客户端不直接使用目标对象，而是通过代理的方式来访问目标对象，可目标对象进行功能增强
2. 适配器模式是为了解决因接口不兼容导致的无法工作，更多的是一种解决方案
   * 1. 桥接模式：

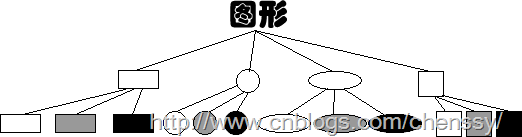
将抽象部分与实现部分分离，使它们都可以独立的变化

一个类具有多个的维度变化，第一个维度具有M种变化，第二个维度具有N种变化，这样会造成这个类具有M\*N种结果，如果利用继承实现，会有M\*N种实现类

如：汽车具有不同的品牌、不同的颜色、不同的形状、如果使用继承需要结合不同品牌、不同颜色、不同形状创建对应的类

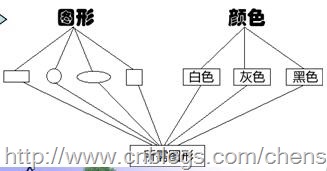
* 未使用桥接模式
* 图形和颜色继承关系：

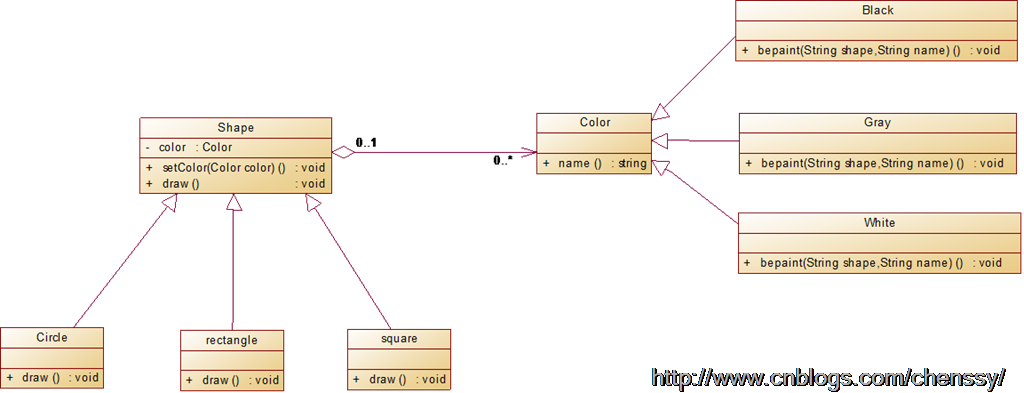
一个图形接口具有多种图形实体类，每一个图形具有不同的颜色，因此需要多种实现类



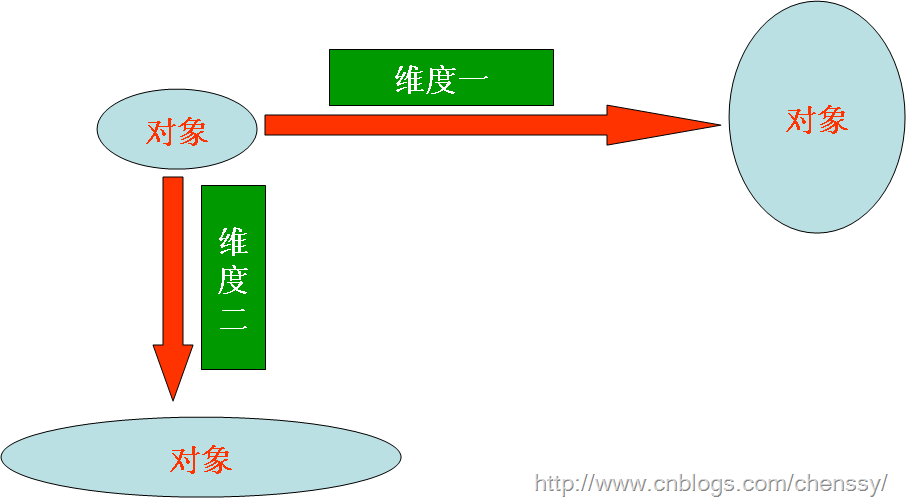
* 使用桥接模式
* 图形和颜色桥接模式：

如果使用桥接模式，将汽车创建为一个抽象类，将品牌、颜色、形状创建为接口，将这些接口利用组合关系组合到汽车抽象类中，这样只需给具体汽车指定具体的品牌、颜色、形状即可





* **桥接模式适合于一个具有多种维度变化的类，将这些维度抽象出来，然后将维度和类组合在一起**



* + 1. 组合模式：

在对象的内部包含一个相同的对象，使得这个对象具有树形的结构

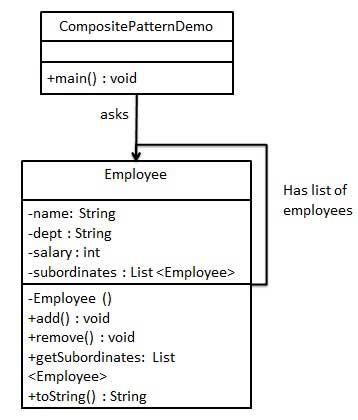
如：树具有树枝和树叶，树枝可以分叉出多个树枝，或者具有多个树叶，而树叶不能再分叉出其他树枝或者具有树叶

如：文件目录，文件目录可以具有文件夹和文件，文件夹里还可以包含文件夹或者文件

如：公司部门和员工，董事会具有多个董事长，董事会下面可以具有其他部门，其他部门里还可以具有其他的员工或者部门

* 优点：

1. 组合模式使得客户端代码可以一致地处理单个对象和组合对象，无须关心自己处理的是单个对象，还是组合对象，这简化了客户端代码
2. 更容易在组合体内加入新的对象，客户端不会因为加入了新的对象而更改源代码，满足“开闭原则”



* + 1. 装饰器模式：

在不改变对象结构的情况下，可以动态的给现有的对象增强一些功能

装饰器和被装饰者实现同一接口，装饰器里组合了被装饰者，装饰器在接口方法里调用或者增强被装饰者的接口方法

* 优点：

1. 可以动态的增强扩展方法
2. 装饰器和被装饰者可以独立发展
3. 可以代替继承

* 缺点：

1. 多层装饰器会造成代码复杂

