常用设计模式

姓名：张贵东

日期：2017年6月21日星期三

1. 代理模式

需要满足3个条件：

（1）执行者、被代理人

（2）对于被代理人来说，这件事情是必须要做的，但是自己又不想亲自去做或没有时间做

（3）持有被代理的引用（获取资料）

比如去火车票代售点购买车票：

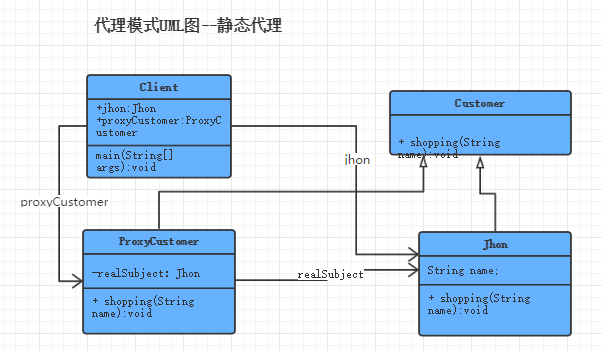
找代购网店购买手机等等

* 1. 例子：代购

代购：顾客（Jhon）+代购商(ProxyCustomer)

对于顾客Jhon来说，买小米NOT3这件事是一定要去做的，但是自己又不想亲自去做，就找了代购商去帮忙

* 1. 静态代理

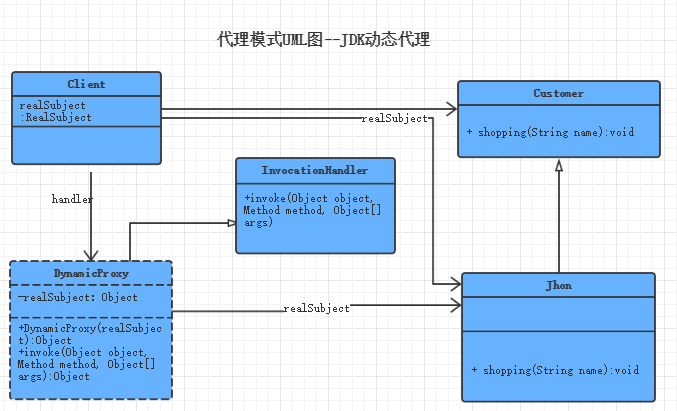


* 1. JDK动态代理

Java.lang.reflect.Proxy类来实现动态代理的，可通过它的newProxyInstance来获得代理实现类。同时对于代理的接口的实际处理，是一个java.lang.reflect.InvocationHandler，它提供了一个invoke方法供实现者提供相应的代理逻辑的实现。

使用这个方式的前提是代理类必须要实现一个接口。

字节码重组。



* 1. CGLIB动态代理

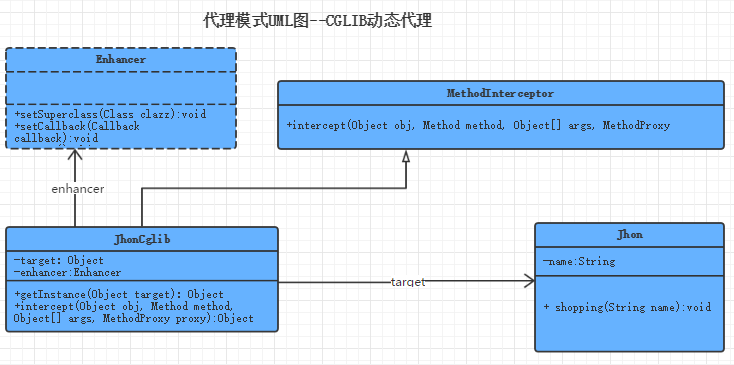
（1）不需要实现一个接口

（2）使用final修饰的类，不能使用这种方式的代理

（3）使用cglib代理会为代理类创建子类，默认重新所有的方法

（4）cglib的代理是不需要你显示的传入被代理对象的引用，是由cglib给我们new出来的，new出来的就是我们的代理类的子类

（5）jdk的动态代理是通过接口来强制转换的，生成以后的代理对象，可以强制转换为接口



完成字节码重组

1. 工厂模式

工厂模式在现实中体现很多，比如

汽车组装生产：他座位的可能是德国的工程生产、轮胎为日本生产、发动机美国生产、最后在上海组装为整车。这些都有共同的动作，加工原料、组装为部件等，也有自己特有的行为。

食品加工厂：产生各类食品

服装加工厂：产生各个品牌的服装

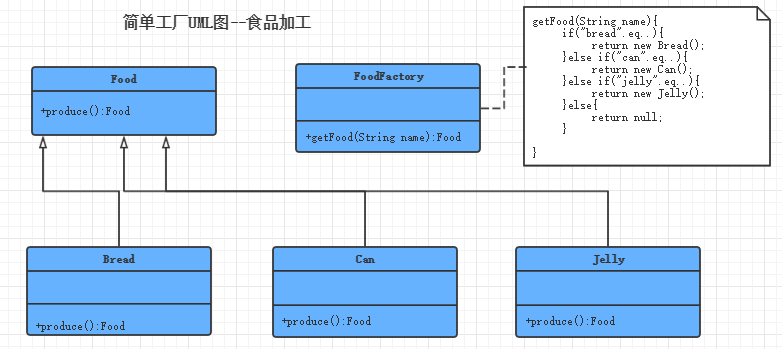
* 1. 食品加工厂

对于一个吃货来说，必须选食品加工的例子了，各类食品在现实生活中应该是由不同的工厂来完成生产的的。

我们生产面包、罐头、果冻，分别应该由面包工厂、罐头工厂、果冻工厂来完成。

* 1. 简单工厂

定义一个工厂类，它可以根据参数的不同返回不同类的实例，被创建的实例通常都具有共同的父类。



存在问题：

（1）必须知道产品的名字才能创建相应的产品

（2）工厂中存在大量的if else…判断，不利于程序的维护

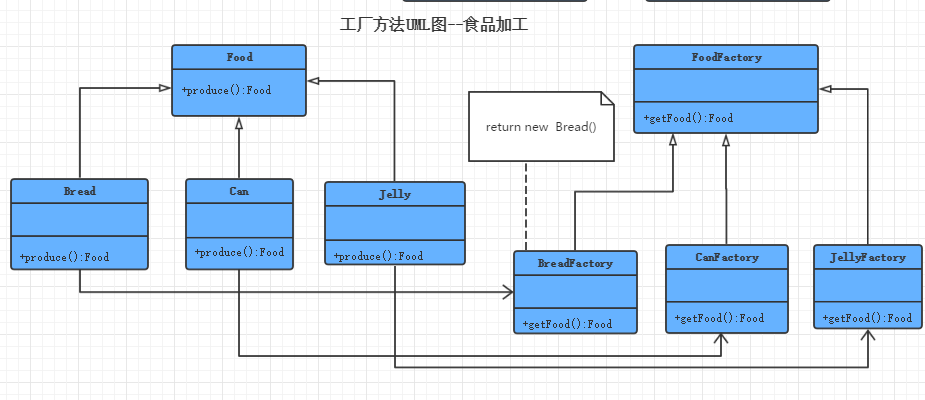
（3）如果要增加一个产品，需要在工厂中修改代码

（4）工厂类的职责过重

（5）现实中应该也不存在这样万能的工厂

* 1. 工厂方法

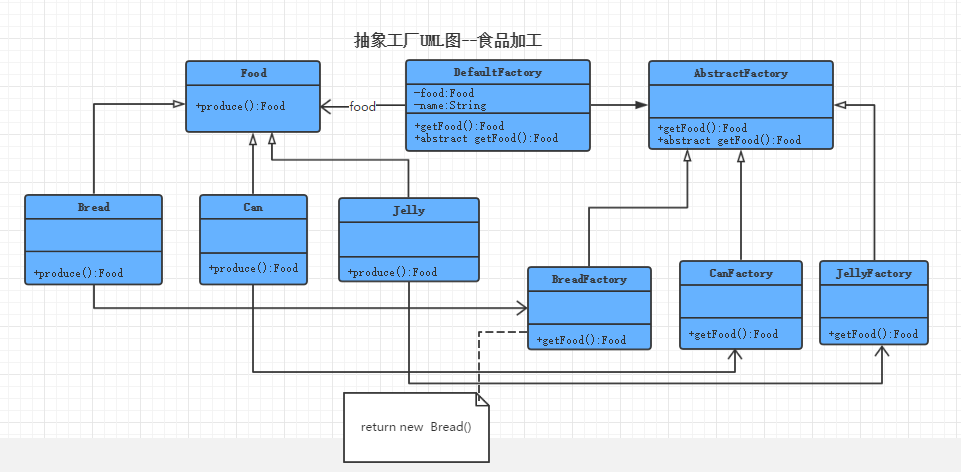
各个工厂各司其职



使用时必须知道什么产品是由哪个工厂生产的

* 1. 抽象工厂

就是前面两者的结合



1. 单例模式

（1）一个类只存在一个实例

（2）这个类可以自动实例化（获取实例的方法）

（3）这个类是可见的

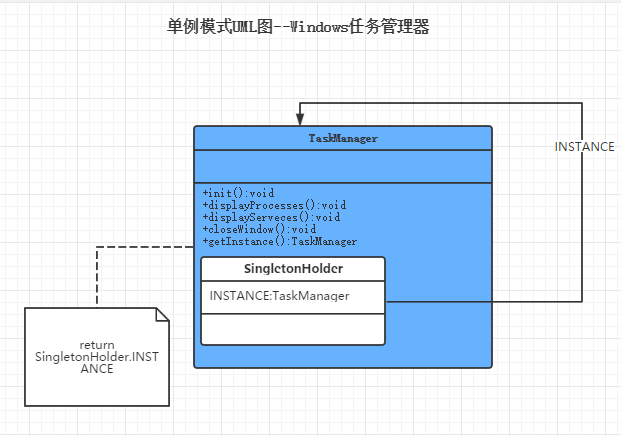
例如：

Windows任务管理器:随便你打开多少次任务管理器，他都只显示一个

在同一个教室中上课的黑板：多个人共享同一块黑板

* 1. Windows任务管理器

随便你打开多少次任务管理器，他都只显示一个，包括初始化窗口、显示进程、显示服务、关闭窗口。



1. 委派模式

特点：

（1）有点类似于中介的功能（委托机制）

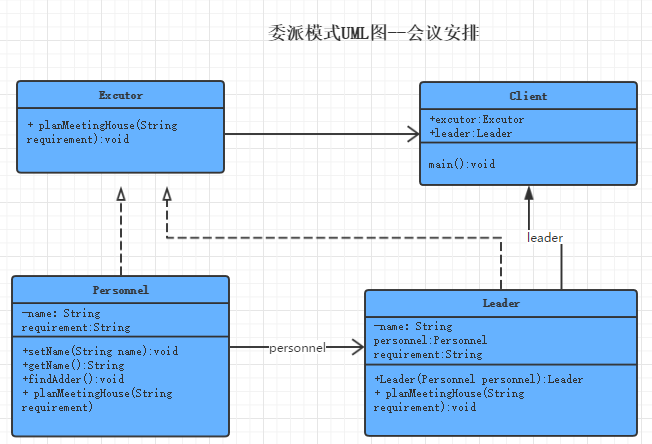
（2）委托人持有被委托人的引用

（3）不关心过程，只关心结果

例如：房屋买卖中，委托人让受委托人帮忙提交申请材料、代为签字（只赋予该事项的权利），委托人与受委托人存在平等的社会关系，只是受托人帮忙执行一些事项而已；委托人只关系资料是否已经提交在这一过程中，受托人需要双方的授权委托书。

会议安排

在公司中，有时需要开会，leader会委派行政人员去准备会议室，在这一过程中，领导与行政人员是平等的社会关系，只是职责不同。领导只关心会议室是否准备完成，具体的准备过程由行政人员去执行。



该图有误：虚线应该改为实线

1. 原型模式

使用原型实例指定创建对象的种类，并且通过拷贝这些原型创建新的对象。原型模式是一种对象创建型模式。

* 1. 工作周报

在短期内，工作周报大同小异，只有小部分存在差异，每次创建周报都需要进行完全的重写一份，非常浪费时间，现在设计的工作周报方案如下：

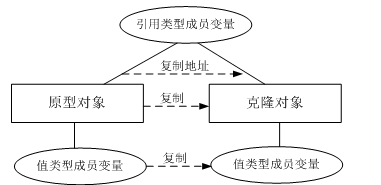
(1)除了允许用户创建新周报外，还允许用户将创建好的周报保存为模板；

(2)用户在再次创建周报时，可以创建全新的周报，还可以选择合适的模板复制生成一份相同的周报，然后对新生成的周报根据实际情况进行修改，产生新的周报。

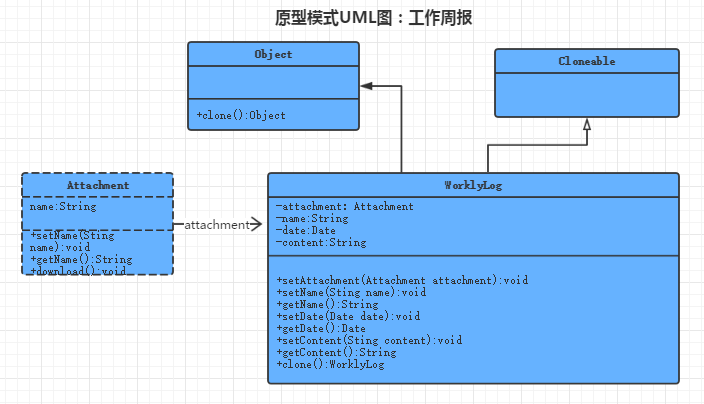
（3）周报中还可以上传附件

* 1. 浅克隆

在浅克隆中，如果原型对象的成员变量是值类型（9种数据类型：8大基本类型+String），将复制一份给克隆对象；如果原型对象的成员变量是引用类型，则将引用对象的地址复制一份给克隆对象，也就是说原型对象和克隆对象的成员变量指向相同的内存地址。简单来说，在浅克隆中，当对象被复制时只复制它本身和其中包含的值类型的成员变量，而引用类型的成员对象并没有复制，如图所示：

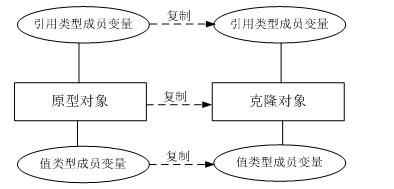


浅克隆示意图

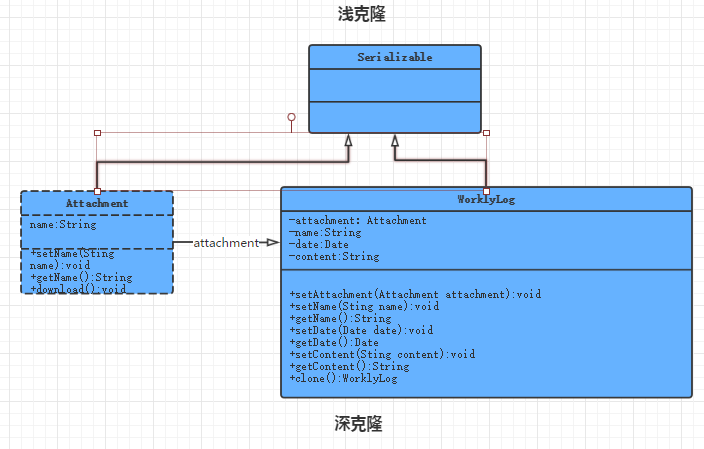


* 1. 深克隆

在深克隆中，无论原型对象的成员变量是值类型还是引用类型，都将复制一份给克隆对象，深克隆将原型对象的所有引用对象也复制一份给克隆对象。简单来说，在深克隆中，除了对象本身被复制外，对象所包含的所有成员变量也将复制，如图所示：



深克隆示意图



1. 策略模式

实现某一个功能有多条途径，每一条途径对应一种算法，此时我们可以使用一种设计模式来实现灵活地选择解决途径，也能够方便地增加新的解决途径。

* 1. 电影票打折方案

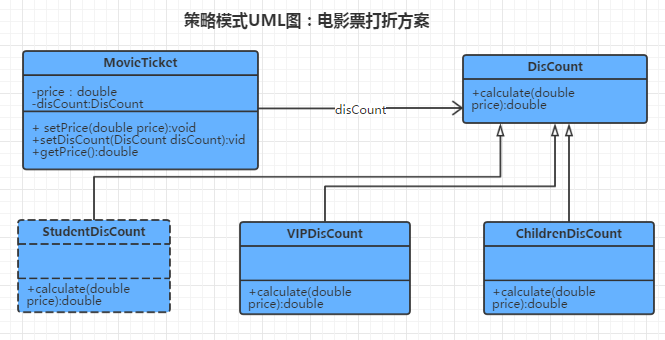
电影院的售票系统：为不同用户类型提供不同的打折方式，具体方式如下：

（1）学生凭学生证可享受8折优惠

（2）年龄在10周岁及以下的儿童可享受每张票减免10元的优惠（原始票价需大于等于20元）；

(3) 影院VIP用户除享受票价半价优惠外还可进行积分，积分累计到一定额度可换取电影院赠送的奖品。该系统在将来可能还要根据需要引入新的打折方式。

设计的UML图如下：



1. 模板方法模式

行为方式相同，产生的结果不同，适合于流水线、标准化的场景中使用。

在现实中有很多这样的场景：

请客吃饭：点单—吃东西--买单，这整个过程点单和买单没有什么变化、就是吃东西不同，每次吃的可能不一样。

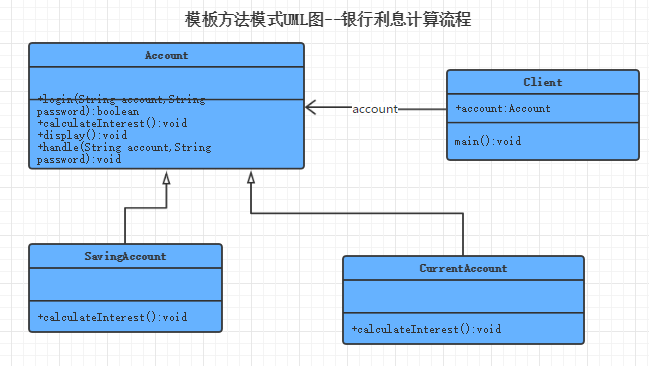
银行利息计算流程：登录系统—》计算利息（用户类型不同，计算方式不一致）--》显示计算结果

* 1. 银行利息计算流程

登录系统：根据账号和密码登录系统，如果正确，则进入，否则提示错误。

计算利息：根据用户类型不同，选择不同的计算方式（定期、活期）

显示计算结果：根据计算的结果，显示出来



1. 装饰器模式
2. 责任链模式