design pattern详述

**1.singleton**

设计模式简述

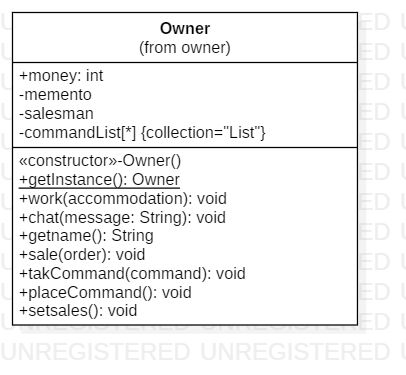
Singleton Pattern涉及到一个单一的类，该类负责创建自己的对象，同时确保只有单个对象被创建。这个类提供了一种访问其唯一的对象的方式，可以直接访问，不需要实例化该类的对象。主要意图是保证一个类仅有一个实例，并提供一个访问它的全局访问点。并且解决一个全局使用的类频繁地创建与销毁的问题。

1.1Owner部件API实现

1.1.1API描述

考虑到农场只能有一个主人，所以Owner采用singleton实现。Owner的实例生成只在第一次调用其构造函数时候使用。使用getInstance函数创建唯一实例，若唯一实例存在则返回这个实例。当唯一实例Owner要雇佣员工的时候，需要将Worker类传入，判断是否存在，不存在再将其加入，解雇时在找到这个员工将其删除。

1.1.2类图

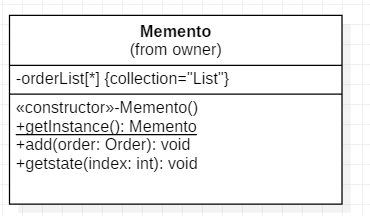


1.2Memento实现

1.2.1API描述

Menmento在程序运行期间记录所产生的订单，因而采用单例模式实现。

1.2.2类图



**2Factory Method Pattern**

设计模式简述

工厂方法模式（Factory Method）又叫虚拟构造器（Virtual Constructor）模式或者多态工厂模式（Polymorphic Factory），在工厂方法模式中，父类负责定义创建对象的公共接口，而子类则负责生成具体的对象，这样做的目的是将类的实例化操作延迟到子类中完成，即定义了一个用于创建对象的接口，由子类来决定究竟应该实例化哪一个类。其优点在于客户代码可以做到与特定应用无关，适用于任何实体类；能连接并行的类层次结构，具有良好的封装性，代码结构清晰且扩展性好。其缺点在于需要Creator和相应的子类作为Factory Method的载体，如果应用模型不需要Creator和子类存在，则需要增加一个类层次。

2.1API描述

在本例中，将雇员的创建交给Employeefactory完成，雇员的具体实现类都实现people接口，创建时传入字符串参数如“farmer”创建制定的雇员。

2.2类图



**3decorator**

**设计模式简述**

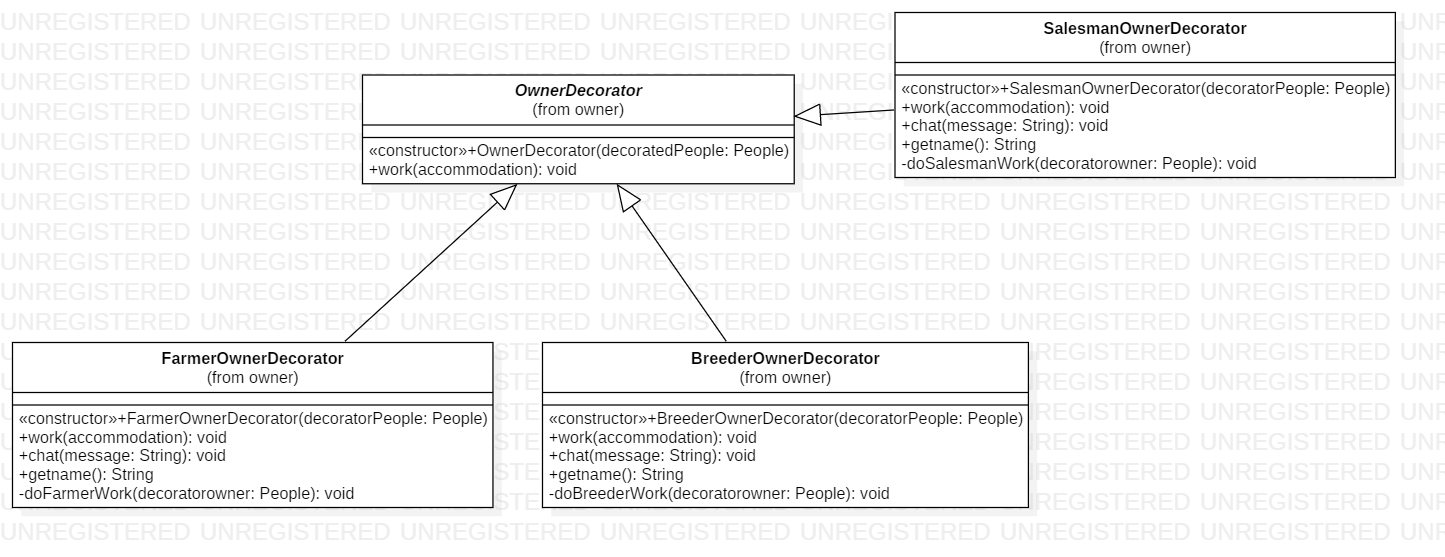
装饰器模式（Decorator Pattern）允许向一个现有的对象添加新的功能、同时又不项变其结构。这种类型的设让模式属于结构型模式，它是作为现有的类的一个包装。这种模式创建了一个装饰类，用来包装原有的类，并在保持类方法签名完整性的前提下、提供了额外的功能。

装饰器模式相比生成子类更为灵活、能够扩展个类的功能、并且可以代替继。

3.1API描述|

我们假设了一种情况、当农场中没有雇员的时候应该怎么办。按照正常情况来说、农场主在这个时候就会做起雇员应该做的事。FarmerOwnerDecorator,BreederOwnerDecorator,SalesmanOwnerDecorator类继承了People类，使得Owner在没有雇员的情况类通过FarmerOwnerDecorator（）, BreederOwnerDecorator(),SalesmanOwnerDecorator()实现装饰后描述、即可以做雇员应做的事。

3.2类图



**4. Visitor Pattern**

设计模式简述

在访问者模式（Visitor Pattern）中，我们使用了一个访问者类，它改变了元素类的执行算法。通过这种方式，元素的执行算法可以随着访问者改变而改变。这种类型的设计模式属于行为型模式。根据模式，元素对象已接受访问者对象，这样访问者对象就可以处理元素对象上的操作。

4.1API描述

Breeder实现AnimialVisitor接口 实现visitor模式，让饲养员去鸡舍访问所有的鸡。

调用函数

public void visit(ChickenCoop chickenCoop)

{

chickenCoop.accept();

}

4.2类图



**5. Command Pattern**

设计模式简述

命令模式（Command Pattern）将一个请求封装成一个对象，从而使用不同的请求把客户端参数化，对请求排队或者记录请求日志，可以提供命令的撤销和恢复功能。这是一种数据驱动的设计模式，它属于行为型模式。请求以命令的形式包裹在对象中，并传给调用对象。调用对象寻找可以处理该命令的合适的对象，并把该命令传给相应的对象，该对象执行命令。

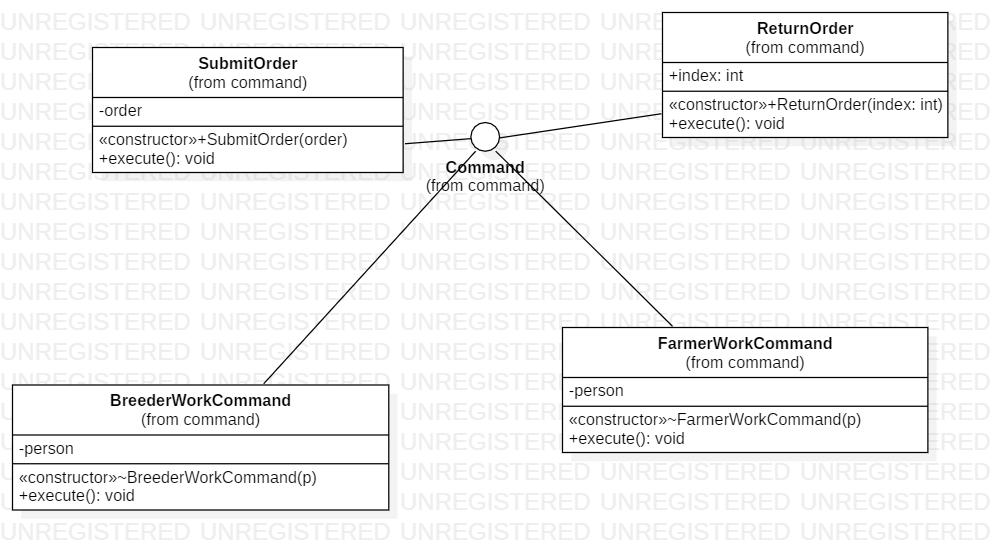
通过使用此设计模式降低了系统耦合度，容易添加新命令，同时也会使得系统中有过多具体命令类，过于繁杂。

5.1API描述

在卖出订单和退回订单 时使用了Command模式

Command为命令接口 有两种command takeCommand(加入订单)和运行命令 placeCommand Owner中调用 takeCommand(加入订单)和运行命令 placeCommand 两种命令

5.2类图



**6. Memento Pattern**

设计模式简述

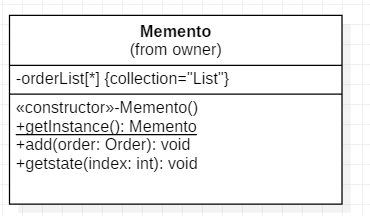
备忘录模式就是在不破坏封装的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态，这样可以在以后将对象恢复到原先保存的状态。客户不与备忘录类耦合，与备忘录管理类耦合。

6.1 API 描述

在退货的功能中实现备忘录模式。

备忘录类就是Memento类、用于存储订单列表，当退货时，传入要退货的订单编号，从Memento中取出订单，将Owner的金钱属性等恢复

6.2类图



7. Mediator Pattern

Mediator解耦多个同事对象之间的复杂交互关系。创建中介者，每个同时对象都用与中介者的交互来替代原本同事对象之间的交互。反过来client则可以通过中介者统一管理所有对象。其优点为解耦多个相似对象之间的复杂交互，从而可以独立的改变他们之间的交互逻辑，从而降低了类结构的复杂度，将多对多模式转化为多对一模式。但是原本的同事对象之间交互越复杂，中介者的逻辑就会越复杂。

7.API描述

聊天室功能实现中介者 雇员们和主人在聊天室聊天，聊天室充当中介者，每个人把要说的话发给他，聊天室展现出来

7.2类图



8.Proxy Pattern

代理模式（Proxy Pattern）中，一个类代表另一个类的功能。这种类型的设计模式属于结构型模式。在代理模式中，我们创建具有现有对象的对象，以便向外界提供功能接口。

为其他对象提供一种代理以控制对这个对象的访问。

8.1Api描述

我们在此时假设一种情况，农场主在有雇员在的时候是不会自己动手工作的，因此农场主需要指派雇员进行工作，所以农场主此时是代理，因此我们想要卖出产品时，我们需要农场主做代理指派salesman进行工作，salesman是这个操作的执行者。

8.2类图

