1 题材

本项目选用既定题目中的“欢乐农场”。目的是在农场情景下设置多个与之相关的角色与操作流程，用课程中学习的设计模式来实现其 framework。框架中含有四个主要部分：牲畜类，作物类，警报系统类，以及生产产品类。用户可以调用框架完成对“农场”中各种元素的操作，如收获和灌溉等等。

2 Design Pattern 汇总

编号	Design Pattern Name	实现个（套）数	Sample Programs 个数	备注
1	Template Method	2	1	
2	Strategy	2	1	
3	Iterator	1	1	
4	Visitor	1	1	
5	Prototype	1	1	
6	Observer	1	1	
7	State	1	1	
8	Command	1	1	
9	Memento	1	1	
10	Factory	1	1	
11	Proxy	1	1	
12	Singleton	2	1	
13	Flyweight	1	1	
14	Builder	1	1	

3 Design Pattern 详述

3.1 Template Method

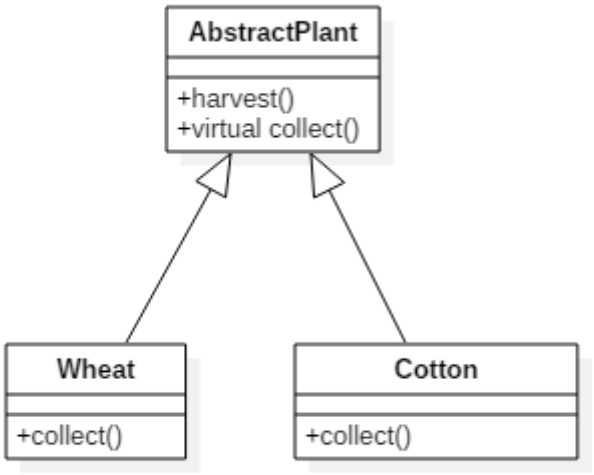
3.1.1 农作物类

3.1.1.1 API 描述

农作物类描述了农场中任何种类的农作物的属性及操作方法。其抽象父类 AbstractPlant 定义了所有作物的公有属性，包括田野面积，和作物是否成熟。用户通过实例化 PlanttsField 对象来获得一个存储了所有作物田的数组 farmField，并通过其成员函数 add()来加入新的作物田（种下新的作物）；用户也可以直接实例化 Cotton 或 Wheat 子对象，同样可以调用其中操作。

Template 设计模式实现如下：在抽象父类中定义了 harvest()操作，其中使用到一个名为 collect()的纯虚函数。函数 collect()在所有子类中都得到重载，因此 harvest 操作在每个子类中都得到特有的重新定义

3.1.1.2 类图

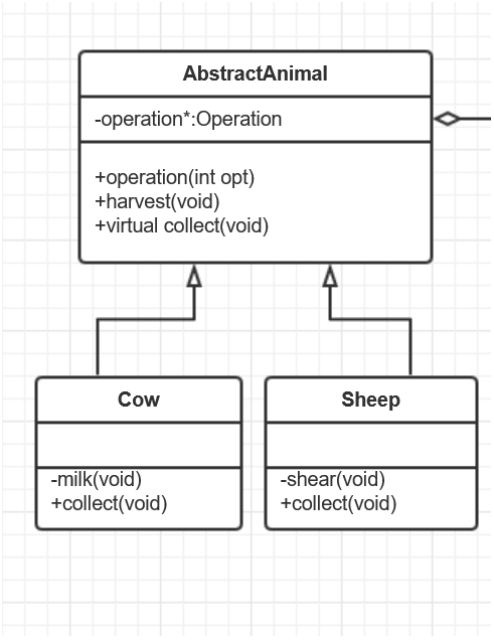


3.1.2 牲畜类

3.1.2.1 API 描述

牲畜类描述了农场中任何种类的牲畜的属性及操作方法。其抽象父类 AbstracAnimal 定义了所有牲畜的公有操作“收获”。与作物类似，收获操作中也使用到了纯虚函数 collect()，并将其在每个子类中重载。使得同样是调用 harvest 成员函数，奶牛对象会开始 milk()流程，而绵羊对象则会开始 shear()流程。用户通过实例化 Cow 或 Sheep 对象来开始对其操作

3.1.2.2 类图



3.2 Strategy

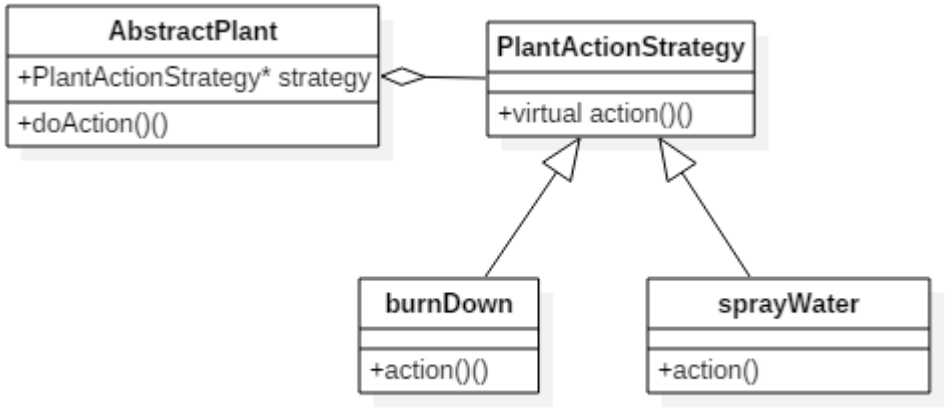
3.2.1 农作物通用操作

3.2.1.1 API 描述

农作物类内置一个 doAction 操作，其中通过指针指向不同的策略对象，以达到对某一作物对象进行灌溉或焚烧操作的目的。策略抽象父类中定义一个纯虚函数

action，在两个子类中进行重载。用户通过对某一作物对象调用 doAction 操作，并传入不同的 int 值参数，来选择并执行不同的策略。

3.2.1.2 类图

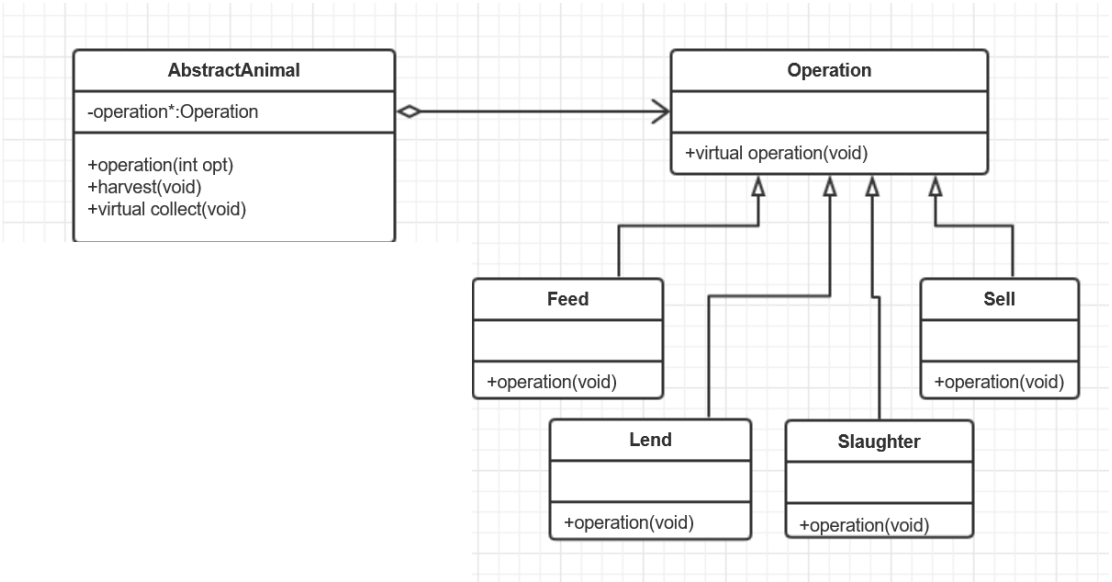


3.2.2 牲畜通用操作

3.2.2.1 API 描述

与作物类似，牲畜对象内置一个指向 Operation 类实例的指针。Operation 类中的 operation 函数经过重载能完成喂养，出租等不同操作。用户通过调用牲畜对象中的 operate 操作，传入一个 int 值来选择对应的策略

3.2.2.2 类图



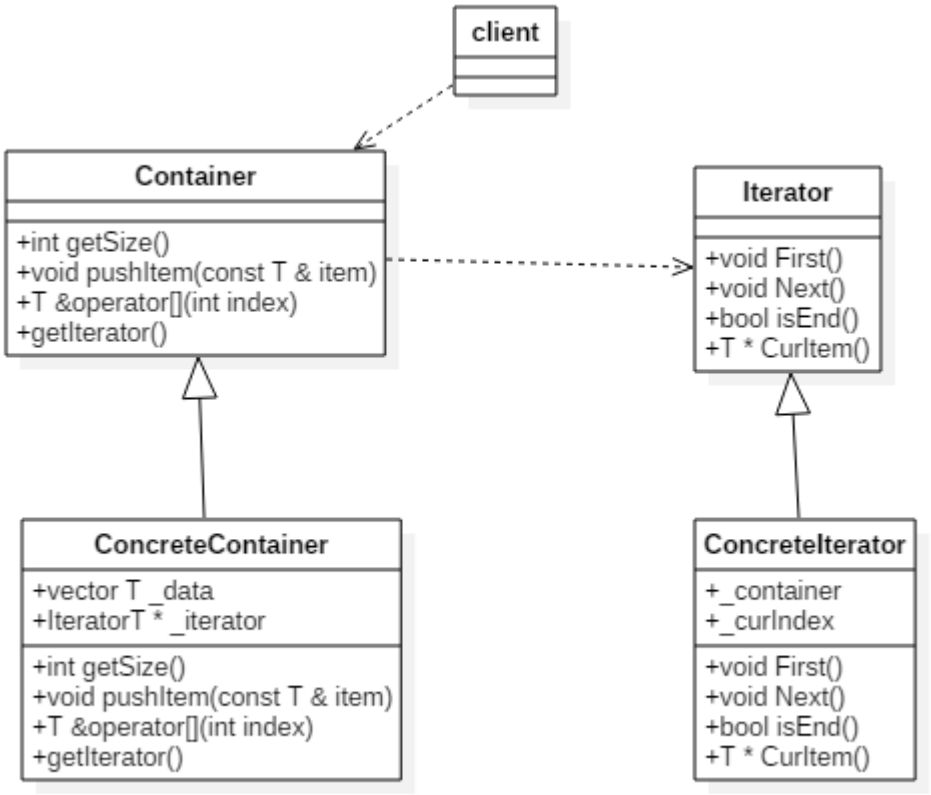
3.3 Iterator

3.3.1 作物田遍历迭代器

3.3.1.1 API 描述

Container 类构造了一个类链表的容器，在 PlantsField 中被用于存储所有的作物田对象；并且 Container 实现了迭代器模式，用户可以在构建 Plantsfield 实例之后使用 getIterator()成员函数来获取，用以迭代当前 field 实例中保存的所有作物对象。另外，PlantsField 类中的 print()函数也运用迭代器将所有作物打印输出。

3.3.1.2 类图



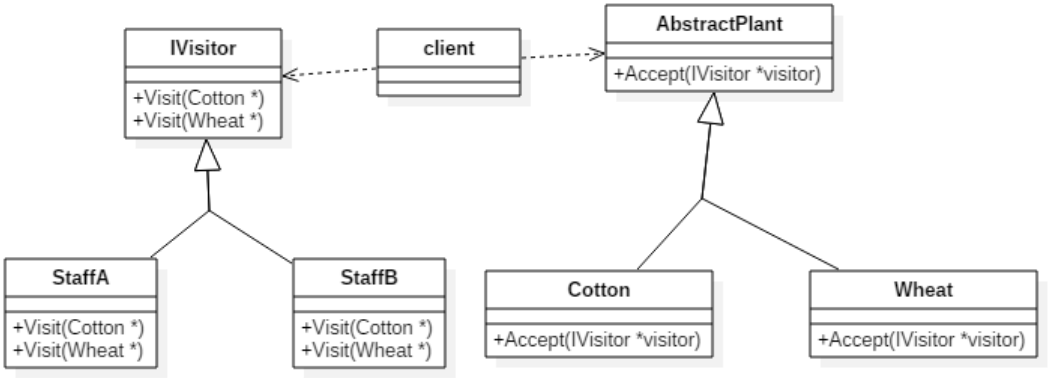
3.4 Visitor

3.4.1 作物类额外操作

3.4.1.1 API 描述

作物类中实现了 Visitor 模式，在抽象父类中增加 Accept()函数来接受 visitor 的访问，并在子类中重载以实现对不同 visitor 的支持，达到了增加新操作而不需要更改类结构的目的。用户可以继承 IVisitor 类来继续增加自定义的新操作。

3.4.1.2 类图



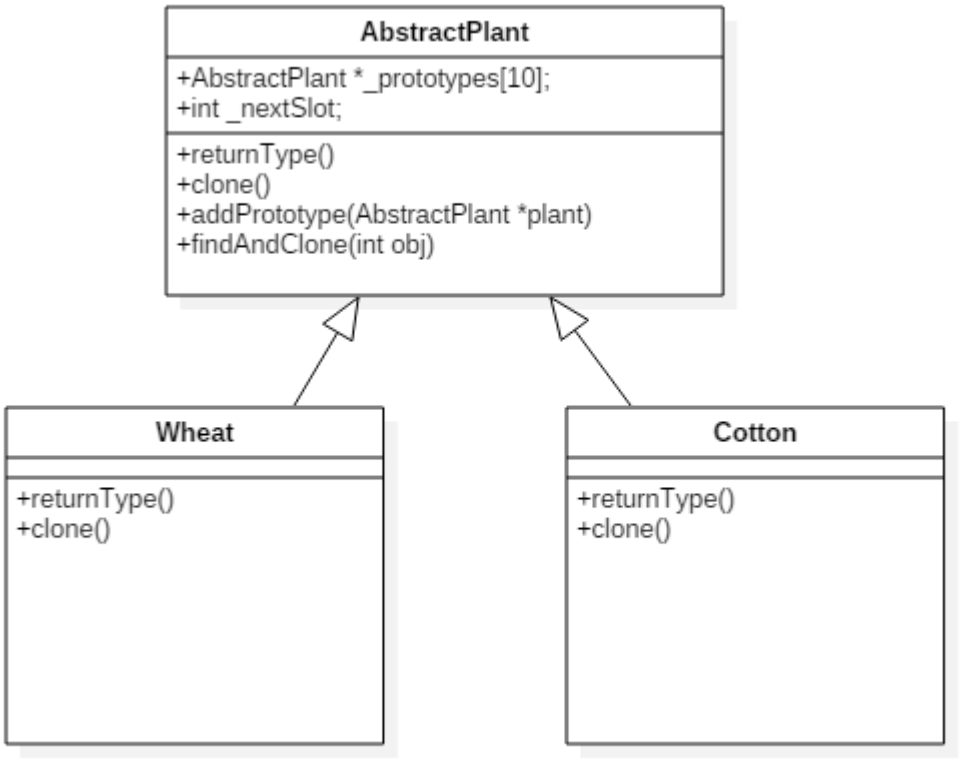
3.5 Prototype

3.5.1 新建作物田

3.5.1.1 API 描述

在 PlantsField 的成员函数 add()中，嵌套了 Prototype 设计模式，使得每次新建一块作物田时，并不是重新实例化一个对象，而是从一个可以装载 10 个原型的数组中寻找是否有现成用例，若有则从现有的原型中复制得出。此设计模式与其它部分嵌套，用户无法直接调用，但理论上可能增加实例化的速度

3.5.1.2 类图



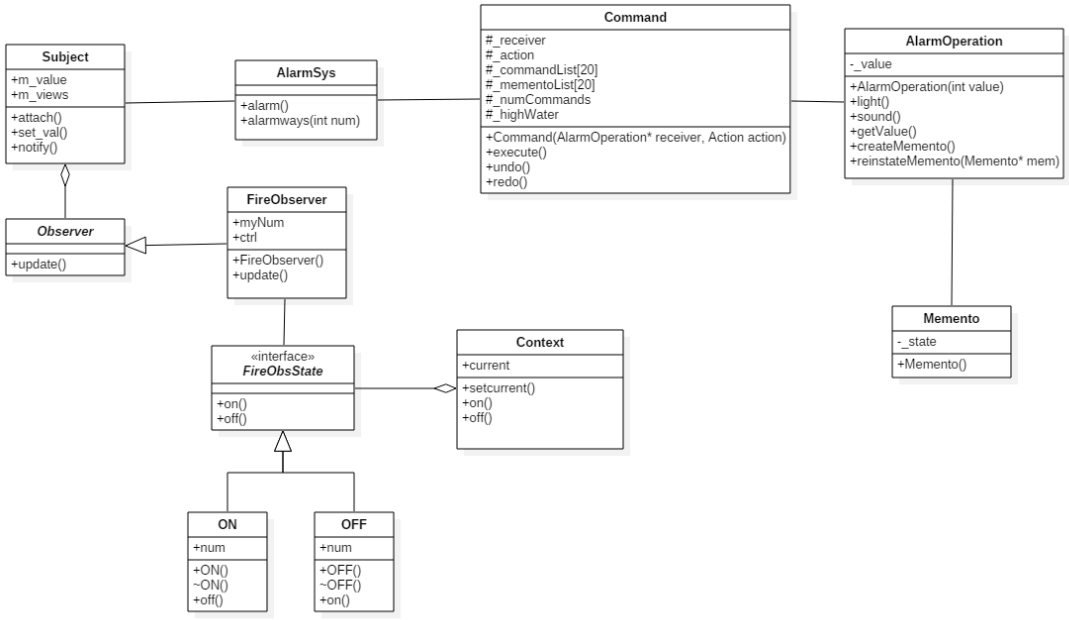
3.6 Observer

3.6.1 警报系统中央开关

3.6.1.1 API 描述

使用 Observer 模式创建农场警报系统。系统设置一个中央开关，各警报器监听开关状态，当打开时即鸣警。用户实例化 Alarm 类之后，即可调用 Context::on()/off()函数控制开关。此时警报器 Alarm 0~4 即会根据中央开关的状态，做出相应的行为。此设计模式与 Command 模式相结合，内嵌了控制台操作界面，因此用户只需实例化后即可进行操作。

3.6.1.2 类图



3.7 State

3.7.1 警报系统报警模式

在警报系统中可对警报器设置报警模式。根据 Memento 类中的_state 属性，警报器在中央开关打开之后，会有闪烁灯光和扬声器播音两种反应模式。该设计模式与 Command 模式结合，用户在实例化系统类之后即可通过控制台指示，调整警报器的 state。模式的类图整合在 3.6.1.2 中

3.8 Command

3.8.1 警报系统命令输入

实例化警报系统对象之后，将自动要求用户输入命令（数字字符），来选择对系统的操作或修改。模式类图整合在 3.6.1.2 中

3.9 Memento

3.9.1 警报系统命令撤销、重做

整合在命令系统之中，实现了对命令的撤销和重做。输入命令时按照提示，能观察到状态的回溯。模式类图整合在 3.6.1.2 中

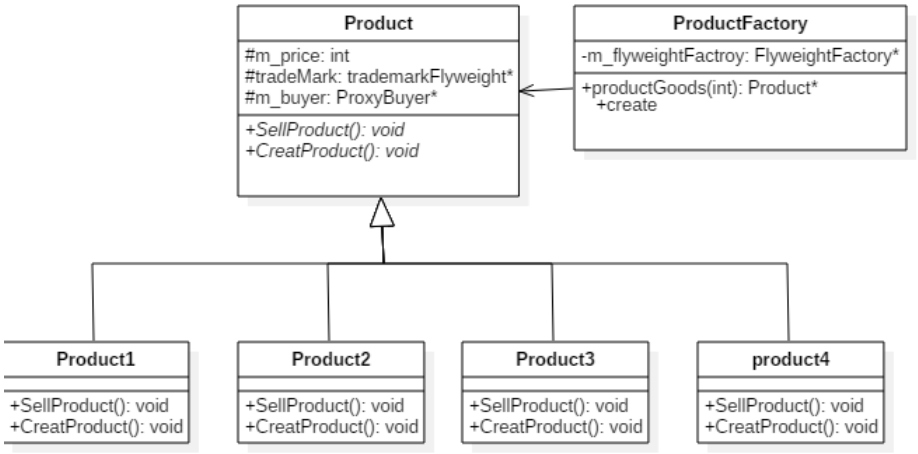
3.10 Factory

3.10.1 生产产品

3.10.1.1 API 描述

在农场工厂中生产多种产品。实例化 ProductFactory 类之后，即可调用其成员函数 productGoods()来获取一个产品，通过更改参数(整数值)来从既定的产品列表中选择一个，返回一个 Product 类型的对象。

3.10.1.2 类图



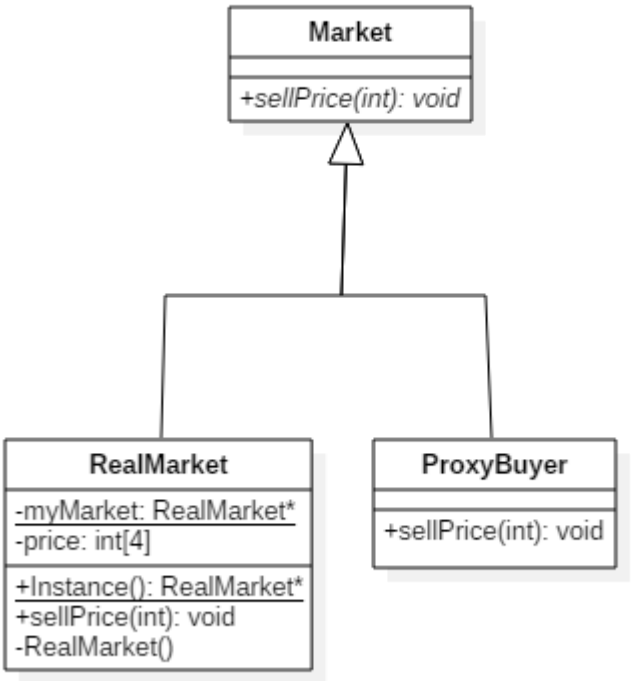
3.11 Proxy

3.11.1 查看市场价格

3.11.1.1 API 描述

封装在产品类之中。用户可以通过调用 Product 中的 SellProduct 方法来试图将某个产品对象卖出。此时产品会通过一个 ProxyBuyer 对象试图与市场连接。若市场没有被创建，则会实例化一个市场。ProxyBuyer 会返回当前产品在市场中的价值。

3.11.1.2 类图



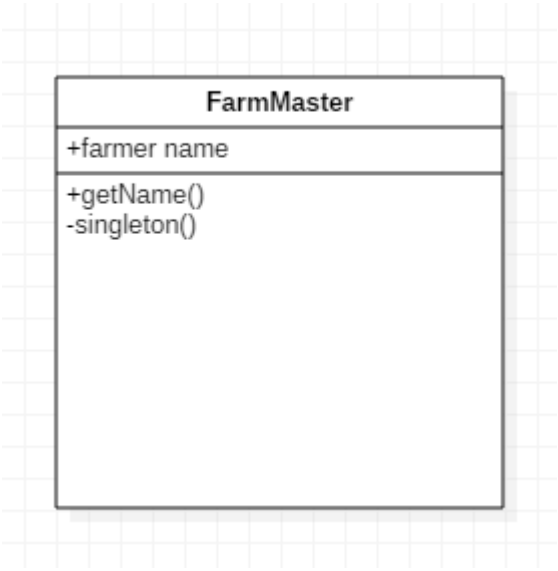
3.12 Singleton

3.12.1 农场主人

3.12.1.1 API 描述

一个唯一且不可创建多个实例的农场主人角色。用户可简单地试图实例化多次来检测其是否符合 Singleton 设计模式

3.12.1.2 类图

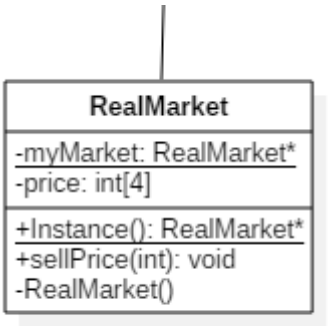


3.12.2 产品出售市场

3.12.2.1 API 描述

在产品——市场关系中，市场只能存在一个实例，以防止产品有多个售价的情况出现。此例嵌套在 Proxy 模式的 Market 类之中。

3.12.2.1 类图



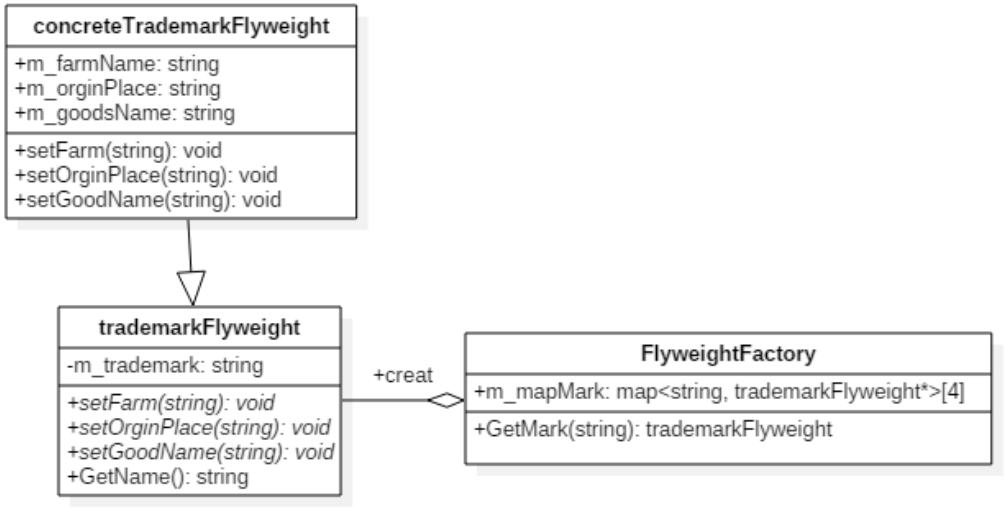
3.13 Flyweight

3.13.1 产品标签

3.13.1.1 API 描述

每个产品有一个“标签”属性，描述了该产品的大部分信息。本系统中使用 Flyweight 设计模式来管理既定的几个标签对象，使所有产品都能引用，且避免重复创建。

3.13.1.2 类图



3.14 Builder

3.14.1 新建建筑

3.14.1.1 API 描述

农场中可兴建建筑，且可以根据需求用有限的部件创建不同的实例。用户可以直接调用 BuildFarm 函数，其返回一个 BuildingBuilder 对象。再调用该对象内部的选择性构造函数 configure…()，即可创建不同类型的 Buiding 实例。

3.14.1.2 类图