单例模式

单例模式：保证一个类仅有一个实例。

要做到这一点，核心办法是把构造函数设置为私有的。

**public** **class** **ClassMaster** {

**private** String id;

*// 班主任名称*

**private** String name;

**private** String gender;

**private** **ClassMaster**() {

}

}

把构造函数设置为private，就意味着，除了ClassMaster自己，其它任何类都不能实例化ClassMaster对象。

*// 唯一实例*

**private** **static** ClassMaster instance = **new** ClassMaster();

在ClassMaster类中定义一个ClassMaster类型的变量，赋值为new出来的自己的实例。

（要注意：必须用static修饰符，否则会造成死递归的严重的错误）

访问实例：

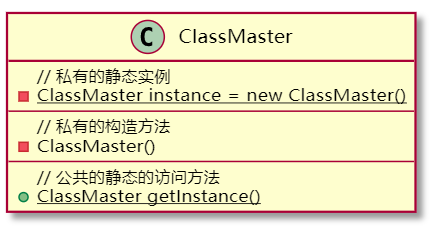
*// 外部类可以通过这个方法访问唯一的实例*

**public** **static** ClassMaster **getInstance**() {

**return** instance;

}

单例模式图：



Spring中的单例

类变量使用@Autowired注解，能够实现自动注入实例对象。

任何自动注入实例对象，默认只有一个实例对象，是单例的。

**@Autowired**

**private** UsersService usersService;

简单工厂模式

解决 **代码重复、耦合紧密** 的问题。

所谓工厂，顾名思义就是生产物品的。程序的工厂，是生产实例对象的。实现简单工厂，需要两个步骤：

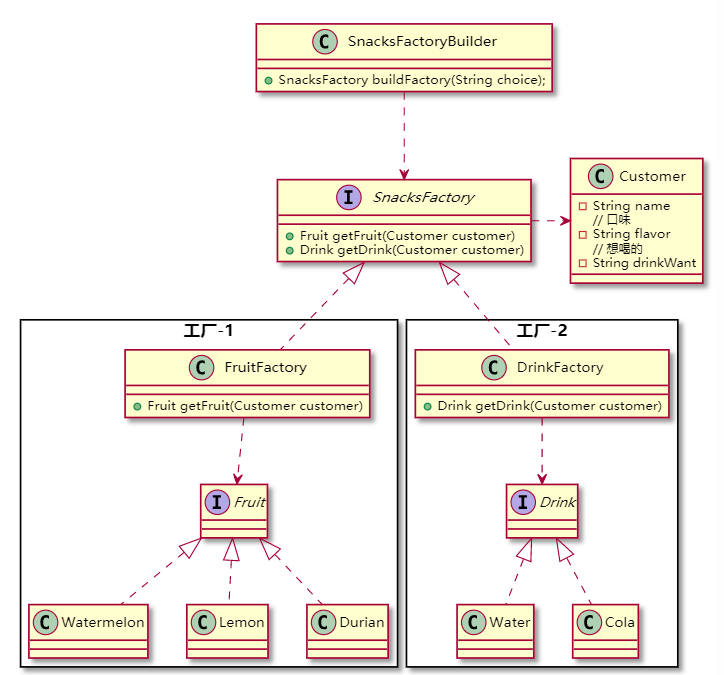
1.从具体的产品类抽象出接口。记住Java强调面向接口编程。意味着工厂应该生产 **一种** 产品，不应该生产某 **一个** 产品。

2.把生产实例对象的过程，收拢到工厂类中实现。

抽象工厂模式

简单工厂适合创建**一种**对象。但有时候会遇到复杂得问题，需要创建一个**系列**、**多种产品**得时候，简单工厂模式就不太适用了。

简单工厂得主要作用是把多个产品抽象，使用一个工厂统一创建；那么抽象工厂得主要作用是吧多个工厂也进一步抽象。



1.工厂接口

2.工厂得工厂

观察者模式

1.观察什么

观察者模式 的核心是要知道观察什么，什么对象发生变化了需要发出通知？

在天气项目中，显然，天气信息是核心，天气发生变化了，需要通知大家都知道。所以，我们先抽象出天气信息对象。

**public** **class** **WeatherData** **extends** **Observable** {

*// 城市*

**private** String cityName;

*// 时间*

**private** String time;

*// 温度*

**private** String temp;

}（去掉了所有属性的setter方法，初始化完毕以后就不能改了。）

天气信息类WeatherData **继承**了Observable类。Observable类是Java提供的，继承了就表示是核心的、需要被观察的类。

2.数据变化后发起通知

时间和气温是监听的重要信息，所以增加一个新的方法来专门处理：

**public** **void** **changeTemp**(String time, String temp) {

**if**(time == **null** || temp == **null**) {

*// 输入数据为空是有问题的，不处理*

**return**;

}

*// 与原数据不同，说明发生了变化*

**if**(!time.equals(**this**.time) || !temp.equals(**this**.temp)) {

*// 标记变化*

**super**.setChanged();

**this**.time = time;

**this**.temp = temp;

*// 发出通知，参数是额外的信息*

**super**.notifyObservers("温度变化已通知");

}

}

在changeTemp（）中，如果天气数据与原来不同，则会标记变化并发出通知。

父类Observable提供的方法setChange（）就是标记被观察者对象发生了变化。

父类Observable提供的方法notifyObservers（）就是发出通知；如果需要发送额外的信息，在参数中传入信息对象，可以是**任意对象**，需要自己根据具体的需求场景而定。

（如果不想发送额外信息，可以写为super.notifyObservers(null)）

3.谁接收通知

需要了解天气的类，就是接收通知的类，通知叫作观察者。

观察者需要实现Observer接口，也是Java提供的，实现此接口表示作为观察者。

**public** **class** **WeatherObserver** **implements** **Observer** {

**private** String name;

**@Override**

**public** **void** **update**(Observable o, Object arg) {

System.out.print(**this**.name + "观察到天气变化为：");

System.out.print(o.toString());

System.out.println(" " + arg);

}

}

作为观察者，实现Observer接口后，要自己实现update（）方法，方法签名是接口中定义好的，属于固定写法。

\*第一个参数就是被观察者对象，被观察者对象都需要继承自Observable

\*第二个参数就是额外的信息，具体说就是调用super.notifyObservers()是传入的参数

象；传入什么对象，arg的值就是什么对象。

（如果传入的是super.notifyObservers(null)，那么这里arg值就是null，注意避免空指针异常。）

Update（）方法的作用就是接收通知。实际上，系统在super.notifyObservers()发出通知后，立即调用所有观察者的update（）方法，完成通知的过程。