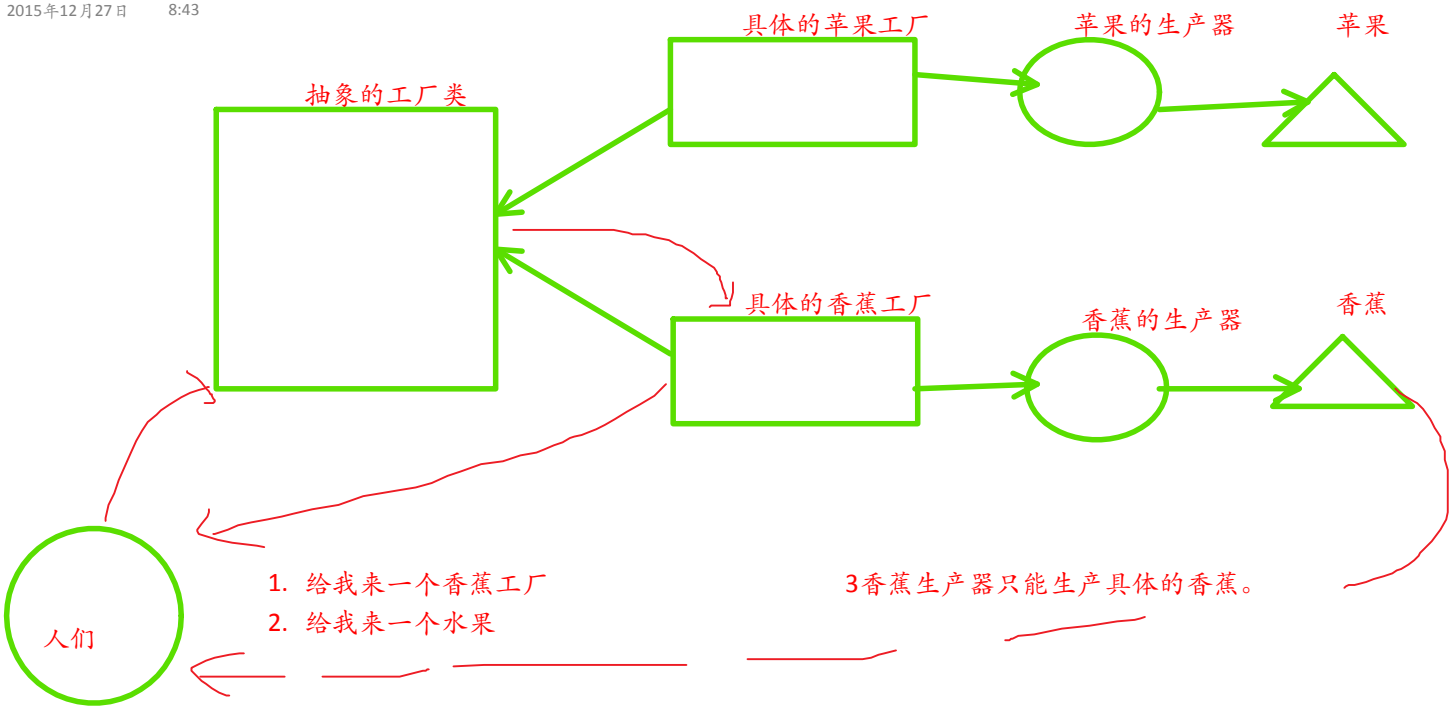
# 1 工厂方法模式

2015年12月27日 8:43

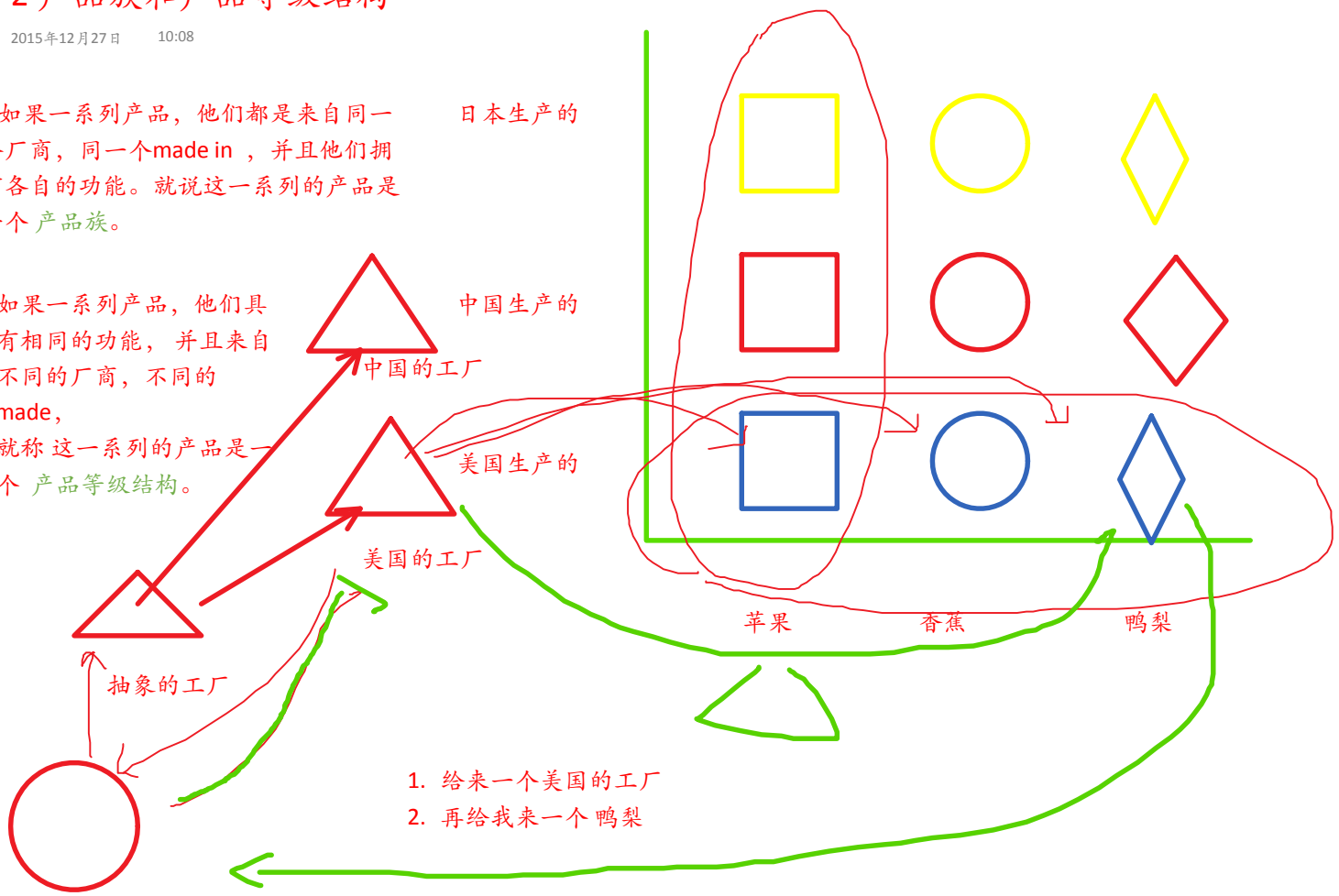


# 2 产品族和产品等级结构

2015年12月27日 10:08

1 如果一系列产品，他们都是来自同一个厂商，同一个made in ，并且他们拥有各自的功能。就说这一系列的产品是一个 产品族。

2. 如果一系列产品，他们具有相同的功能，并且来自不同的厂商，不同的made，
就称 这一系列的产品是一个 产品等级结构。

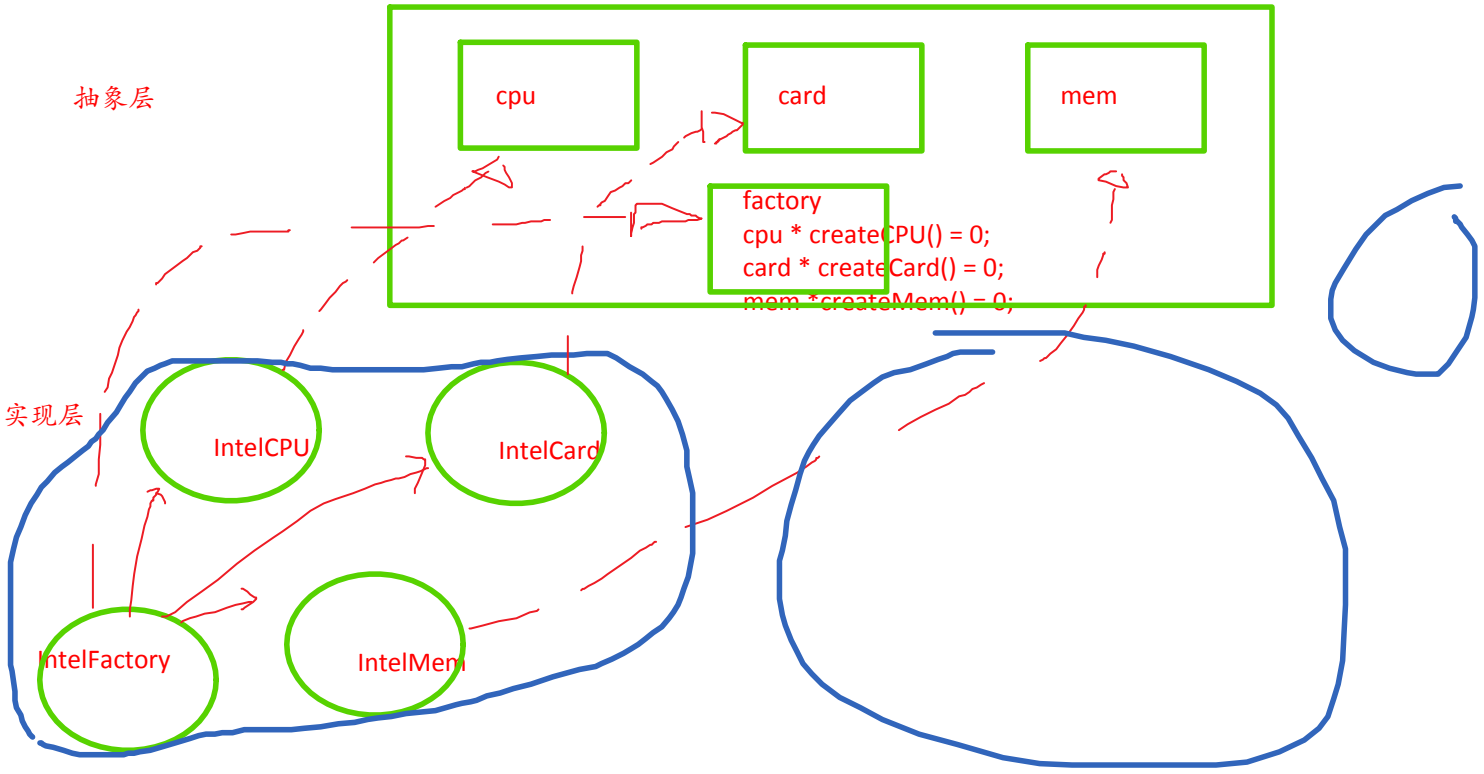


# 3 抽象工厂电脑组装

2015年12月27日 11:02

1. 组装一台intel系列电脑

高层业务逻辑



# 4 工厂三兄弟

2015年12月27日 11:43

简单工厂模式+“开闭原则”=工厂方法模式

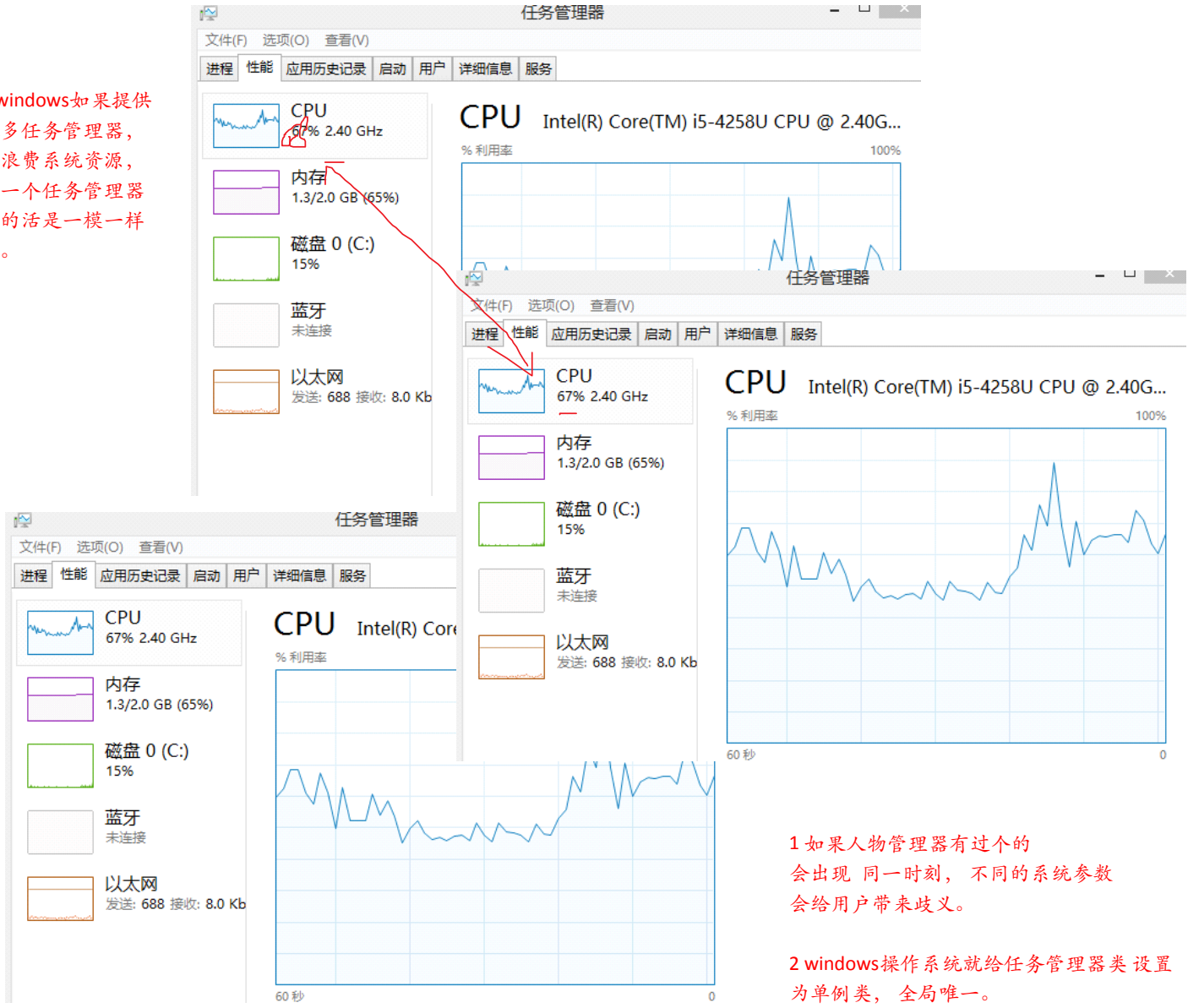
工厂方法模式+“产品族”=抽象工厂方法模式

简单工厂模式（规模较小的模型）
工厂方法模式（中等）
抽象工厂方法模式（复杂的产品等级和产品族）

# 5 任务管理器 单例模式

2015年12月27日 11:58

1 windows如果提供很多任务管理器，会浪费系统资源，每一个任务管理器干的活是一模一样的。




1 如果人物管理器有过个的会出现 同一时刻， 不同的系统参数会给用户带来歧义。

2 windows操作系统就给任务管理器类设置为单例类， 全局唯一。

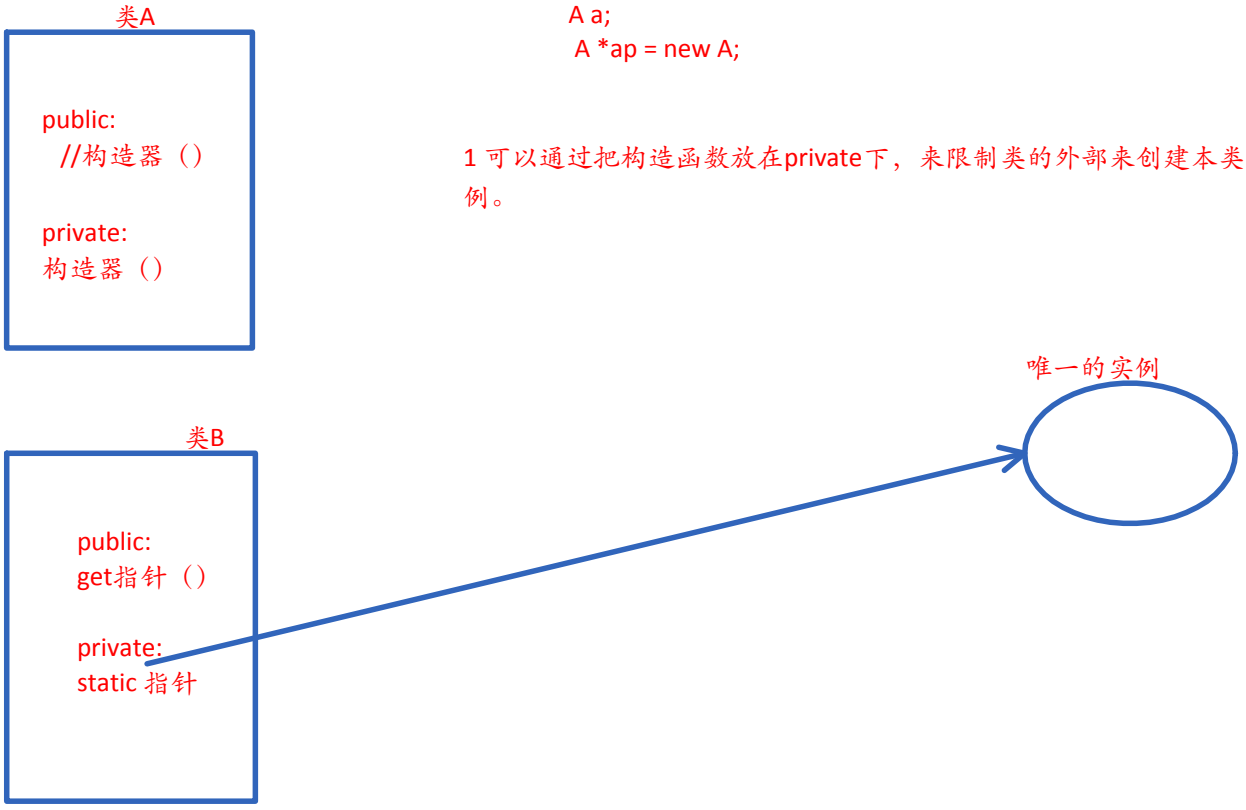
# 6 单例模式

2015年12月27日 14:52

在创建一个类的对象的时候，一定会调用一个类的构造函数。

A a;
A *ap = new A;

1 可以通过把构造函数放在private下，来限制类的外部来创建本类的实例。



# 7 当懒汉式遇见多线程

2015年12月27日 15:24

单线程的处理流程

```
main()
{
  func1();
  func2();
}
```

func1();

func2();

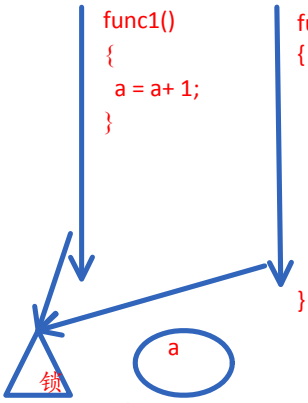
func1（）函数，功能是对a加1；

func2（）函数，功能是如果a==1 让a加1；

```
main()
{
  int a;
  func1();//在一个线程中执行
  func2(); //在另一个线程中执行。
}
```

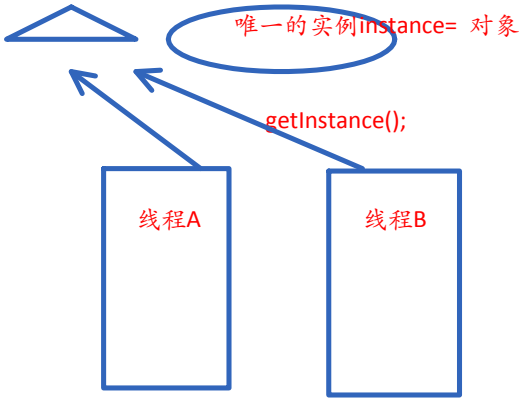
```
func1()
{
 a = a+ 1;
}
```

```
func2()
{
    if (a == 1)
    {
        a = a +1;
    }
    else {
        pass;
    }
}
```




1. func1 函数执行到a=a+1, 计算出a+1 的结果是2.正要把2 赋值给变量a的时候，
2. func2正好判断if(a==1), 进入if语句块， 准备执行a=a+1;
3. func1 将2 赋值给a，此时的a已经是2；
4. func2 再执行a=a+1, a就变成了3；

锁



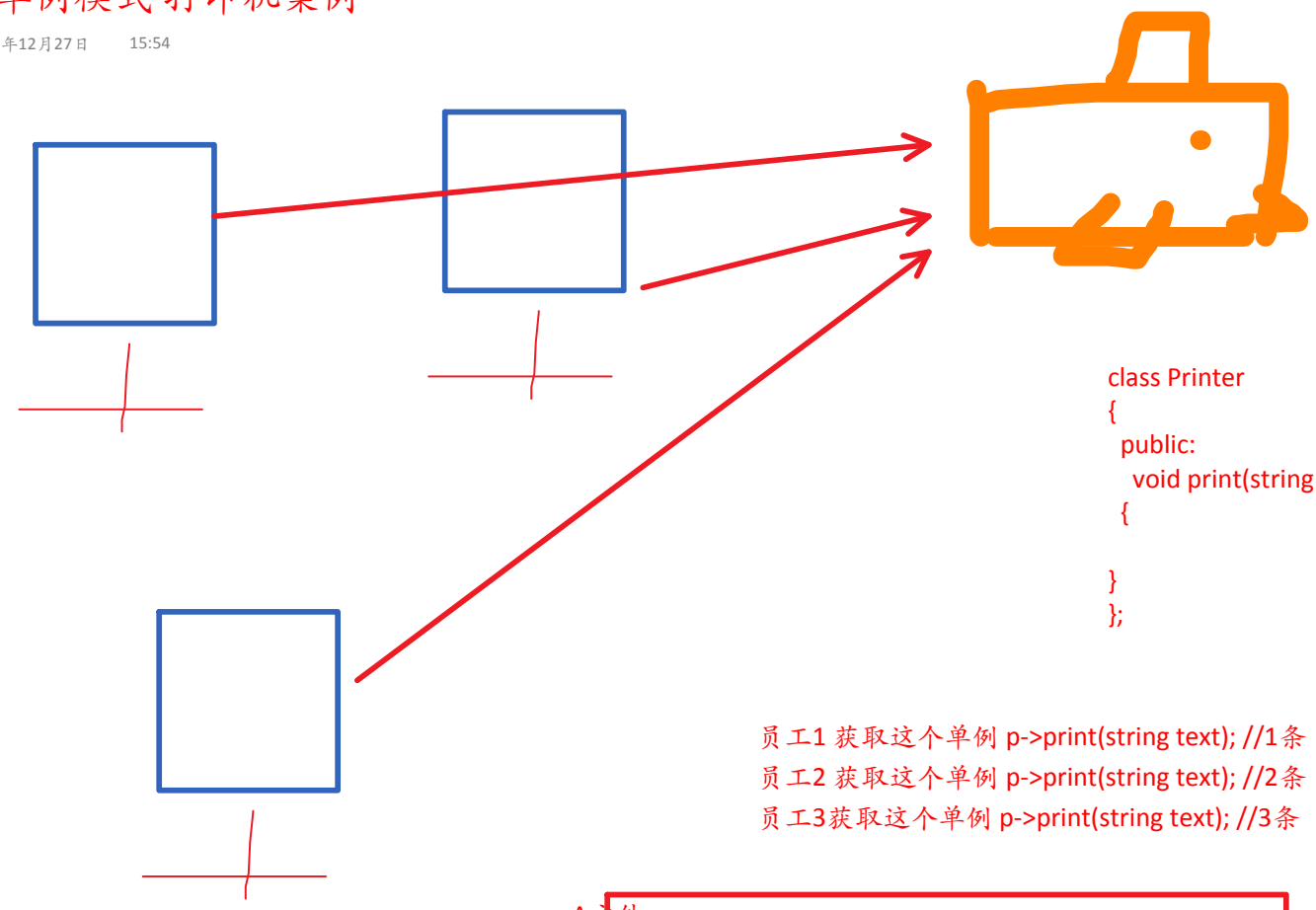

1 线程A第一次调用getInstance()方法
发现instance是NULL， instance = new SingelTon();
当刚刚new SingelTon 实例之后，刚要给instance赋值.

2 线程B调用了getInstance（）；判断instacne 此时依然是NULL
也会调用new SingelTon();

3 这样A。B 分别都会调用new，出现了两个new，违背了单例。

# 8 单例模式 打印机案例

2015年12月27日 15:54



```
class Printer
{
 public:
   void print(string text)
 {

 }
};
```

员工1 获取这个单例 p->print(string text); //1条
员工2 获取这个单例 p->print(string text); //2条
员工3获取这个单例 p->print(string text); //3条

A文件

# 9 代理模式

2015年12月27日 16:39