用面向对象的语言来写面向过程的程序。

语言的本身只会提供面向对象特性的方便性。

写程序是需要用面向对象的指导思想。

面向对象：封装，继承，多态

类：对 对象方法和属性的抽象

对象：是世界上存在的真实物体

实例：就是 java new 出来那个东西，实例存在计算机里，对象真实存在。

类，实例 和 对象 是什么关系

类为计算机与人之间的沟通而存在。对象是人类的语言，只有人类能懂，实例是计算的语言，只有计算机能懂。对象抽象成类，计算机把类构造成实例。

写代码的时候要不要多写注释

答案是不要！

在重构里面，代码坏味道，注释就是代码坏味道，如果你在开发的过程中，需要用注释的阐述一段代码，那就说明你这段代码是有问题！什么时候需要写注释，在敏捷开发里面，代码即文档。
需要描述一些特殊的设计意图

简单工厂，工厂方法，抽象工厂，建造者 有什么区别，都属于构造系列的设计模式

为了单一职责，我们使用了简单工厂，但是增加一个产品的时候，我们需要在工厂中修改代码，这样又违反了开闭原则（对扩展开发，对修改闭合）。为了不违反这个开闭原则，我们引入工厂方法。但是判断逻辑又被实现到 Client 类里了，不同的是，Client 所面对的是产品的工厂，而不需要关心产品的创造过程。设计模式没有最正确的，只有最适合的。在做软件需要的时候，要懂得假设我们的客户是猪！通过抽象接口约束系列产品的生产。这个时候，抽象工厂就出来了，它是用来产生系列产品。有了系列产品以后，客户发现演习顺序错了。所以我们引入了一个导演，让他来主导演习。那导演是依赖各个不同的表演建造者来构造演习的。

简单工厂：一个工厂对应多个产品

工厂方法：一个工厂对应一个产品

优点：封装性好，客户不需要关注产品艰辛的创造过程，只需要知道这么一个工厂就行了

扩展性好，增加产品的时候，不需要修改工厂逻辑

缺点：会导致工厂泛滥

抽象工厂：一个工厂生产一系列的产品

优点：封装性，上层模块，不需要关注这个产品的创造过程。

约束产品族的非公开状态。

缺点：在系列产品中，添加一样产品

构造者和工厂模式有什么区别

意图不一样，工厂所关注的整个产品产生过程，而建造者它更关注的是产品的组合。Show 产品包含两个组件，一个是开车，一个模特摆 pose，而建造者所关注的就是这两个产品的组合。工厂模式对应的是产品的工厂，而建造者模式对应的是车间，它们所关注的产品的粒度

是不一样的。

1. 首先要谈一场恋爱
2. 要找准学习的高效率点
3. 和同年龄的人比，你比他多什么东西

设计模式的课程会有三个阶段

1. 举相应例子给大家讲基础模式，每个模式都是怎么样的
2. 设计模式的 Pk
3. 结合项目来讲设计模式

建造者 vs 工厂方法

首先我们在组装这个超人的时候，发现少给了一个标志属性。而这种

错误是因为我们没有把组装流程标准化所导致的！利用建造者模式，封装产品的组装流程，让装配工人能够按照标准化的流程来组装超人。我们发现要一个超人，居然需要问装配工人要，有点扯！正常来说，要产品应该是问工厂拿的，所以我们引入工厂方法。工厂管理超人生产的数量，装配工人（SuperManWork）它只关注超人是怎么按照正常的流程生产出来的。

明天晚上上完课之后，唠嗑，话题是本科毕业之后要不要考研

单例模式

保证整个系统只有一个实例对象。

有什么场景会用到单例：

1. 数据缓存对象
2. 工具方法

实现方式是通过对构造函数私有化来实现对象单例。

存在问题：

1. 多线程问题，会导致实例被创建多次
2. 加入了同步关键字后，会引发效率问题
3. 加入双检锁，缩小锁的范围

第一个线程调用获取实例的方法时，执行到 new singleton()这句话，但是只执行了 1,3 两个步骤，这个时候，_instance 已经不为 null 了，但是构造器没有初始化完成，然后第二个线程进来，由于此时 _instance 已经不为 null 了，所以在判读非空的时候就直接返回了，这时如果线程 2 使用这个构造器没有初始化完成的 singleton 对象的话，就会报错！

解决方案：

1. 在变量中声明 volatile 关键字，让线程强制读取主内容中的变量，而不去读取变量副本。
2. 通过饥汉的方式来解决
3. 使用静态类来保存 instance

在软件设计当中，我们应该遵循一个法则，简单，正确。不为了炫耀技术来实现需求，只有简单，正确的实现方式，它生命周期才是最长，因它可维护性高。所以选择懒汉还是饥汉的方式是根据具体的场景定的。

单例演变过程

我们是为了节省内存空间，让这些空间的数据可以在一个系统的全局范围内共用。但是我们在使用单例的时候，出现了多线程问题，问题解决多线程问题，我们使用了锁，但是锁又是会损耗性能。在这种情况下，我们使用各种奇淫技巧来平衡这个问题，包括饥汉模式和懒汉

模式，平衡这个性能和空间的问题。

设计模式讲究的就是一个平衡，应用架构讲究也是平衡，做架构师其实也在做平衡。

模板方法

首先我们在实现需求的时候，发现子类有重复的代码，而且这个代码都是一个标准流程。这个时候我们引入模板方法设计模式，然后重复的部分抽象到父类中。

模板方法的优点：

1. 模板方法是了封装不变，扩展可变部分
2. 提取公共部分，便于代码的维护
3. 行为由父类控制，子类实现

类，抽象类，接口之间关系

类是为了把对象转换成计算机和人都懂得一个表述，封装对象的属性和行为，面向对象就只要有一个特征就行了，那就是封装。就是说只要有封装这个特性，面向对象就成立。

继承是为了减少代码而存在的，封装和继承之间有一个互斥的关系，那就是继承是打破封装的。

抽象类就是为面向对象的特性中的继承而生的，它是一种特殊的类。抽象类也是为了描述对象的。

接口和抽象类有什么关系

接口是为抽象行为而生的，是一种特殊的抽象类。

我们在设计时候，什么时候用接口，什么时候用抽象类。

接口有一个特点，它偏向于行为，就是说把关注的是行为，而抽象类它关注的是一类东西。举个例子，严老师是一个人，人可以一个抽象类，抽象出人的一个共有的属性和行为，而且接口是面向行为的，吃饭接口，走路接口。

抽象类还有一个功能，就是对接口的描述。

继承是纵向扩展的，而接口是横向扩展。这个可以理解为，抽象类只能继承一个，而接口可以实现多个。说白了就是为了解决单继承的问题。

毕业前要不要考研

1. 如果你毕业了以后，从事的是应用层面的软件开发的话，那就不要考研了。应用层面的包括业务系统，网站，app。
2. 如果你想从事底层开发，包括算法，网络协议，操作系统，人工智能这类比较底层

工作了以后要不要考研，工作了以后这个考研，是按需考研

1.职位提升需要，做顾问需要有光环。这种考研一般都是不着急的，等你有这个一个机会时候，你再考虑去考研。选择国外大学，听起来比较高大上。

2.技能瓶颈，需要考研。这种需要有针对性地选择专业较强的学校。