JAVA高级工程师

面向对象原理课程

导师简介

- 我来自你们所说的A厂,在所谓的该枪毙的年龄35+ 之后用一年时间入职了现在的公司.你们在大厂之路上的辛酸和痛苦我都曾亲身经历.

- 我做过小公司马仔,也担任过国有银行CTO,与诸位是同路人,从业十余年,从未离开过一线,从服务千万用户的金融系统到服务二十亿用户的互联网系统,软件研发之路上的各个领域都能跟大家做很好的探讨

- 作为大厂的认证面试官,在实际招聘中见过各型候选人,他们的失败和成功经历都可以给你们最有力的指导.

- 作为每年最大购物节的高可用保障一号位,我可以给你们带来最新鲜的资讯和最严谨的工程训练

课程结构

- 一、类的基本结构
- 二、类之间的关系
- 三、面向对象的核心特性

类的定义

具有相似点的事物就是同一类

关键词

共性沉淀

能力沉淀

架构沉淀

赋能

场景面试题

从“一句话需求”中识别出类

面试题串讲

问题:客户在天猫小店扫码购物,请给出你要建模的类

- 普通答案:

Shop --- 商铺

Goods --- 商品

Custom --- 客户

——一句话评价:没有从一个软件工程师的视角来回答问题

面试考察点

从“一句话需求”中识别出类,是考察候选人是否达到软件工程师思维的重要方式

通用这类问题的回答技巧: Who When What Where How Result

高分答案

- Who --- 客户. (附上英文类名)
- When. --- 下单时间、支付时间、超时规则
- What --- 商品
- Where --- 天猫小店
- How. --- 二维码、扫码设备、支付验证策略、支付工具、支付金额
- Result --- 订单、营销策略、售后条款、异常挽回策略

面试题复盘

时间、地点、人物、过程、结果、异常

类的要素

属性和方法

类具有的特性

属性最小化原则。即“属性不可拆分”

看一段代码


```java
package example;
/**
 *
 * @author senyang
 * @version $Id: SHop. java, v 0.1 2020年03月07日 11:44 PM senyang Exp $
 */
public class Shop{
  private String name;

  private String address;
}
```

属性不可拆分

Address还可以拆分为:省、市、区、街道...

劝分不劝合的原则

```
private String province;
private String city;
private String Street;
…
```

单一化的拆分,给予系统底层精细化维度管理的能力

类具有的功能

方法单一化原则。即“一个方法只做一件事”

错误的方法设计

一个方法中,将修改与删除写在了一起

```
public Shop modify(Shop newShop, Boolean isDelete){
    if(!isDelete){
        return shopDao.update(newShop);
    }else {
        shopDao.delete(newShop);
        return null;
    }
}
```

违背了“方法单一化原则”

1、参数复杂
2、容易误调用造成故障

正确的写法


```
public class Shop{
    private String name;
    private String privice;
    private String city;
    private String Street;
    private ShopDao shopDao;
    public Shop modify(Shop newShop){
        return shopDao.update(newShop);
    }
    public void delete(String shopId){
        shopDao.delete(shopId);
    }
}
```

你知道UML中怎样表达一个类呢?

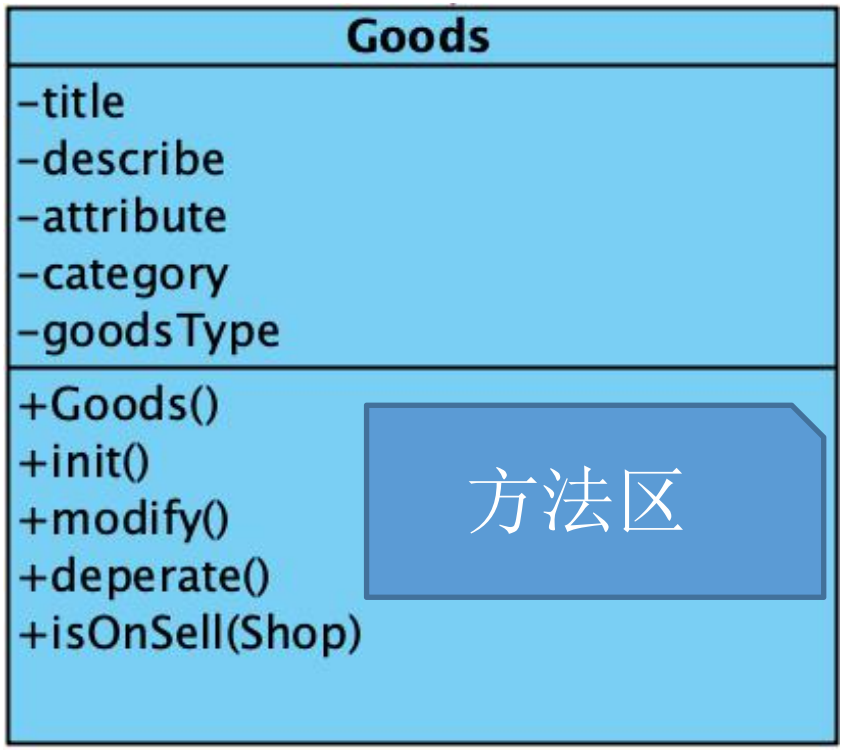
类的UML表达

答案要点1:

分三个区域的框图

类的UML表达

类的UML表达



UML中,类之间关系的表达

学习目标:
知道有哪些关系
知道UML中怎样表达这些关系
能正确识别类之间的关系

泛化关系

子类与父类: 继承

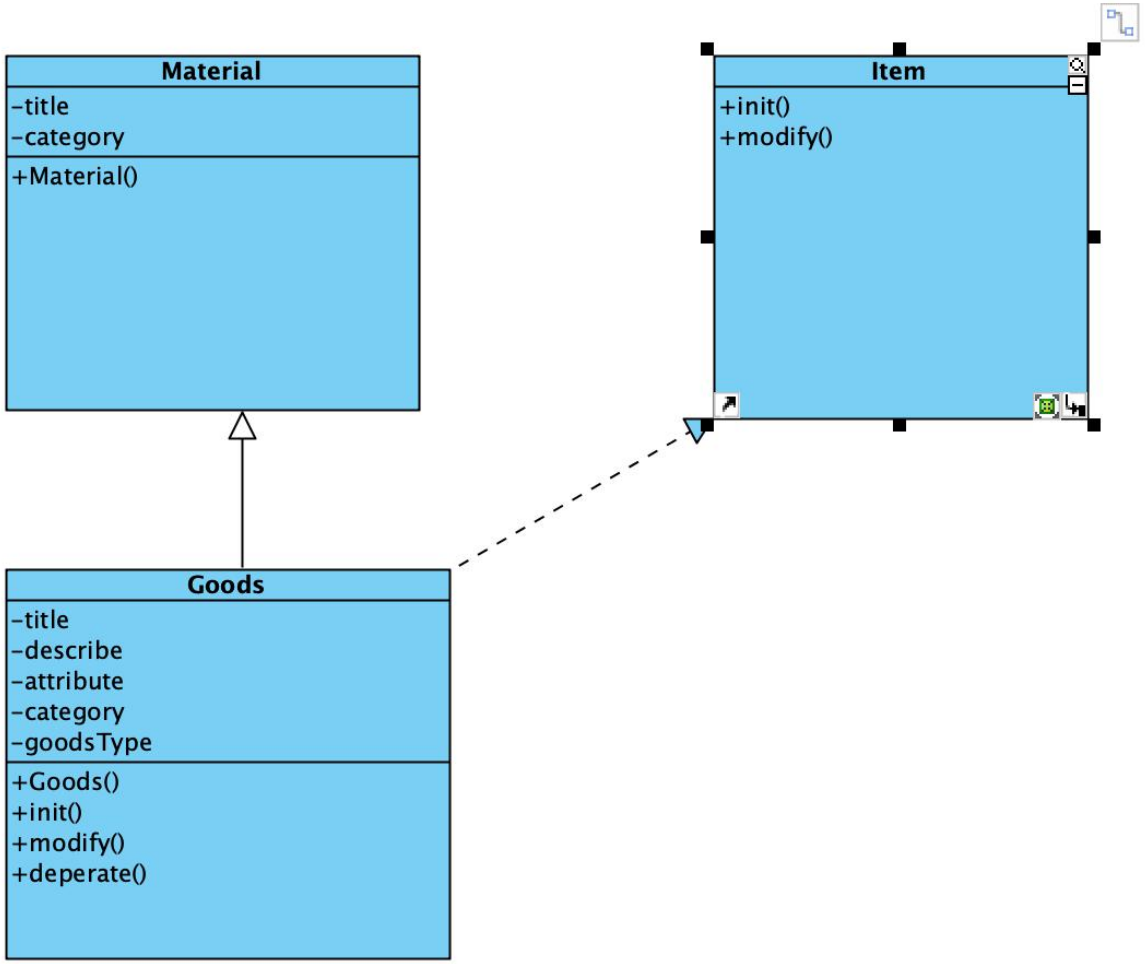
泛化关系

接口与接口:　　继承/扩展

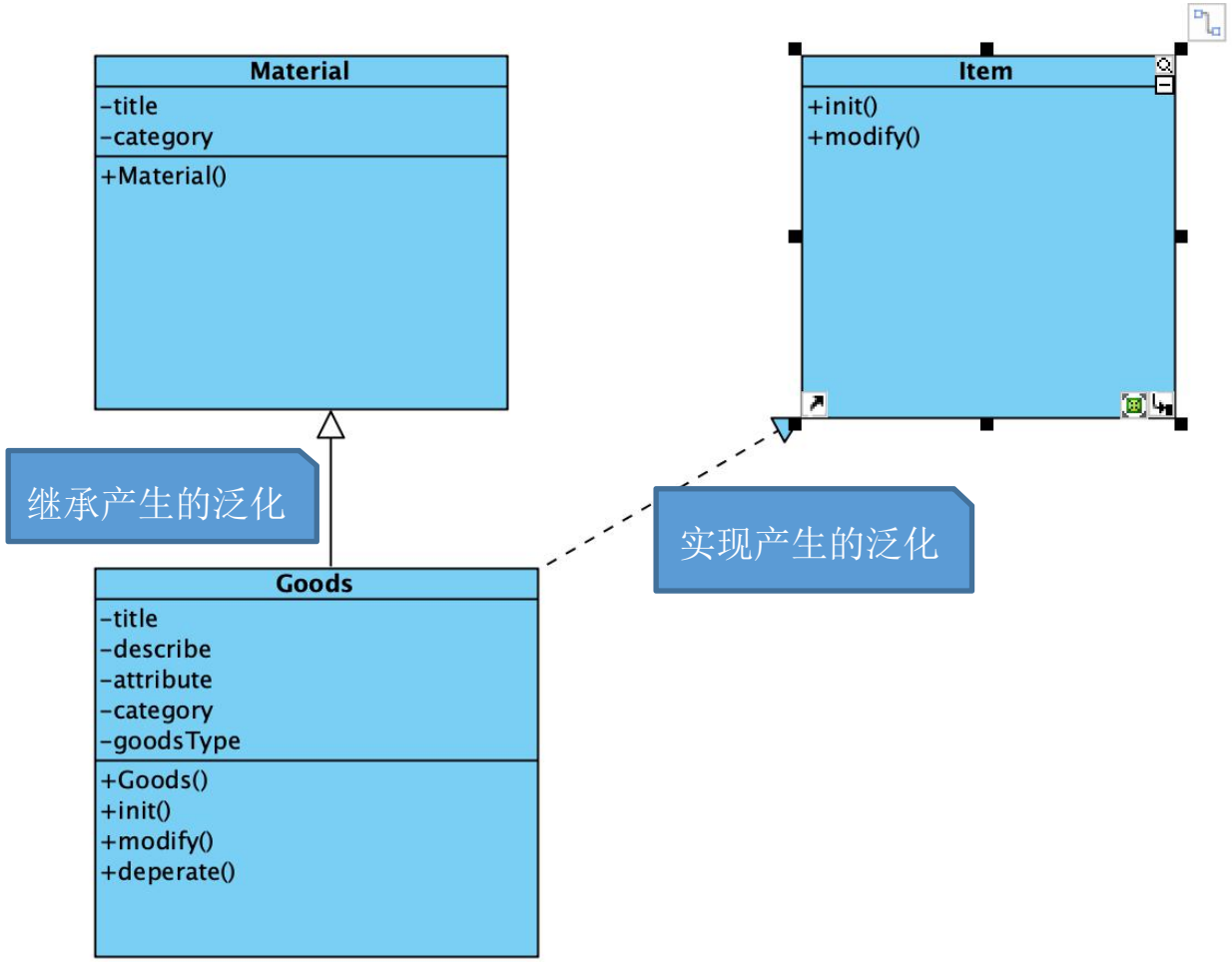
泛化关系

类与接口:　　实现

类的UML表达



类的UML表达



类的UML表达

- 类的继承

```
public class Students extends Person{
    private int age;

    public Students(){
    }

    public void printName()[
        System.out.println("直接输出属性: " + age) ;
    }
}
```

类的UML表达

- 实现一个接口

```
public class Person implements Item{
    private String name;
    private int age;
    static{
        System.out.println("父类静态代码段");
    }
    public Person(){
        System.out.println("父类构造方法");
        age = 100;
    }
    public String getName() {return name;}
    public int getAge() {return age;}
    public void setAge(int age) {this.age = age;}
    @Override
    public void init() {}
    @Override
    public void modify(){}
}
```

类的UML表达

- 扩展一个接口

```
public interface CanList extends Item{
    public int getIndex();
}
```

类的UML表达

- 继承一个父类同时实现一个接口

```
public class Students extends Person implements Item f
    private int age;
    static{
        System.out.println("子类静态代码段");
    }
    public Students(){
        age = -3;
        System.out.println("子类构造方法");
    }

    public void printName(){
        System.out.println("直接输出属性" + age);
    }
}
```

类本身支持继承,为什么还需要接口呢?他们有什么区别?

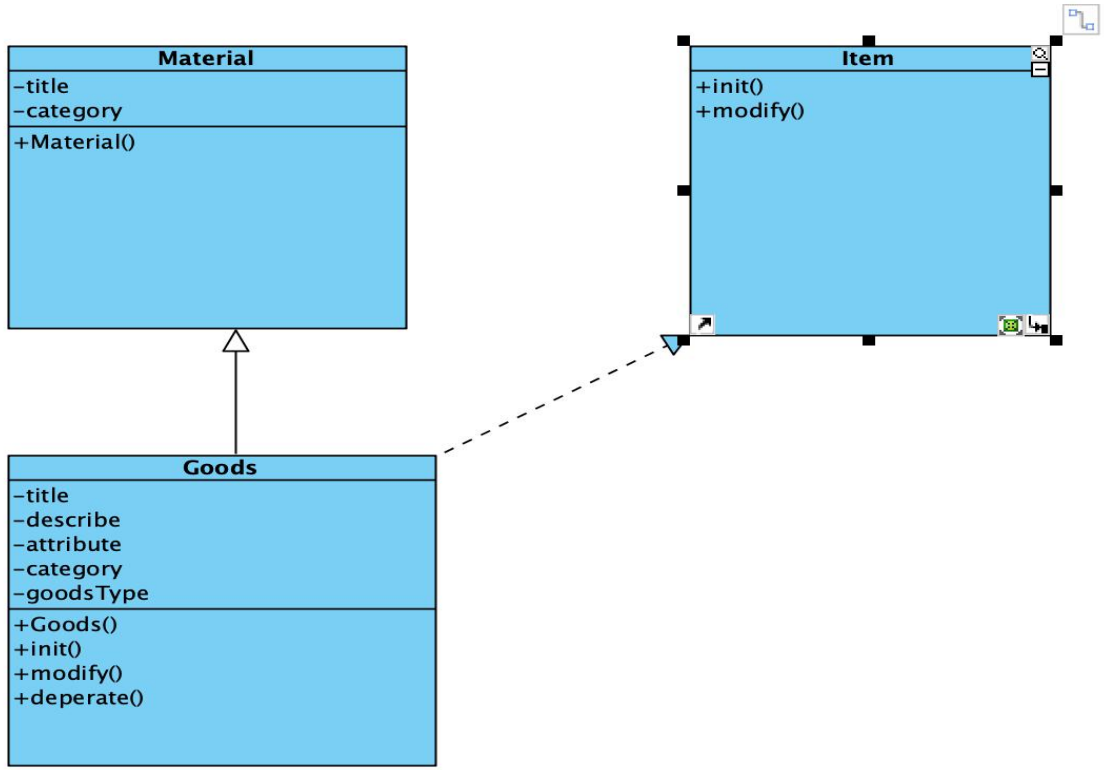
频度:中

难度:三星级

通过率:中

类继承的目的在于公共特征的传承;接口的实现是为了宣告一个能力的约定

类的UML表达



一家提供到家服务的电商,它的服务人员同时具备商品和人的属性,怎么办?

频度:中

难度:三星级

通过率:中

面试题:一家提供到家服务的电商,它的服务人员同时具备商品和人的属性,怎么办?

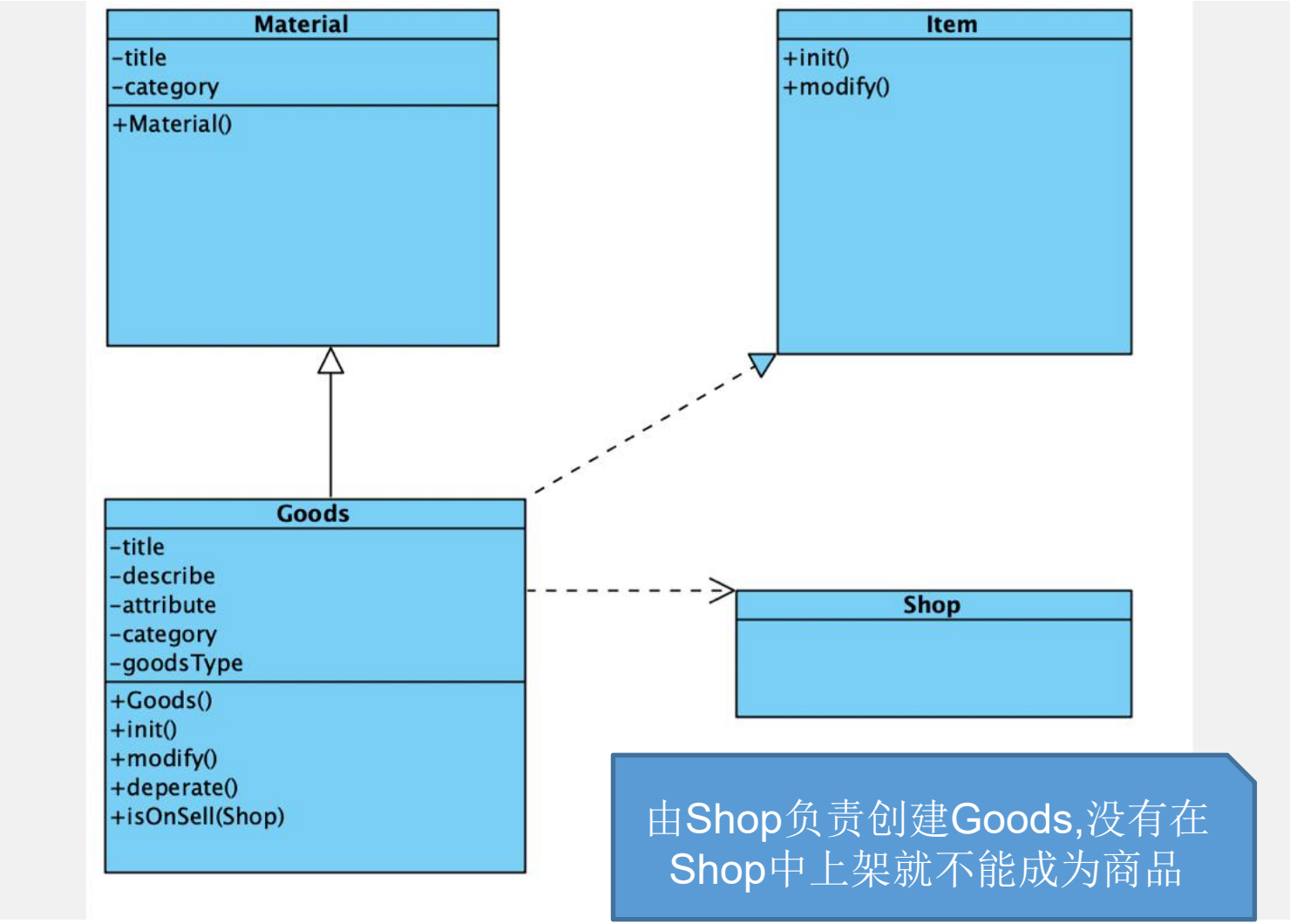
Java中,只能有一个父类,可以采用实现接口的方式叠加其他父属特性

C++中,直接使用多重继承

- 依赖关系

- 对于两个相对独立的对象，当一个对象负责构造另一个对象的实例，或者依赖另一个对象的服务时，这两个对象之间主要体现为依赖关系。

类的UML表达



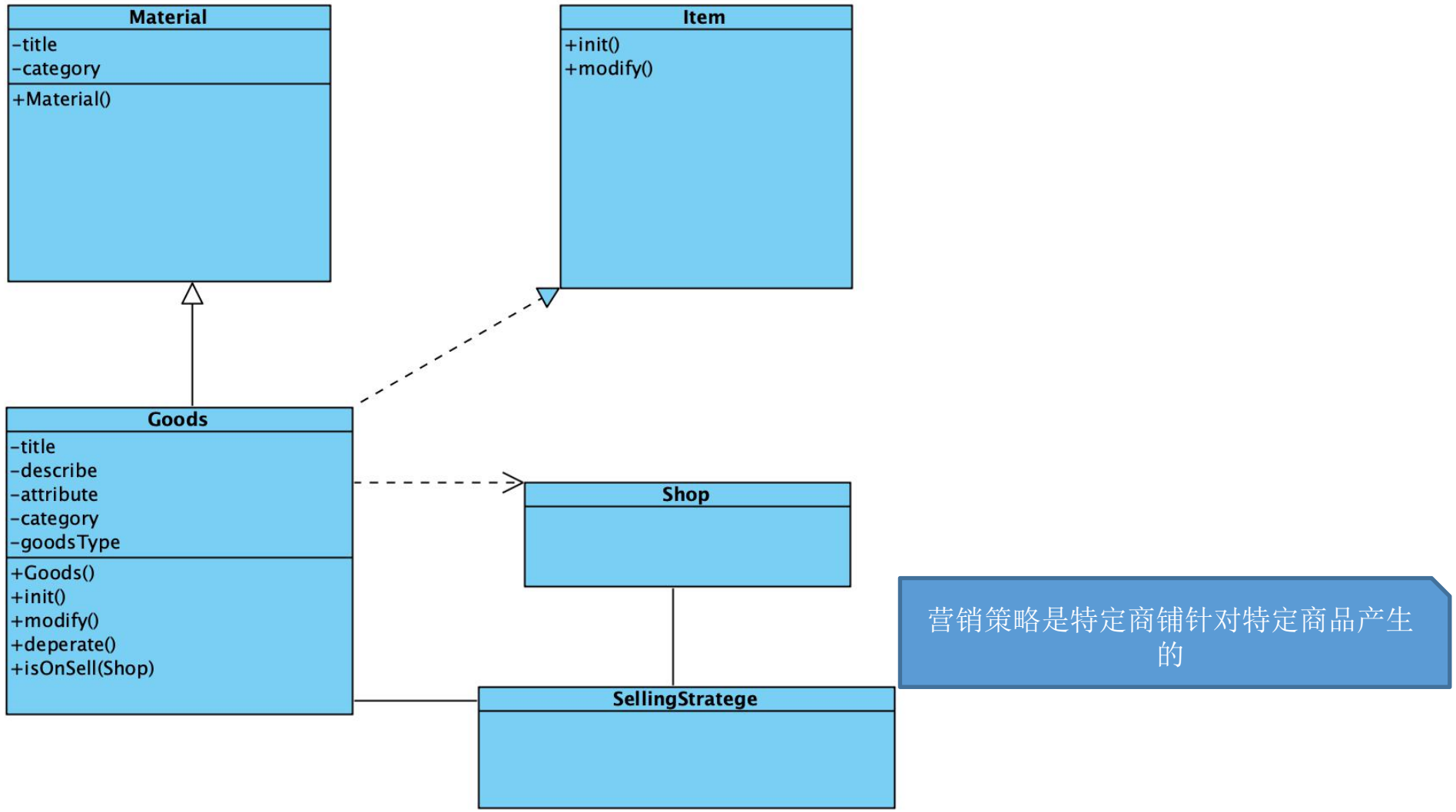
```java
/**
  * @author senyang
  * @version $Id: Shop. java, V 0.1 2020年03月07日 11:44 PM senyang Exp $
  */
public class Shop{
    private String name;
    private String privince;
    private String city;
    private String Street;
    private ShopDao shopDao;
    public Shop modify(Shop newShop){
        return shopDao.update(newShop);
    }
    public void delete(String shopId){
        shopDao.delete(shopId);
    }
    public Goods upload(GoodsRequest request){
        return covertRequestToGoods(request);
    }
    private Goods covertRequestToGoods(GoodsRequest request){
        return new Goods();
    }
}
```

通过Shop构建Goods

关联关系

- 对于两个相对独立的对象，当一个对象的实例与另一个对象的一些特定实例存在固定的对应关系时，这两个对象之间为关联关系。

类的UML表达



```java
/**
 * @author senyang
 * @version $Id: SellingStratege. java, v 0.1 2020年03月08日 1:24 PM senyang Exp $
 */
public class SellingStratege{
    private Shop shop;
    private Goods goods ;
}
```

至少要有这两个基本属性

聚合与组合之间有什么区别, 正确的识别聚合与组合之间的关系.

频度:中

难度:三星级

通过率:低

聚合

has-a: 天猫小店有一个店主--seller

组合

contain-a :天猫小店中的VIP客户含花呗月卡客户

Why

聚合: 去掉了就不完整了---- 不能去掉

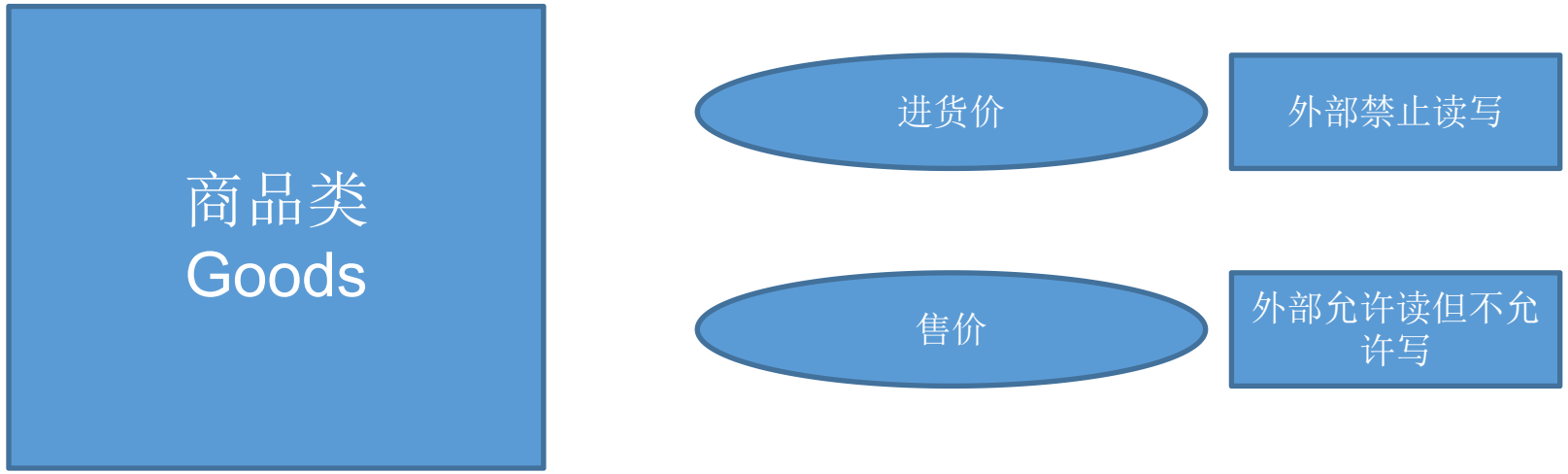
组合: 可以去掉,如花呗月卡客户加入天猫小店VIP客群,但没有花呗月卡这个产品时,天猫小店VIP客群仍然存在

封装、继承和多态

封装

保护隐私、隔离复杂度

封装——保护隐私



封装——隔离复杂度

- 以修改商品属性 - modify()为例

➢ 捞取并锁定商品

➢ 检查修改项合法性

➢ 修改

➢ 提交修改

外部只需直接调用该方法,不需要关心内部复杂的情况

继承

- 抽象和继承是前后衔接的关系，先有抽象，通过抽象得出类，后通过继承来表达抽象结果

继承常被考察的热点

java 子类继承父类各方法的执行顺序

用一个实验来说明问题:


```
import example.Students;
public class Main {
    static {
      System.out.println("Main 类静态代码段");
    }
    public static void main(String[] args) {
      System.out.println("第一次创建子对象");
      Students stu = new Students();
      System.out.println("第二次创建子对象");
      Students stu2 = new Students();
    }
}
```

用一个实验来说明问题:


```
public class Students extends Person implements Item {
    private int age;
    static {
       System.out.println("子类静态代码段");
    }
    public Students() {
        age = -3;
        System.out.println("子类构造方法");
    }
    public void printName(){
        System.out.println("直接输出属性: "+age);
    }
}
```

用一个实验来说明问题:


```
public class Person implements Item {
  private String name ;
  private int age;
  static {
    System.out.println("父类静态代码段");
  }
  public Person(){
    System.out.println("父类构造方法");
    age = 100;
  }
  // ......
}
```

运行结果


```
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java ...
Main 类静态代码段
第一次创建子对象
父类静态代码段
子类静态代码段
父类构造方法
子类构造方法
第二次创建子对象
父类构造方法
子类构造方法
```

总结： Java继承关系下的代码执行顺序

- Main 类静态代码段

----------第一次创建子对象----------

- 父类静态代码段
- 子类静态代码段
- 父类构造方法
- 子类构造方法

----------第二次创建子对象----------

- 父类构造方法
- 子类构造方法

深度解析

- java的机制是先编译成字节码文件.class 然后在JVM上解释器逐行翻译成机器码文件，那从.class 到JVM加载的时候，就执行static代码块和static变量，所以先执行静态代码块，并且执行顺序是先执行父类的在执行子类的。

深度解析

- Static代码块加载后放入栈帧,因此在同一个运行上下文中无需再次执行,这也意味着静态代码块必须重启容器才能被重新加载

多态问题的考点

- 多态知识的考察通常针对java的方法调用机制进行

一道大题:

面试题:Java语言中,如果子类实现了一个父类,调用一个类中方法的过程是什么样子的?

频度:高

难度:四星级

通过率:低

答案要点：编译器识别

编译器查看对象的声明类型和方法名

编译器查看对象的声明类型和方法名

编译器去寻找所有名字为相同但参数类型不同的方法。例如可能存在方法f(int)和方法f(String)，编译器会一 一列出所有当前类中所有名为f的方法，和其父类中访问属性为public且名为f的方法（父类中的私有方法不可访问）

面试题:JAVA方法的调用过程描述

答案要点: 编译器检查参数类型

- 编译器将查看调用方法是提供方的参数类型

面试题:JAVA方法的调用过程描述

编译器将查看调用方法是提供方的参数类型

-
- 如果在所有名为f的方法中，有与所提供的参数完全匹配的方法，则选择这个方法，这个过程称为“重载匹配”。

面试题:JAVA方法的调用过程描述

例如

- 对于调用f("Hello")，编译器就会选择并调用f(String)方法，而不会调用f(int)。
- 另外由于允许类型转换（例如int可以转换为double）.
- 因此如果编译器没有找到与参数类型匹配的方法，或者找到类型转换后有多个方法与之匹配，就会报错。

（注：这里的类型转换以不丢失精度为标准）

面试题:JAVA方法的调用过程描述

静态绑定与动态绑定

- **如果是private、static、final方法或者构造器，编译器会准确的知道应该调用那个方法，这种调用方式成为静态绑定.**
- **与此对应的是，调用方法依赖于隐式参数的实际类型，并且在运行时实现动态绑定。**

面试题:JAVA方法的调用过程描述

高分答案要点:区分静态绑定与动态绑定

面试题:JAVA方法的调用过程描述

当程序运行时，并且采用动态绑定的调用方法时，虚拟机一定会调用与x所引用的对象的实际类型最合适的那个类的方法与此对应，调用方法依赖于隐式参数的实际类型，并且在运行时实现动态绑定。

面试题:JAVA方法的调用过程描述

例如，假设x的实际类型时D，它是C的子类。如果D类定义了方法f(String)，则直接调用它，否则去D的父类中去寻找方法。

举例说明


```
 Students stu = new Students();
stu.printName();
System.out.println("用getter输出："+stu.getAge());
```

这里调用了stu.getAge()

看Students类的定义:

```
public class Students extends Person implements Item{
    private int age ;
    static{
        System.out.println("子类静态代码段");
    }
    public Students(){
        age = -3;
        System.out.println("子类构造方法");
    }
    public void printName(){
        System.out.println("直接输出属性" + age);
    }
}
```

Students并没有去实现getAge()

- 这个时候就会进一步到父类中去查找

面向对象的三大核心特征


```
public class Person{
    private String name;
    private int age;

    public Person(){
        age = 100;
    }
    public String getName() {
      return name;
    }
    public int getAge() {
      return age;
    }
    public void setAge(int age) {
      this.age = age;
    }
}
```

虽然编写者期望的明显是输出Students中的Age属性

然而由于编码者的疏忽:

面向对象的三大核心特征


```
public class Students extends Person{
    private int age;

    public Students(){
    }

    public void printName()[
        System.out.println("直接输出属性: " + age);
    }
}
```

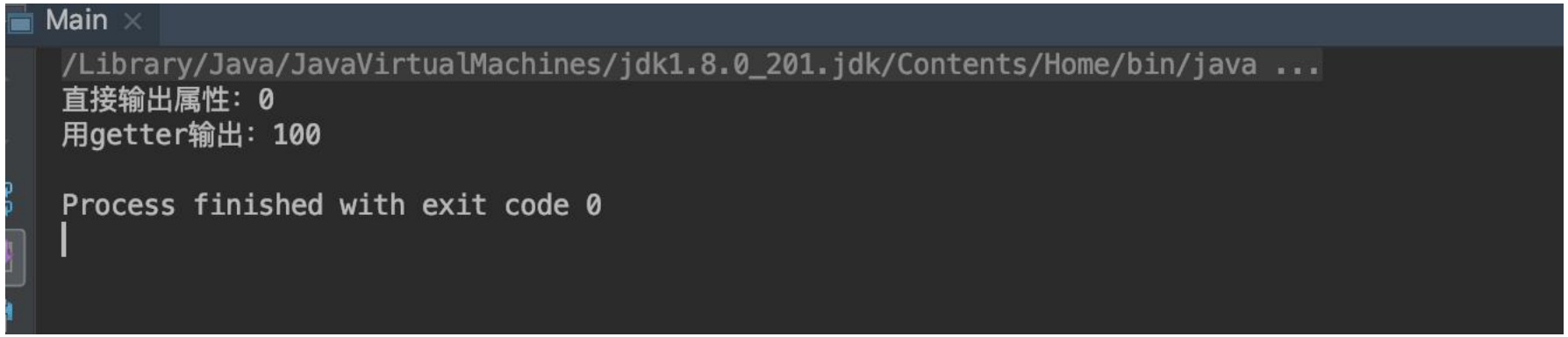
面向对象的三大核心特征

- 子类没有,就到父类去查找

```
public class Person{
    private String name;
    private int age;

    public Person(){
        age = 100;
    }
    public String getName() {
      return name;
    }
    public int getAge() {
      return age;
    }
    public void setAge(int age) {
      this.age = age;
    }
}
```

- 得到了非预期结果,因为就近原则,把父类中的属性值输出了



面向对象的考核要点

- 建模一个类并能正确的在UML中进行表达

面向对象的考核要点

- 正确识别类与类之间的关系

面向对象的考核要点

- 能够正确的设计类的封装粒度,准确对外部开放

面向对象的考核要点

- 熟知在继承的背景下,类的初始化过程

面向对象的考核要点

- 熟知多态叠加继承之后方法的调用规则

- 面向对象知识的考察在面试中属于第二层次的考察,技术深度考察的内容

- 面对这一类考察问题,首先做到知识要全,不能说漏,面试官引导后还答不出来,后面的问题基本你可以认为是面试官礼貌性的完成这次面试了.

- 深度考察两方面,一是实际问题的结合,二是jvm实际运行的规则