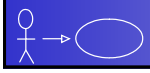
软件工程
第11讲 面向对象分析与设计（3）

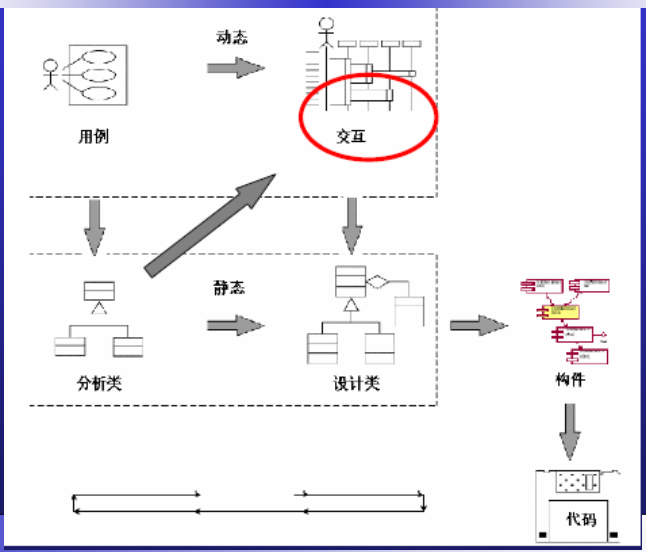
贾西平
Email: jiaxp@126.com

UML动态模型

- 顺序图
  - 最常用。以时间为中心，描述对象间的交互，焦点是消息的时间顺序。
- 协作图
  - 焦点：收发消息的对象结构组织。利用工具可以由顺序图生成。两者合称“交互图”。
- 状态图
  - 对一个类的生命循环建模，对复杂的动态行为有用.
- 活动图
  - 活动到活动之间的控制流



开发流程——动态建模



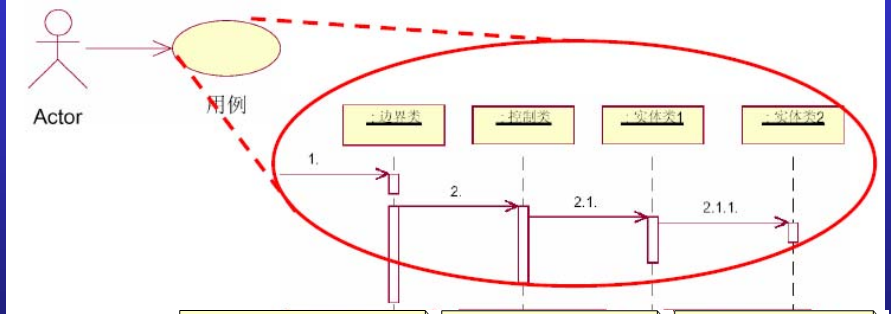
3

交互

你想要啊？你要是想要的话你就说嘛，你不说我怎么知道你想要呢，虽然你很有诚意地看着我，可是你还是要跟我说你想要的。你真的想要吗？那你就拿去吧！你不是真的想要吧？难道你真的想要吗？……

4

交互模式



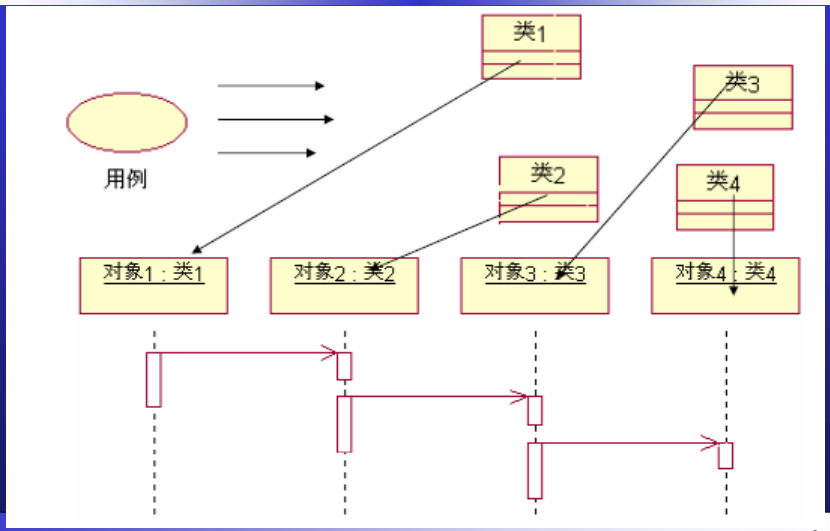
边界类：负责与外界交互，如窗口、通信协议、打印机接口、传感器、终端...

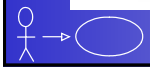
控制类：对应事件流（用例），把边界和实体对象分开。

实体类：对必须存储的信息和相关行为建模，稳定。

5

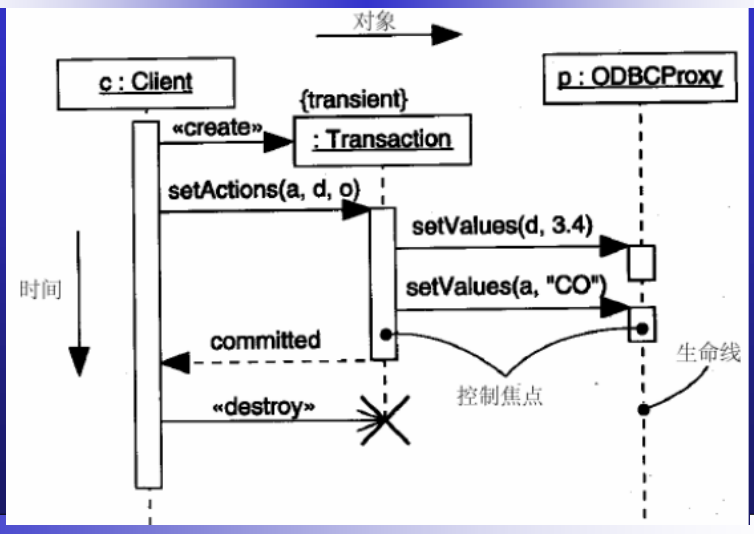
用例图-类图-顺序图





6

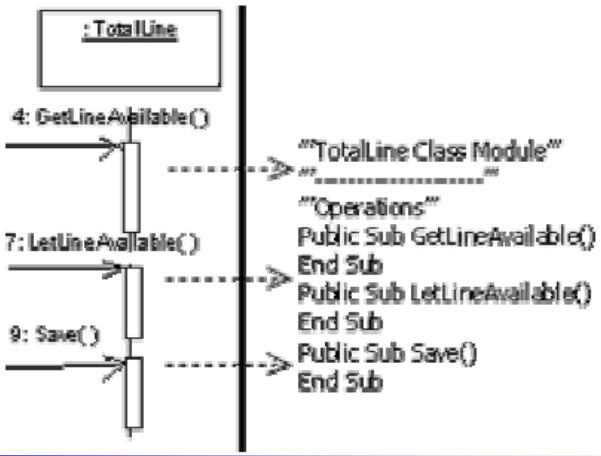
顺序图



7

顺序图和类图的映射

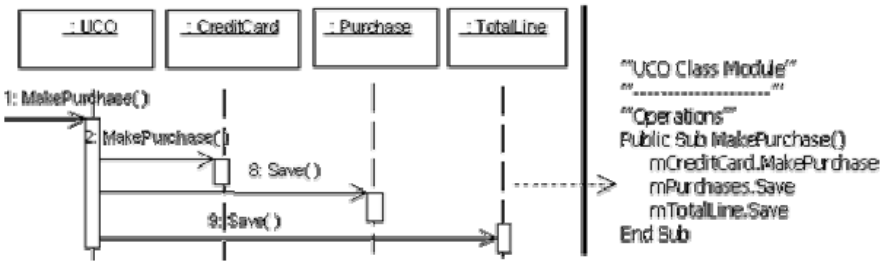
消息的传入：类对象所具有的操作——责任



8

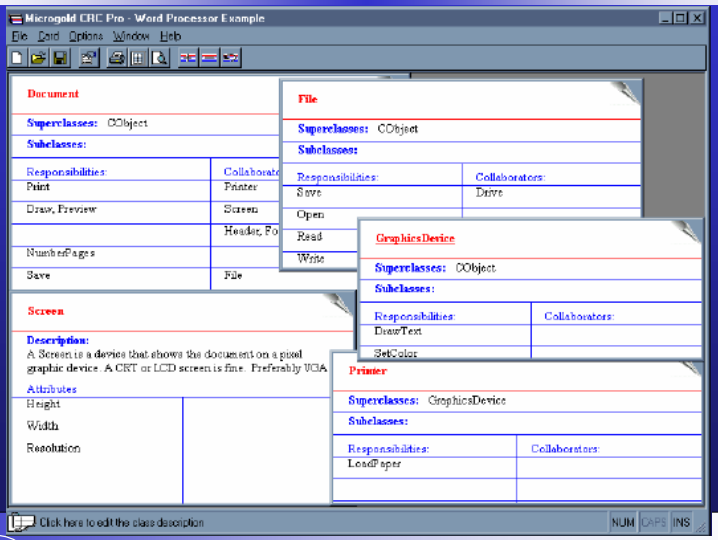
顺序图和类图的映射

消息的传出：类对象完成操作所需合作者——协作



9

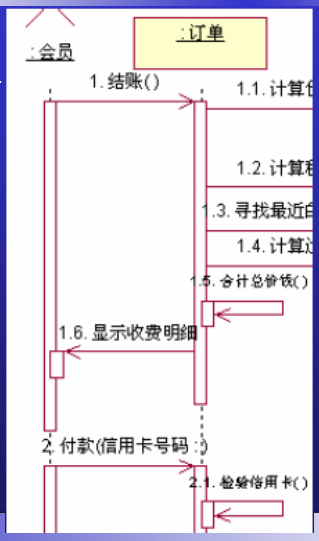
CRC（类-责任-协作）



10

顺序图和类图的映射

消息的传入



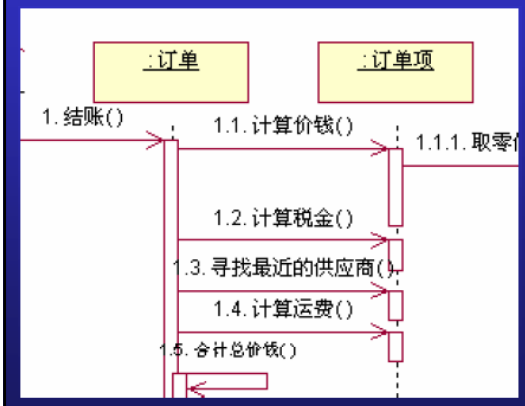
说明订单有以下操作

订单.结账()
订单.合计总价钱()
订单.付款()
订单.检验信用卡()

11

顺序图和类图的映射

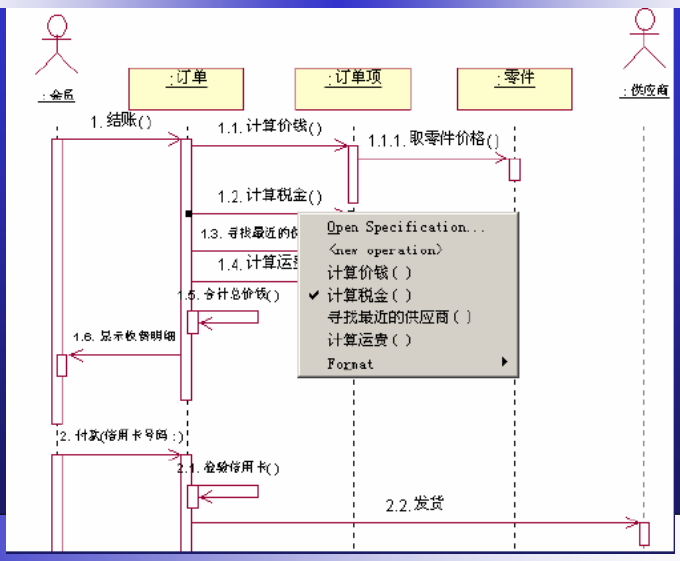
消息的传出




```
订单.结账()
{
    …
    订单项.计算价钱()
    订单项.计算税金()
    订单项.寻找最近的供应商()
    订单项.计算运费()
    …
    (订单.)合计总价钱()
}
```

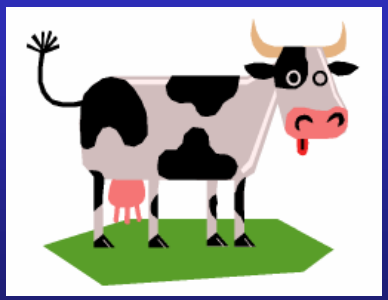
12

建模工具允许自动映射类图



责任分配？

- 每天凌晨4:30，农夫Jones都要起床，到牛栏去挤奶



怎样用OO方法表达这个挤奶过程？

面向过程的奶牛挤奶

- 定义函数Milk()

  --float Milk(struct cow, float amount);
- 挤奶（使奶离开奶牛）的过程：

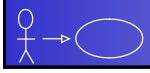
```
struct cow
{  char    name[30];
   float   currentMilkVolume;
   const    float maxMilkVolume = 3.0;
};

struct Cow  Bessie = {"Bessie", 2.5};
// . . . other code here . . .
// now. . .milk the cow
fReturned = Milk (Bessie, 1.3);
```

15

面向过程的问题

- “挤奶”操作是孤立的
  - 没有对挤奶农夫的需求
- Milk()必须告诉哪头牛来挤奶
- Milk()直接访问结构的内部
  - Milk()和结构形成耦合



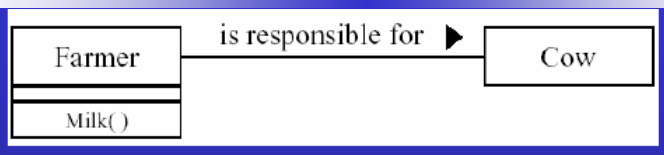
16

面向对象的挤奶

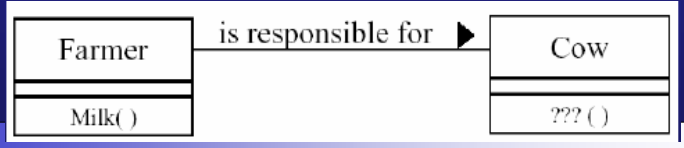
- 涉及的实体
  - 农夫，奶牛
- 涉及的关系
  - 农夫是奶牛的主人

17

农夫挤奶牛的奶？



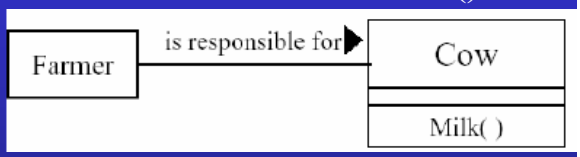
- 但这样行得通吗？
  - 牛奶在奶牛那里，不在农夫那里！
- Farmer::Milk不得不访问Cow的内部行为
- Farmer::Milk要调用Cow的哪个操作？



18

另一种责任分配方法

- 另一种方法－把Milk()放在Cow中



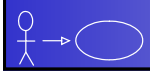

```
Struct cow
{
  char name[30];
  float currentMilkVolume;
  const  float maxMilkVolume=3.0;
};
```

- 为了挤奶，Farmer调用Cow::Milk()操作
- 所有的“结构”信息（数据）隐藏在Cow中
- 在OO模型中，Farmer不是直接从cow中取奶，而是请求Cow自己挤奶

19

考虑...

- 谁知道奶牛有多少奶？
  - 奶牛
- 谁知道在请求挤奶时是否有奶可挤？
  - 奶牛
- 如果农夫请求挤3加仑但奶牛只有2加仑可以提供。谁来决定应该提供2,1还是0加仑？
  - 奶牛



20

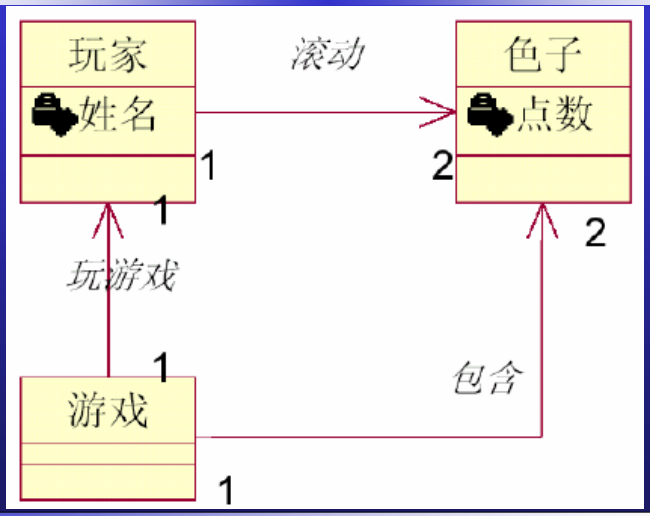
协作面向对象的掷色子

- 一个人掷两颗色子，点数为7则胜，否则负。针对此过程画出协作图。
- “游戏开始，玩家滚出两粒色子…”

怎样用OO方法表达这个掷色子过程？

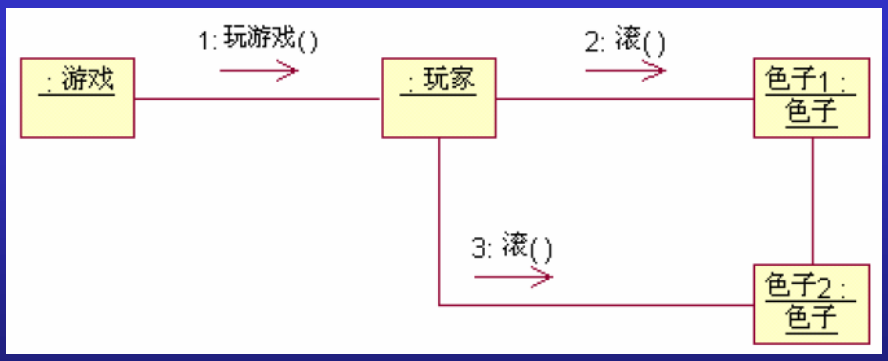
21

先看看有哪些类



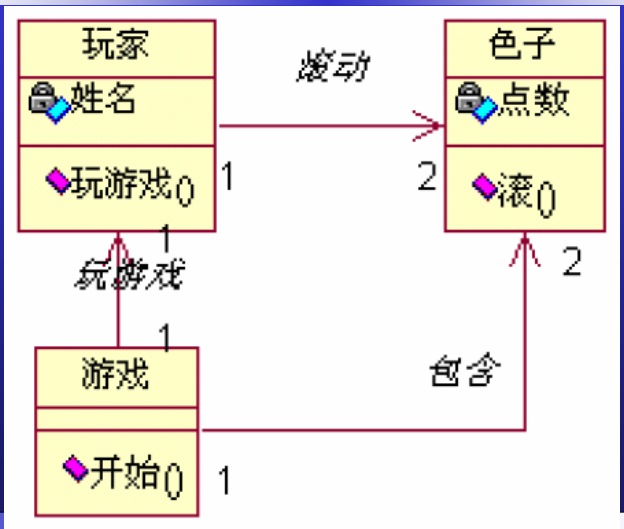
22

分析类的协作



23

加上了方法之后的类图



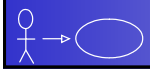
24

总的责任分配原则

低耦合，高内聚

耦合

- 描述设计的组成部分之间的相互依赖

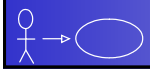


低耦合

- 类间要保持低耦合度
- 目的：复用

内聚

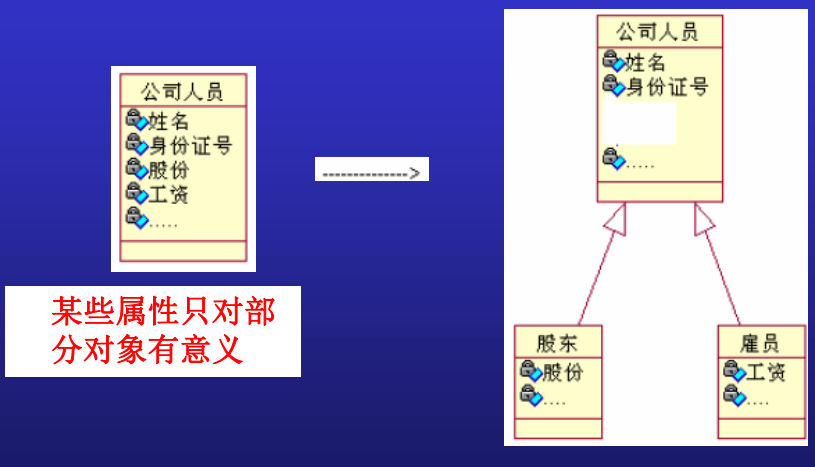
- 描述模块内各元素的紧密结合程度



高内聚

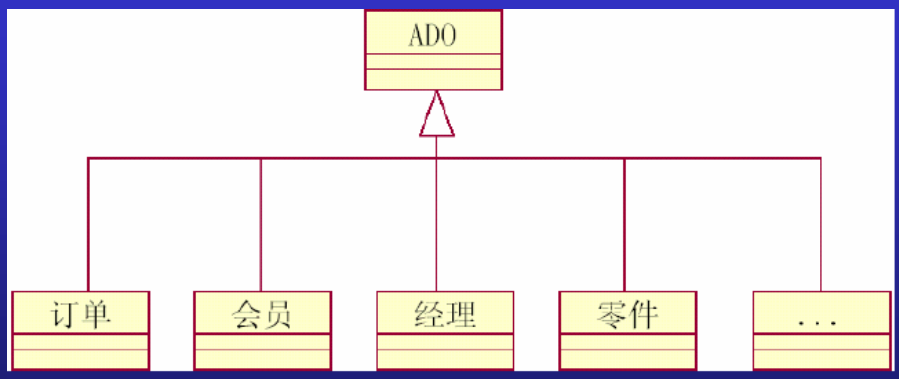
- 类内各元素要保持高内聚
- 小类，短方法——明确责任

低内聚例子（1）



某些属性只对部分对象有意义

低内聚例子（2）

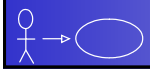


不同领域的类之间形成错误的关系

31

责任分配原则

- 原则1：专家（Expert）原则
- 原则2：老板（Boss）原则
- 原则3：可视（Visibility）原则



32

责任分配原则（1）

——专家（ Expert ）原则

- 把责任分配给信息专家
  - --有足够信息去完成该责任的类。
- 拟人化的抽象思维
  - 我知道什么？我能做什么？这件事该由谁负责？

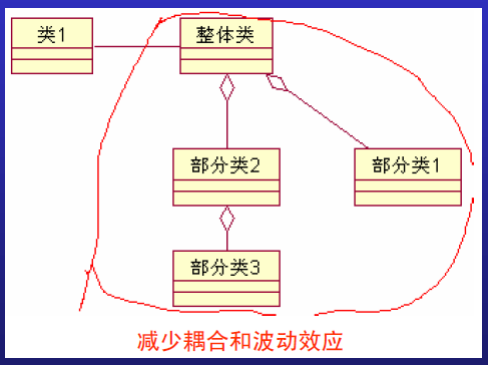
责任分配原则（2）

——老板（Boss）原则

- 聚合/组合结构的消息传递
- 当出现以下情况时，发给A的消息先通过B处理和中转：
  - B聚合A（Aggregation）
  - B组合A（ Composition ）

责任分配原则（2）

——老板（Boss）原则

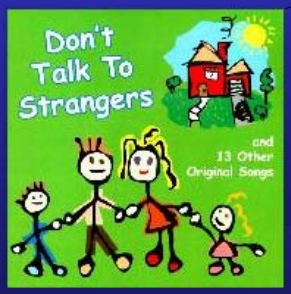


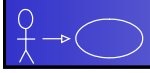
35

责任分配原则（3）

——可视（Visibility）原则

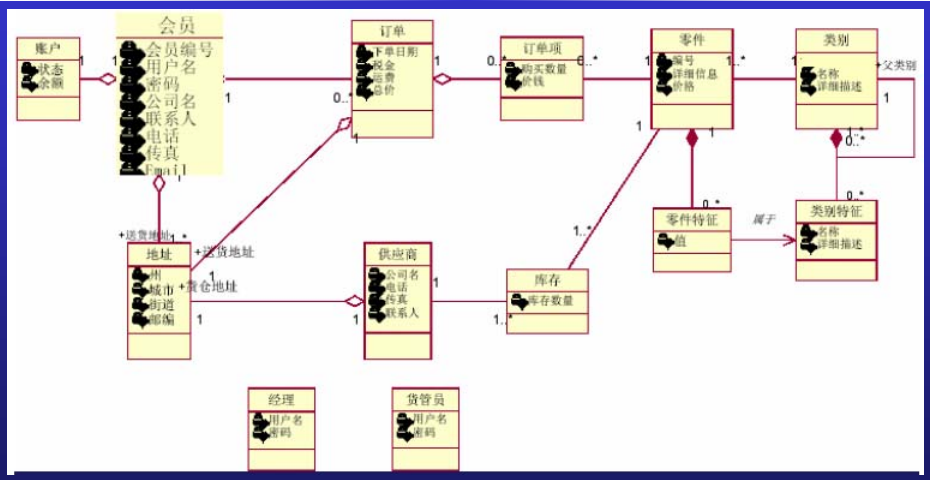
- 两个对象之间有消息传递，则相应类必有关联
- 不要与陌生人说话





36

类图（修订4）



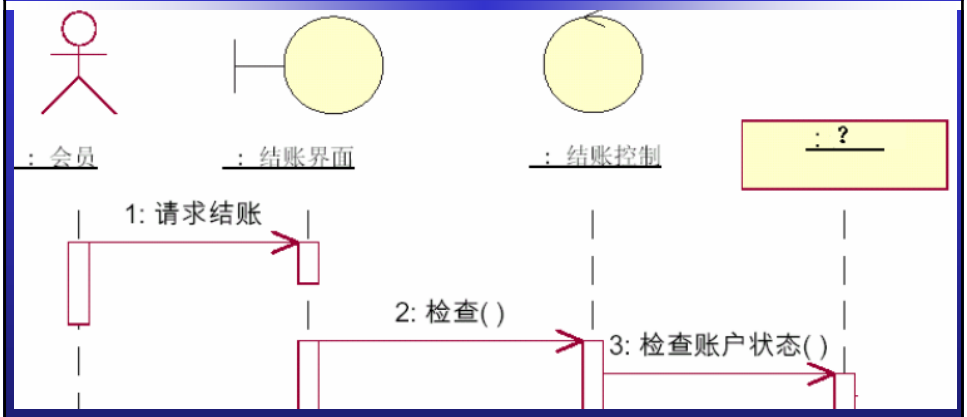
37

用例：结账：基本路径

1. 会员请求结账
2. 系统检查账户是否处于打开状态
3. 系统检查库存是否满足
4. 系统检查会员提交的信息是否充分
5. 系统合计订单总价（订单总价＝所有订单项价钱合计＋税金＋运费）
6. 系统显示收费明细
7. 会员确认
8. 系统通知供应商发货，减少相应库存数量

38

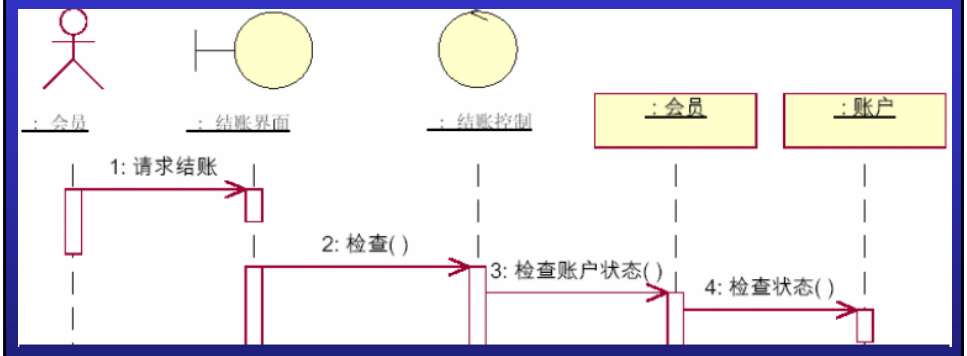
责任分配给谁？



会员？账户？

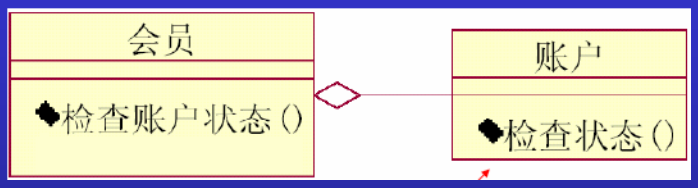
39

应用：专家+老板原则



40

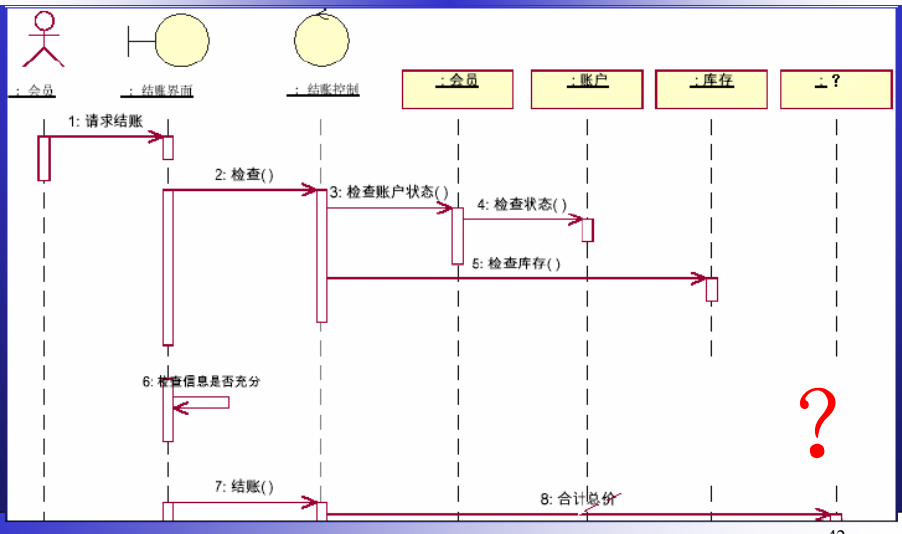
类和代码变化




```
会员.检查账户状态()
{
    …
    账户.检查状态()
    …
}
```

41

责任分配给谁？



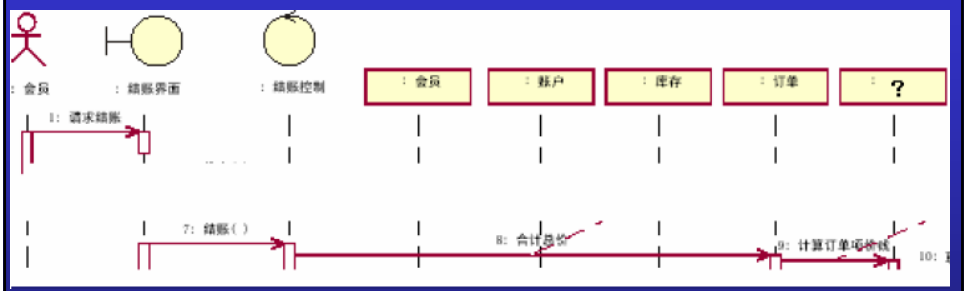
42

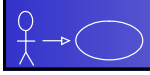
应用专家原则

- “合计总价”操作分配给哪个类？
  - 目的：统计所有订单项的总价钱。
  - 所需信息：每一个订单项的价钱及其总和。
- 考虑：谁知道这些？

  答案：订单

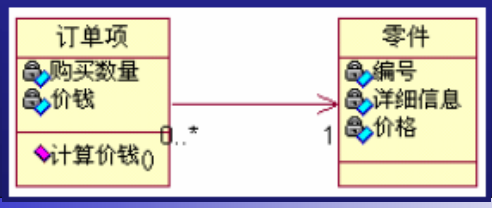
责任分配给谁？





应用专家原则

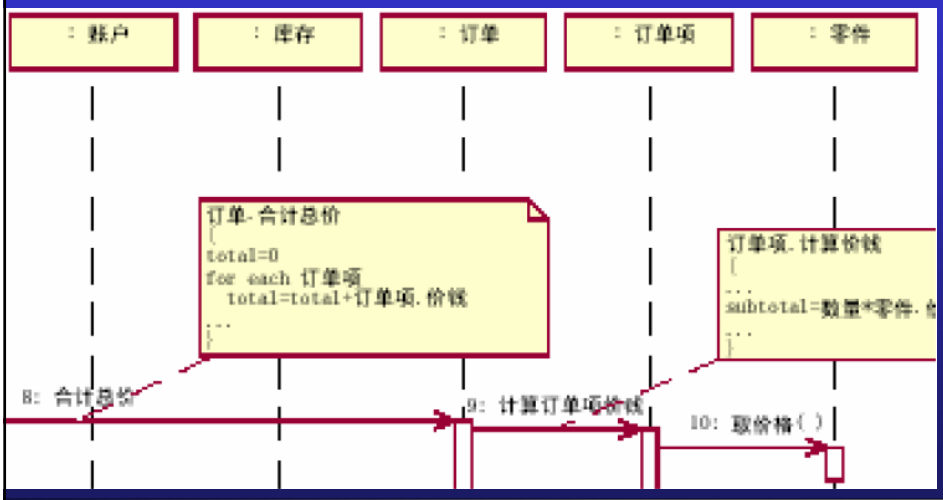
- “计算订单项价钱”操作分配给哪个类？（零件？订单项？订单？）
  - 统计某个订单项的价钱需要用到的信息：
  - 订单项.购买数量×零件.价格
- 考虑：谁知道以上的所有信息？零件？订单项？
- 答案：订单项

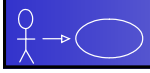


零件不知道订单项，订单项知道零件

45

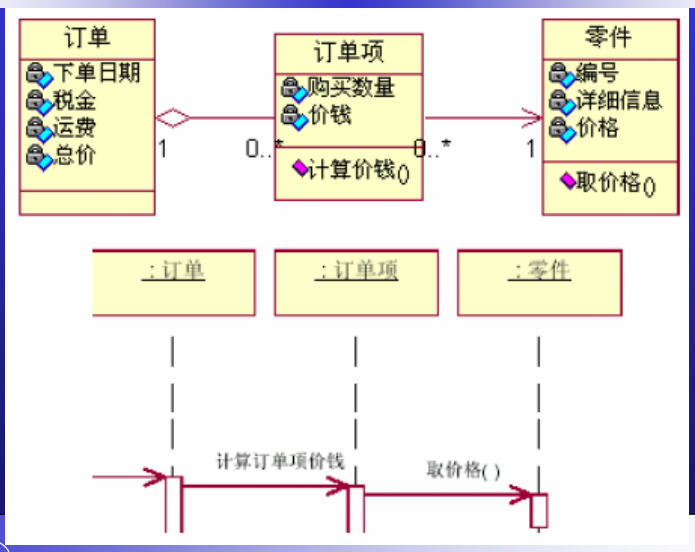
订单项找零件取价格





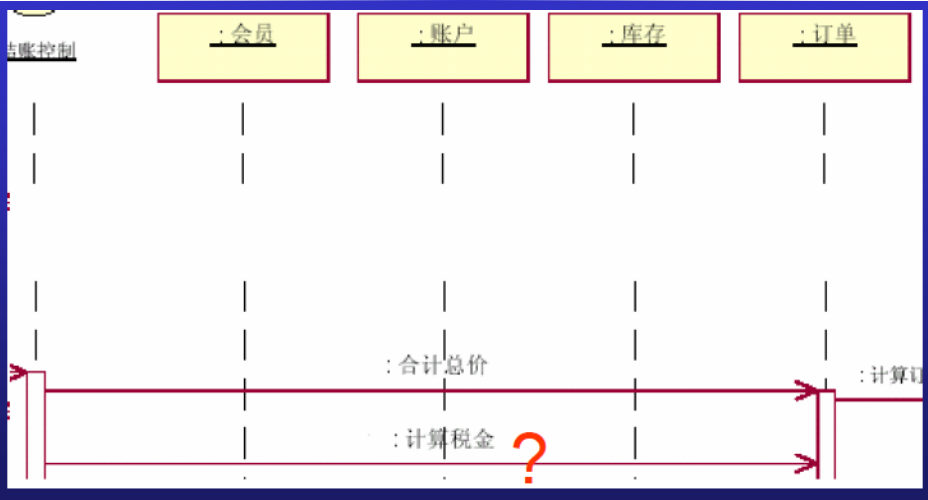
46

应用可视原则



47

谁来计算税金？

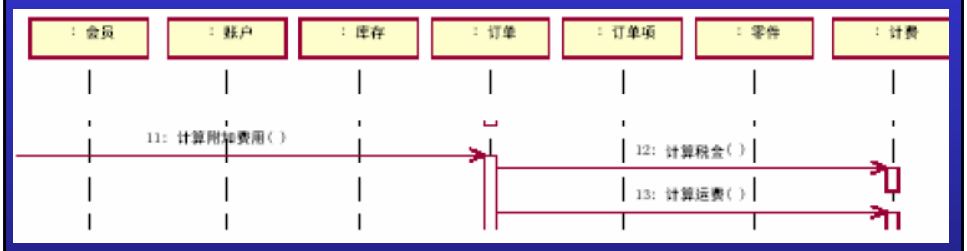


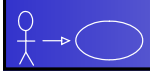
48

应用专家原则

- 计算税金
  - 需要用到的信息：税金＝价钱×相应税率
- 考虑：订单知道所有这些吗？
- 解决方法：添加一个“计费”类，负责计算各种附加费用。
- 计算运费也是如此

增加“计费”类





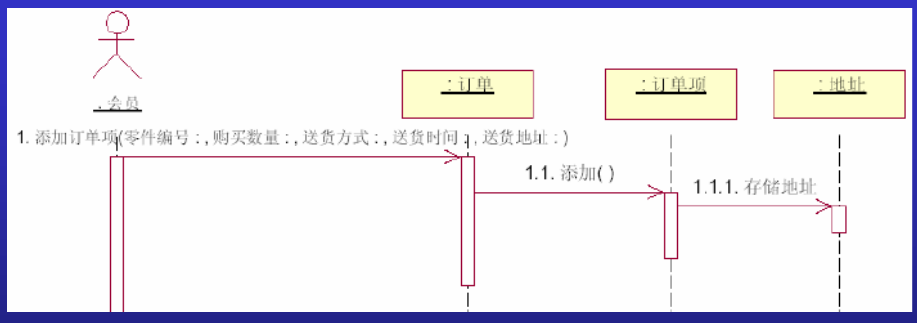
练习

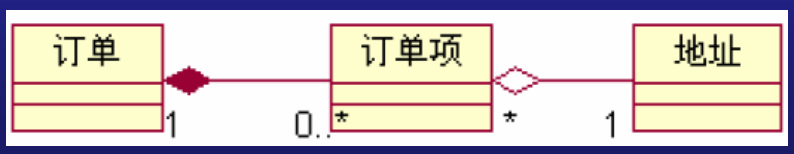
- “创建新的订单项”操作分配给哪个类？
  - 考虑：订单项是不是聚合/组合关系中的部分类
- 答案：

  先分配给订单，再由订单分配给订单项

51

练习：增加新的订单项





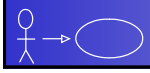
52

顺序图相关视频

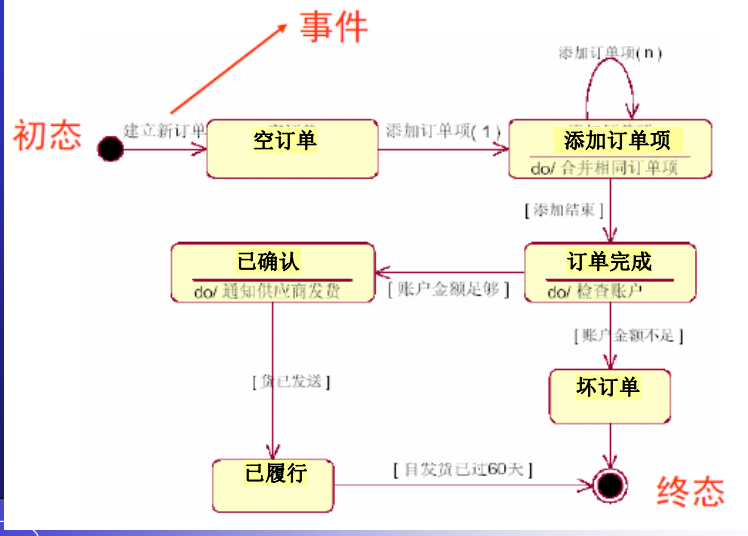
- 概念(9')
  https://www.bilibili.com/video/BV1Q741157ve?p=58
- 建模(16')
  https://www.bilibili.com/video/BV1Q741157ve?p=59
- 风格(13')
  https://www.bilibili.com/video/BV1Q741157ve?p=60

状态图

- 对类的生命周期建模
- 大多数面向商业的应用中的类不需要状态图
- 以下情况经常使用状态图
  - 交互图中产生或接收大量信息的类
  - 界面类
  - 实时系统中的类

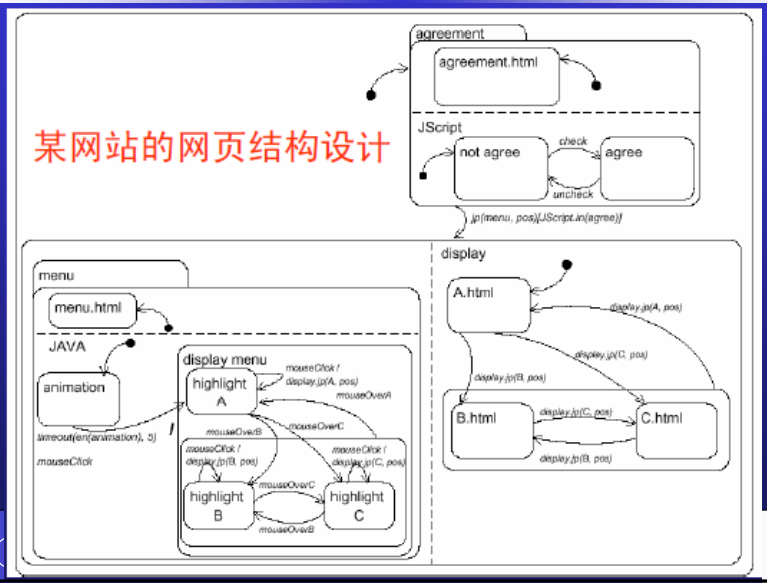


产生和接收大量信息：订单



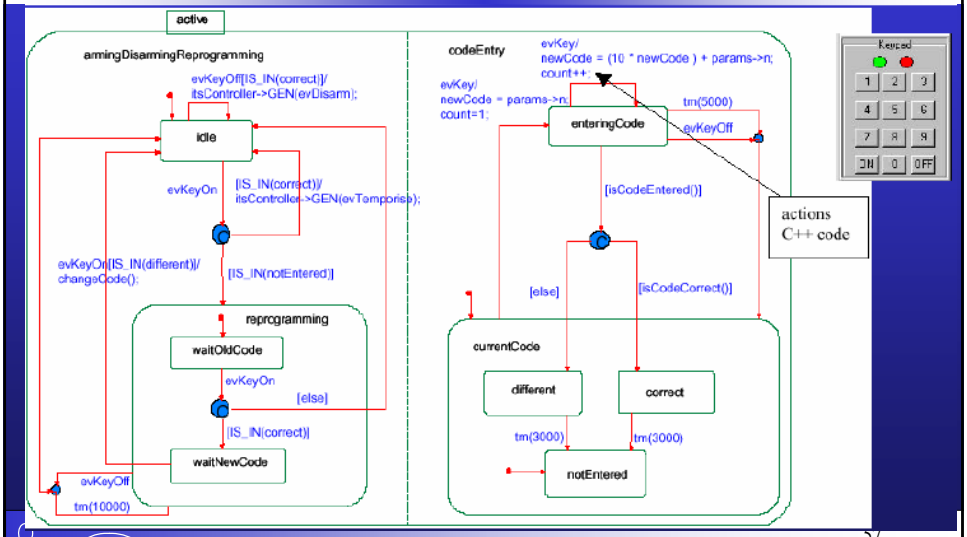
55

界面（包括HTML）



56

实时嵌入系统

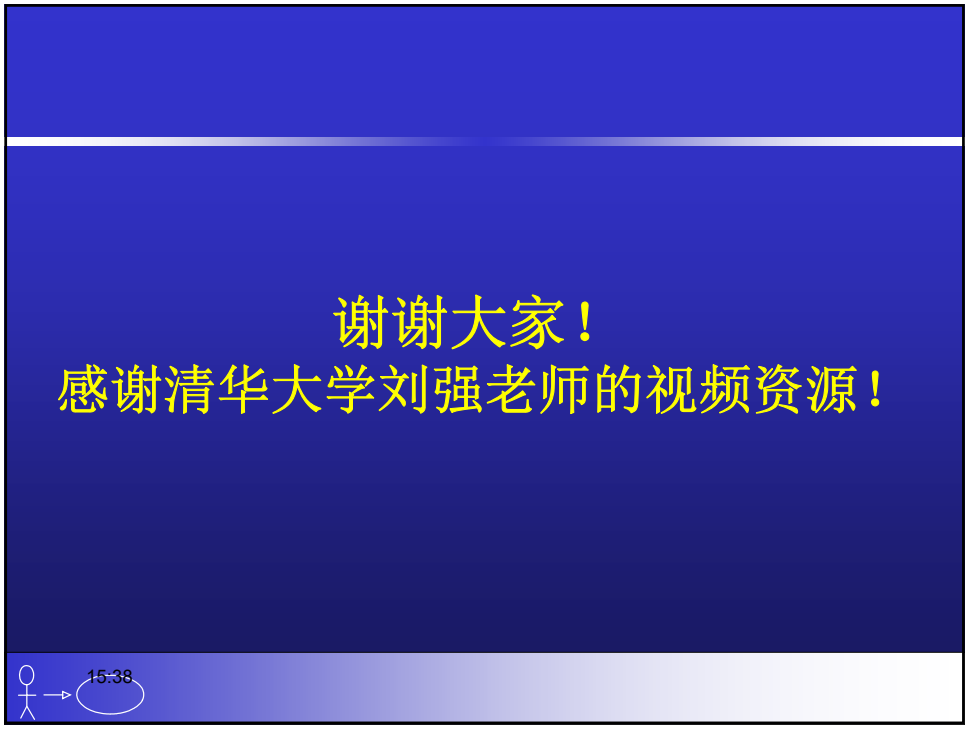


57

相关视频

- 状态建模（12’）
  https://www.bilibili.com/video/BV1Q741157ve?p=61
- 状态图（14’）
  https://www.bilibili.com/video/BV1Q741157ve?p=62
- 状态图精讲（13’）
  https://www.bilibili.com/video/BV1Q741157ve?p=63

58


谢谢大家！
感谢清华大学刘强老师的视频资源！
15:38