UML全程实作

序列图

Think 

核心工作流

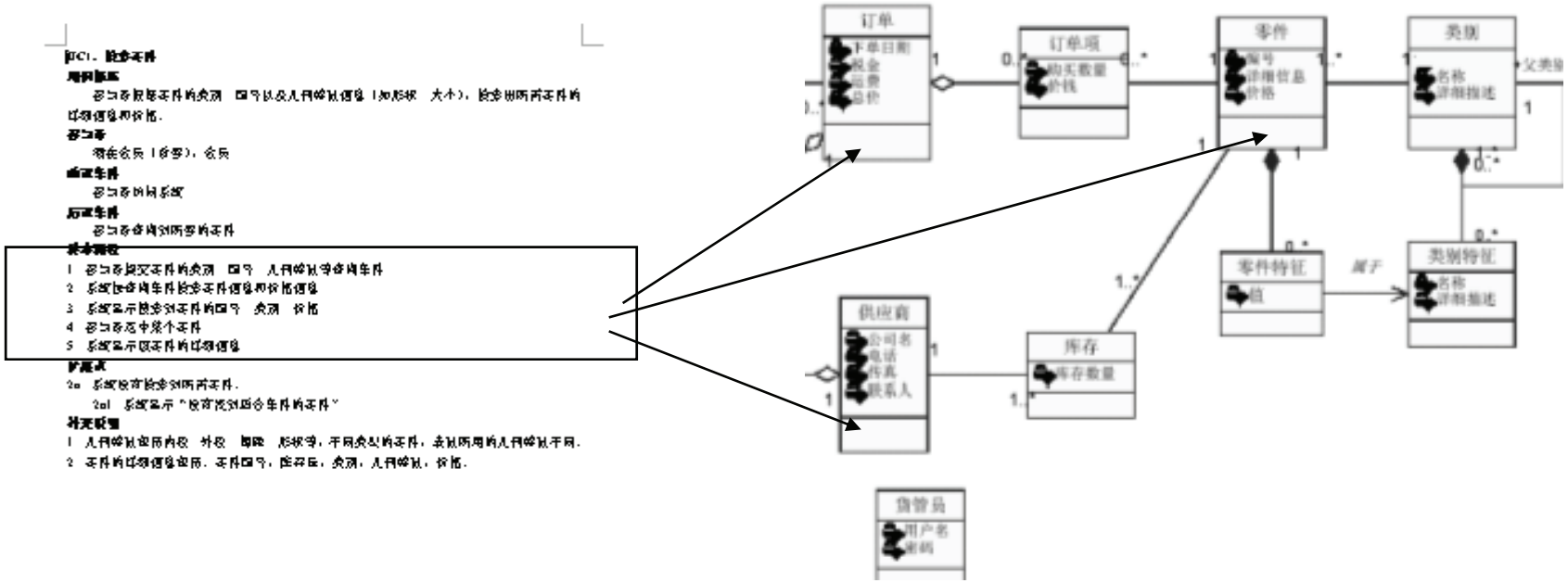
*愿景
*业务建模
    选定愿景要改进的业务组织
    业务用例图
    现状业务序列图
    改进业务序列图
*需求
    系统用例图
    书写用例文档

提升销售

*分析
    类图
    序列图
    状态图
*设计
    建立数据层
    精化业务层
    精化表示层

降低成本

现在已有的工件

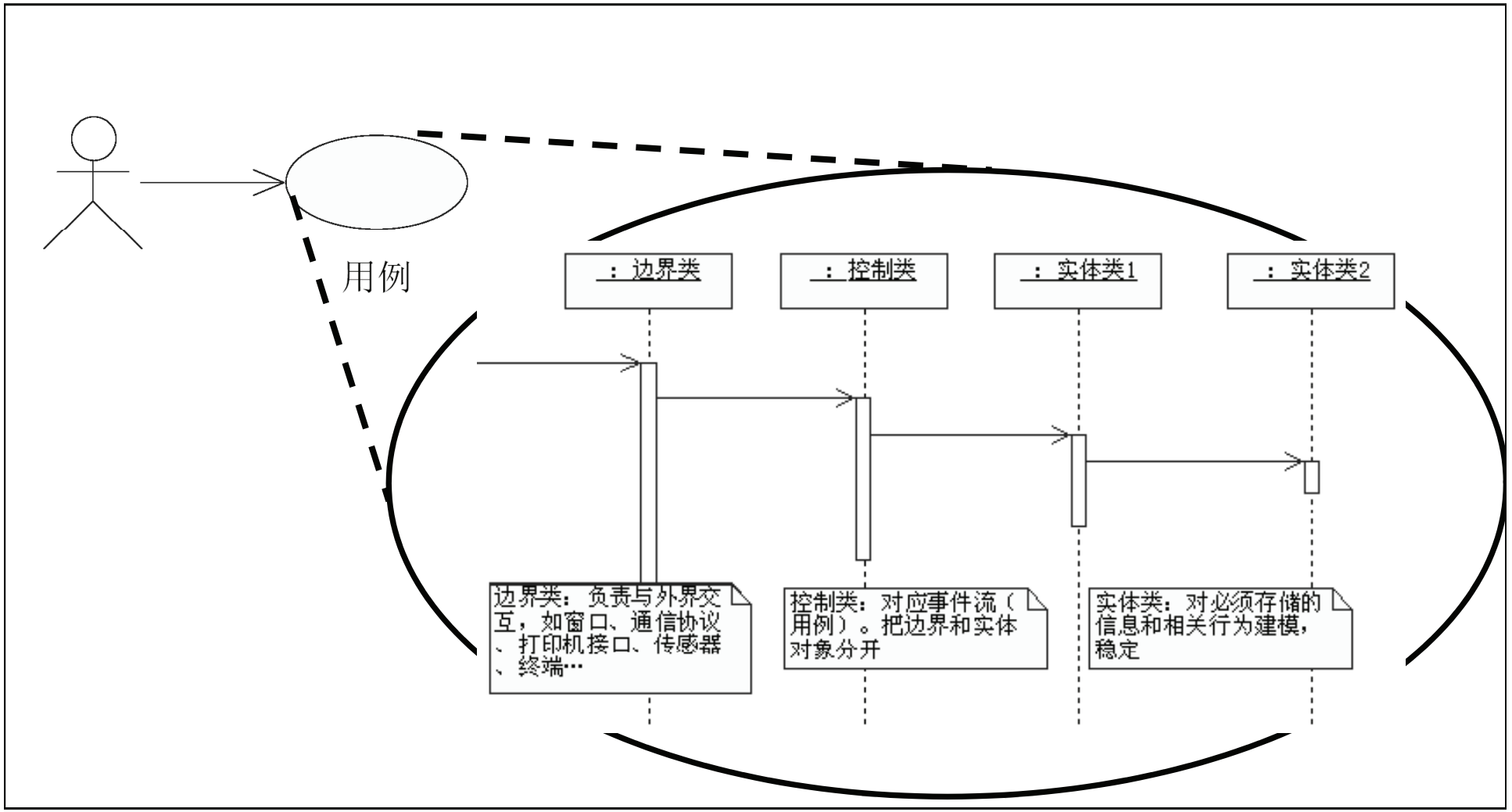


用例文档

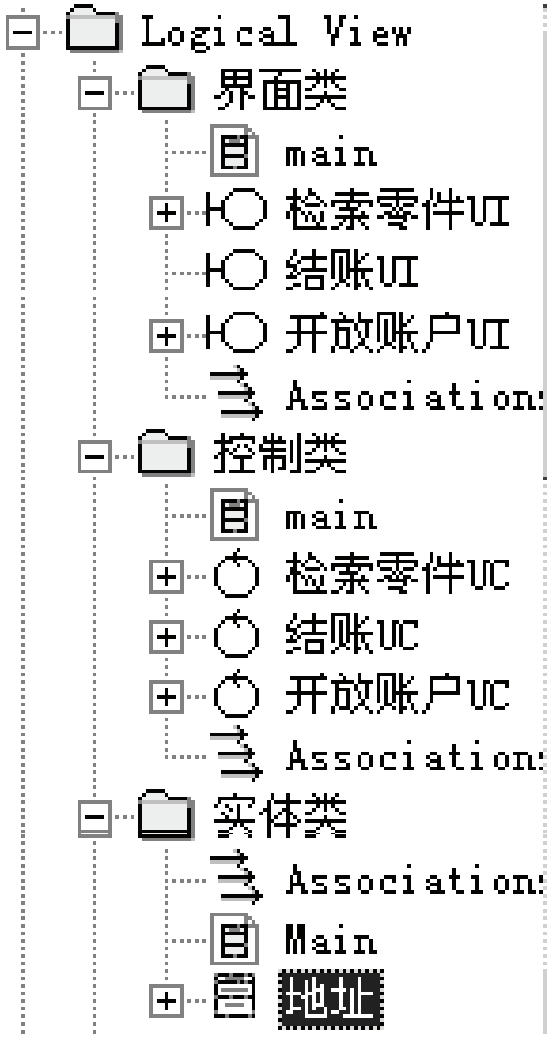
类图

通过画序列图完成责任分配

交互模式


用例
：边界类
：控制类
：实体类1
：实体类2
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

三种类



在分析工作流

❖边界类：用例的每个执行者映射一个边界类

❖责任：输入、输出、过滤

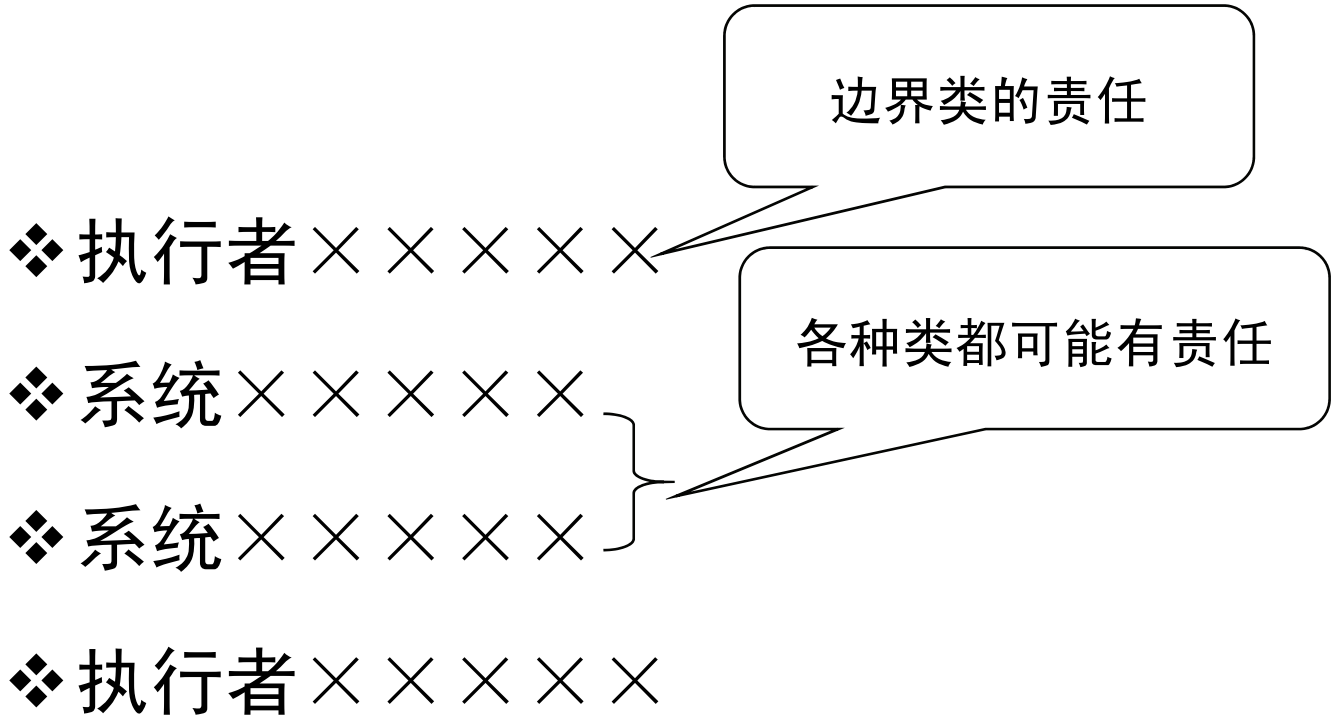
❖控制类（可选）：一个用例映射一个控制类

❖责任：控制事件流，负责为实体类分配责任

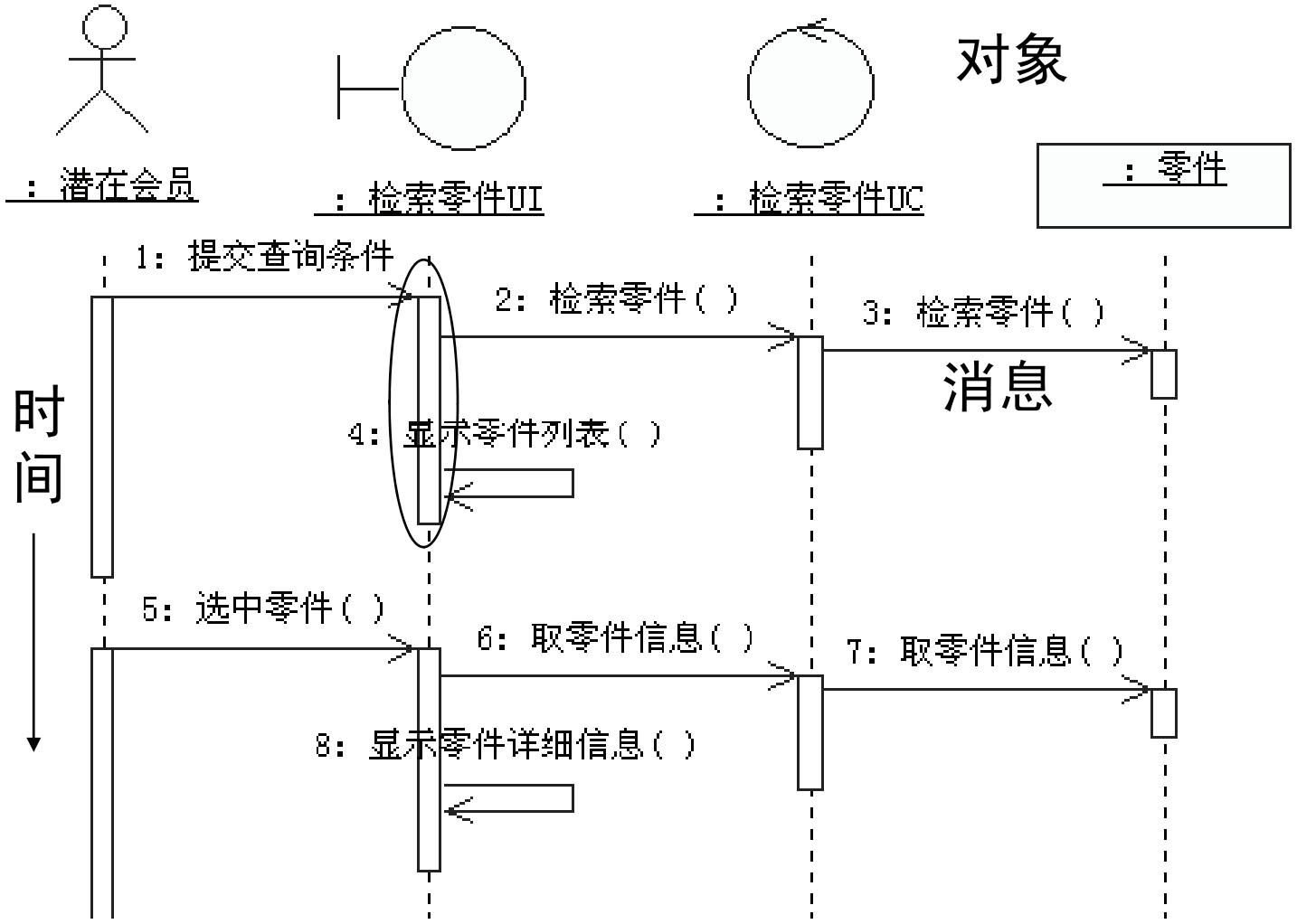
❖实体类：一个用例有多个实体类参与，一个实体类可以参与多个用例

❖责任：业务行为的主要承载体

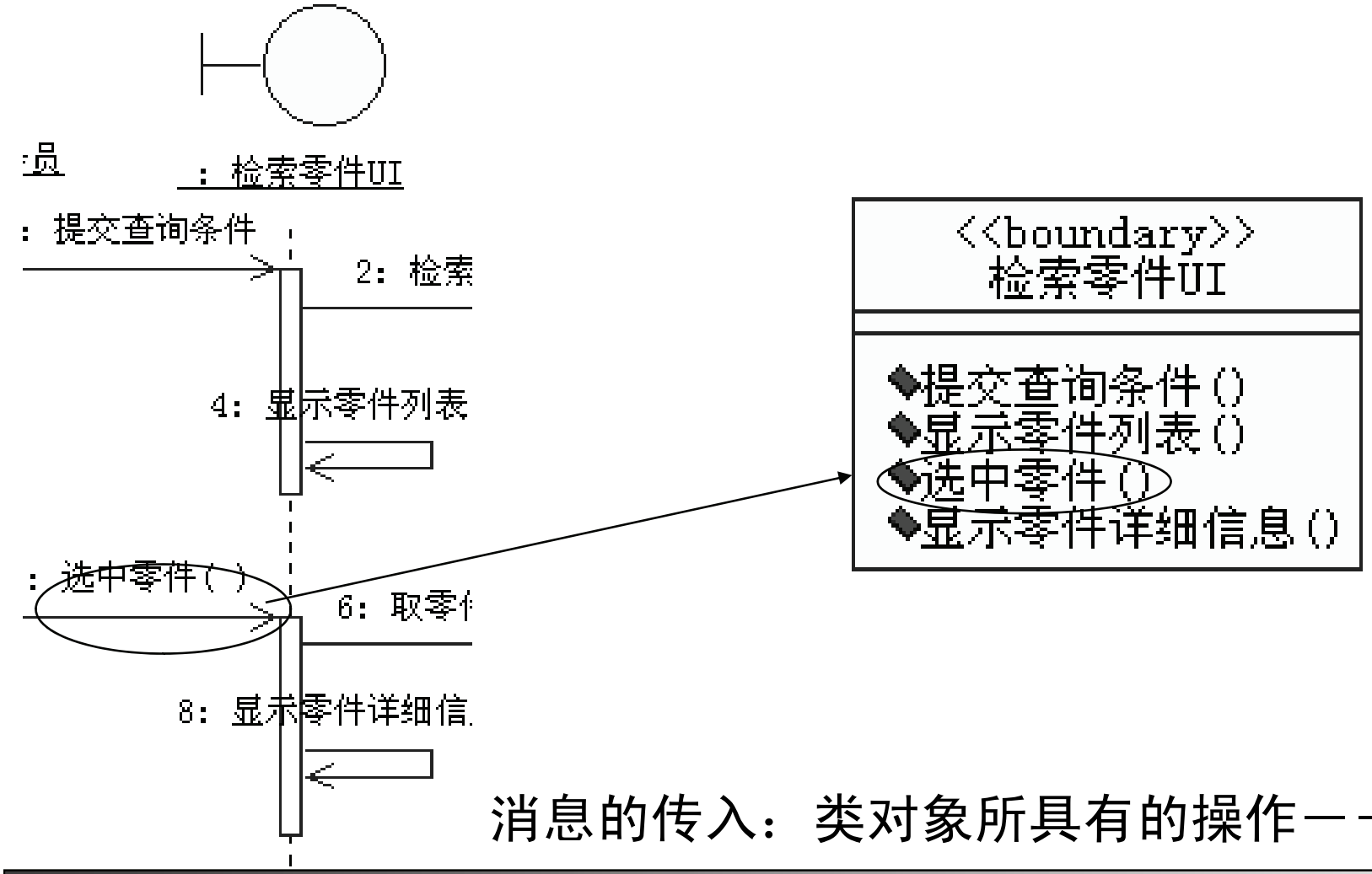
责任分配



序列图

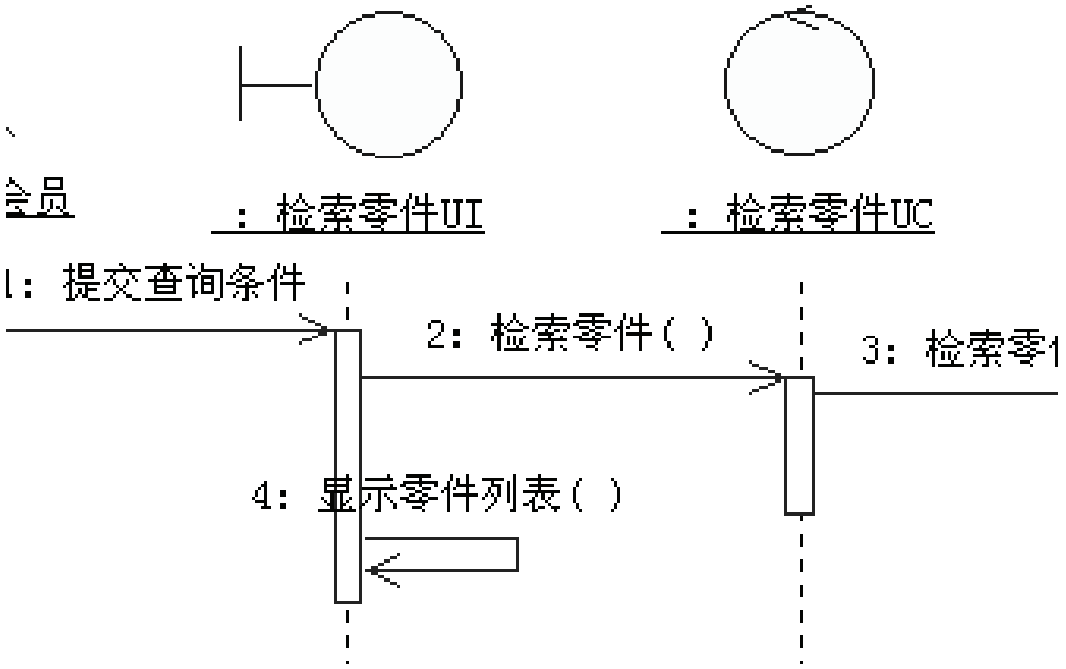

对象
：潜在会员
：检索零件UI
：检索零件UC
：零件
1：提交查询条件
2：检索零件（）
3：检索零件（）
消息
时间
4：显示零件列表（）
5：选中零件（）
6：取零件信息（）
7：取零件信息（）
8：显示零件详细信息（）

序列图和类图的映射



消息的传入：类对象所具有的操作——责任

序列图和类图的映射



检索零件UI.提交查询条件()

{

…

检索零件UC.检索零件()

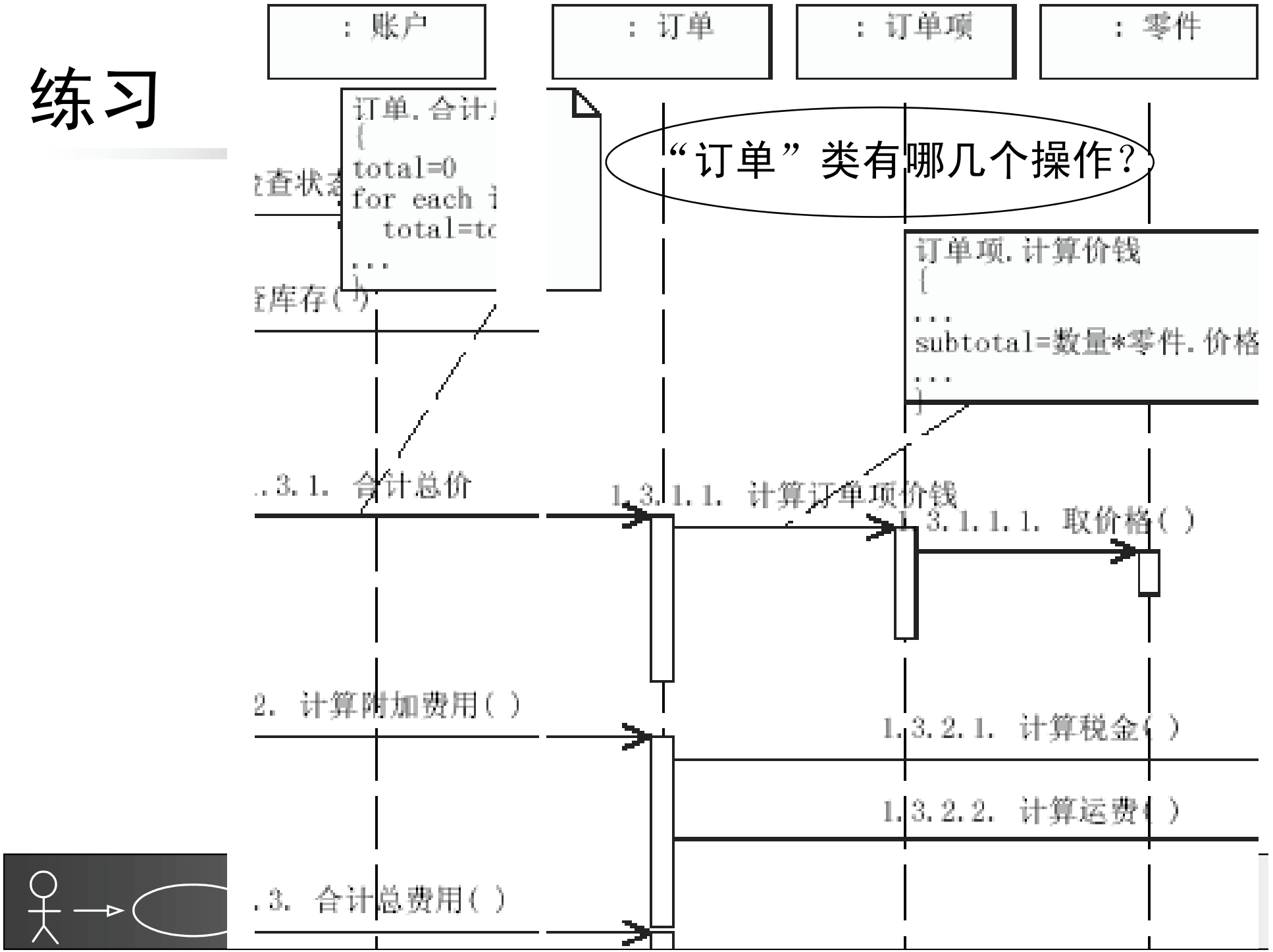
…

(检索零件UI.)显示零件列表()

}

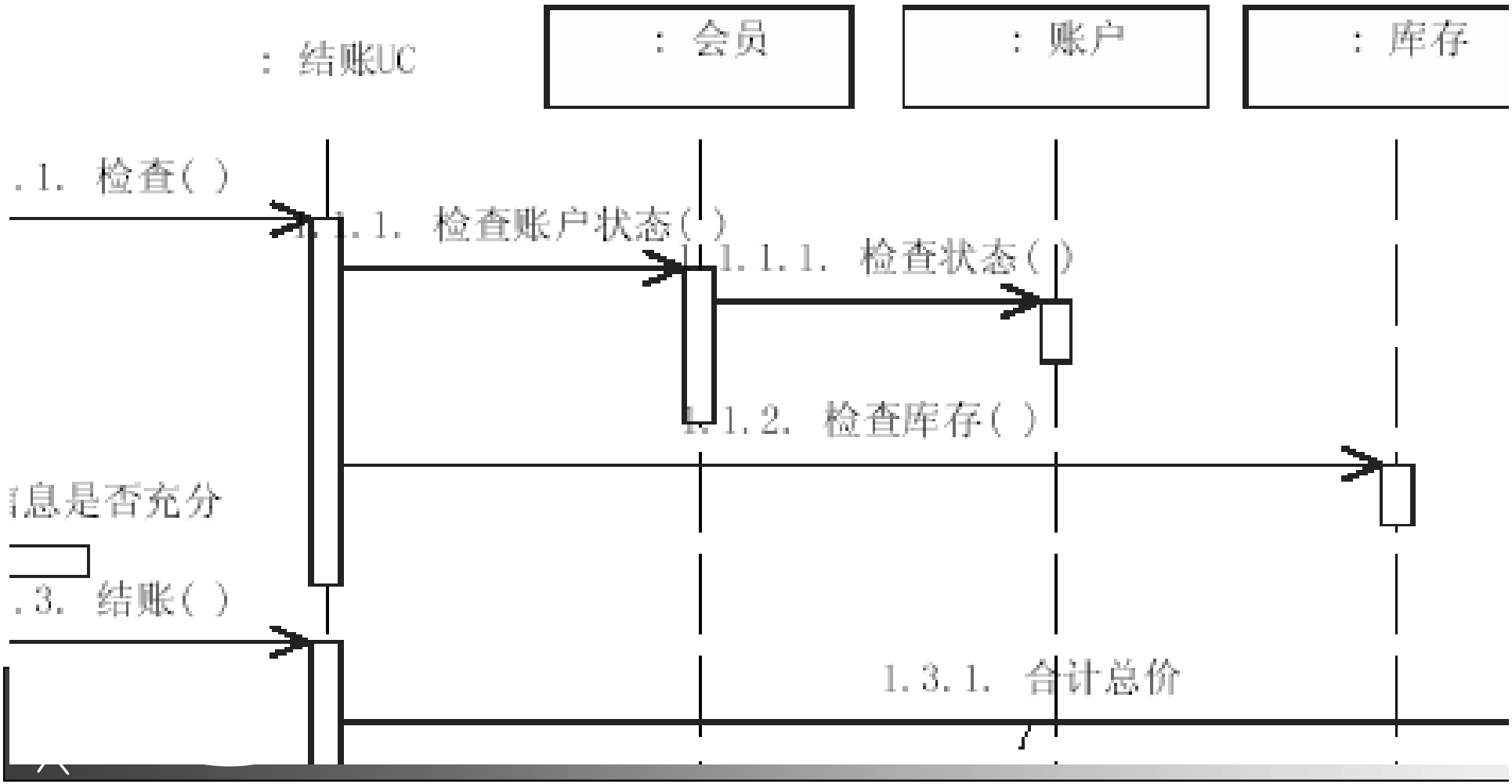
消息的传出：类对象完成操作所需合作——协作

练习

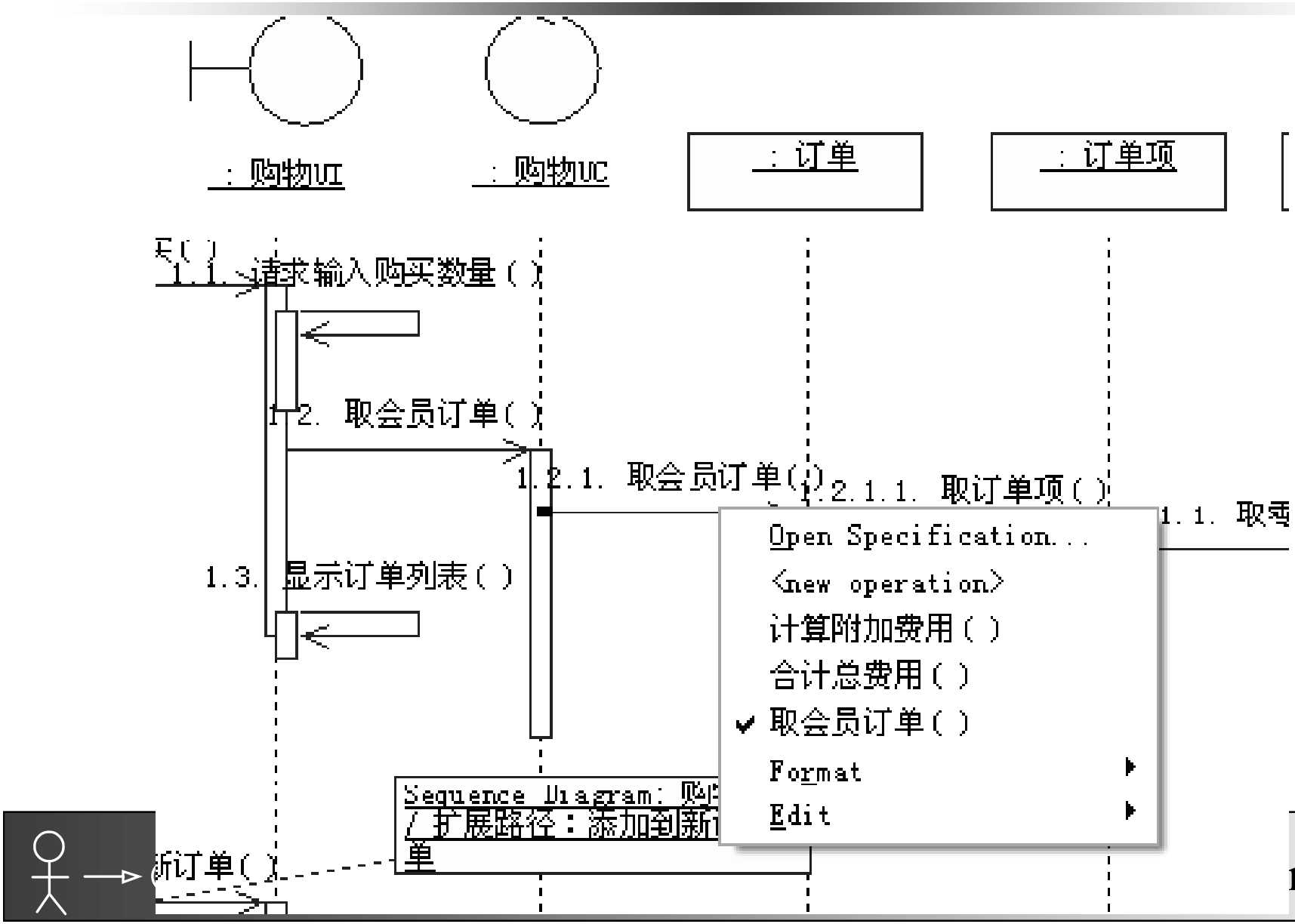

：账户
：订单
：订单项
：零件
订单.合计
{
total=0
for each i
total=tc
...
}
查状态
库存( )
“订单”类有哪几个操作？
订单项.计算价钱
{
...
subtotal=数量*零件.价格
...
}
..3.1. 合计总价
1.3.1.1. 计算订单项价钱
1.3.1.1.1. 取价格( )
2. 计算附加费用( )
1.3.2.1. 计算税金( )
1.3.2.2. 计算运费( )
.3. 合计总费用( )

练习

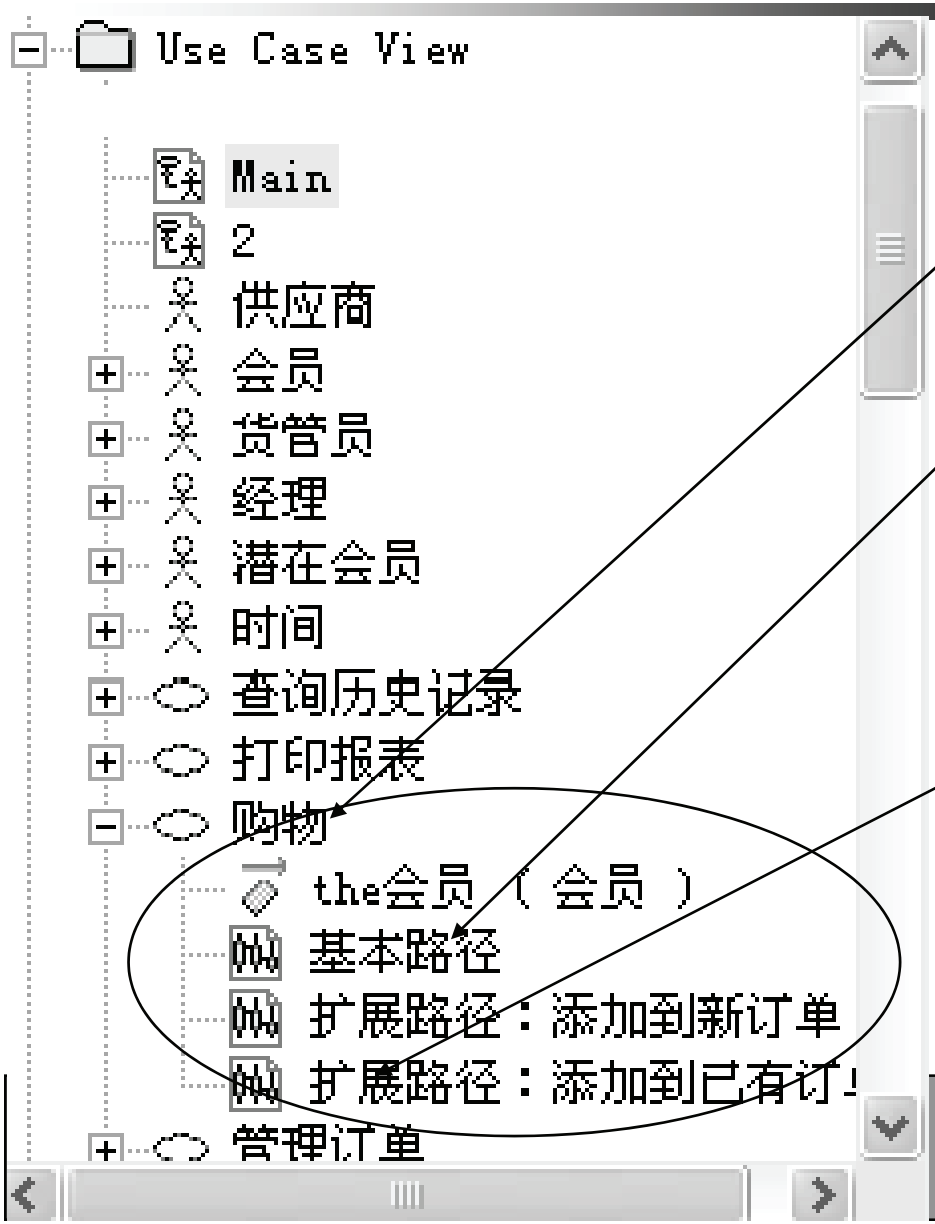
请写出：“结账UC”类的“检查”操作内部的代码



建模工具自动映射

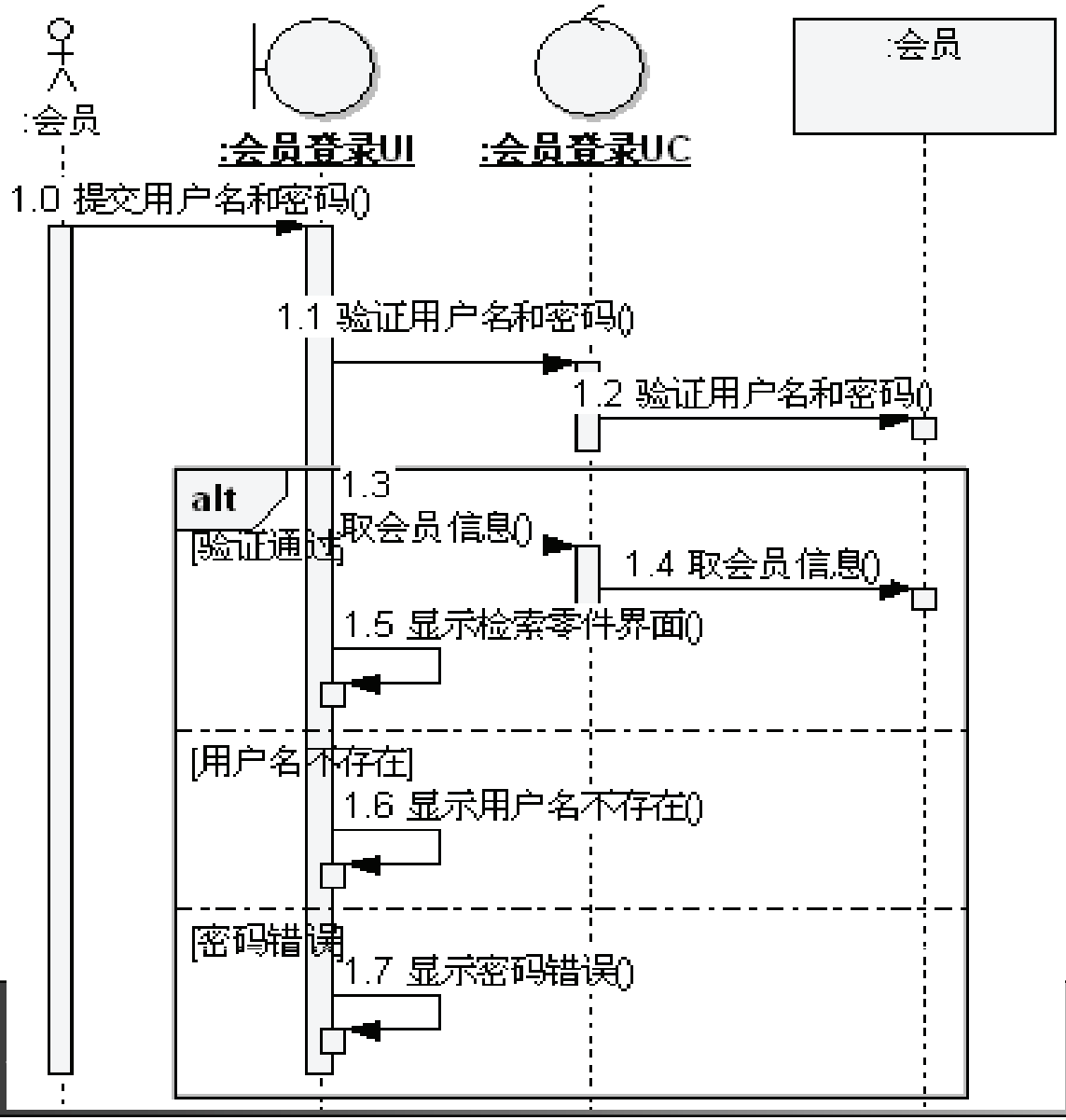


序列图绘制要点



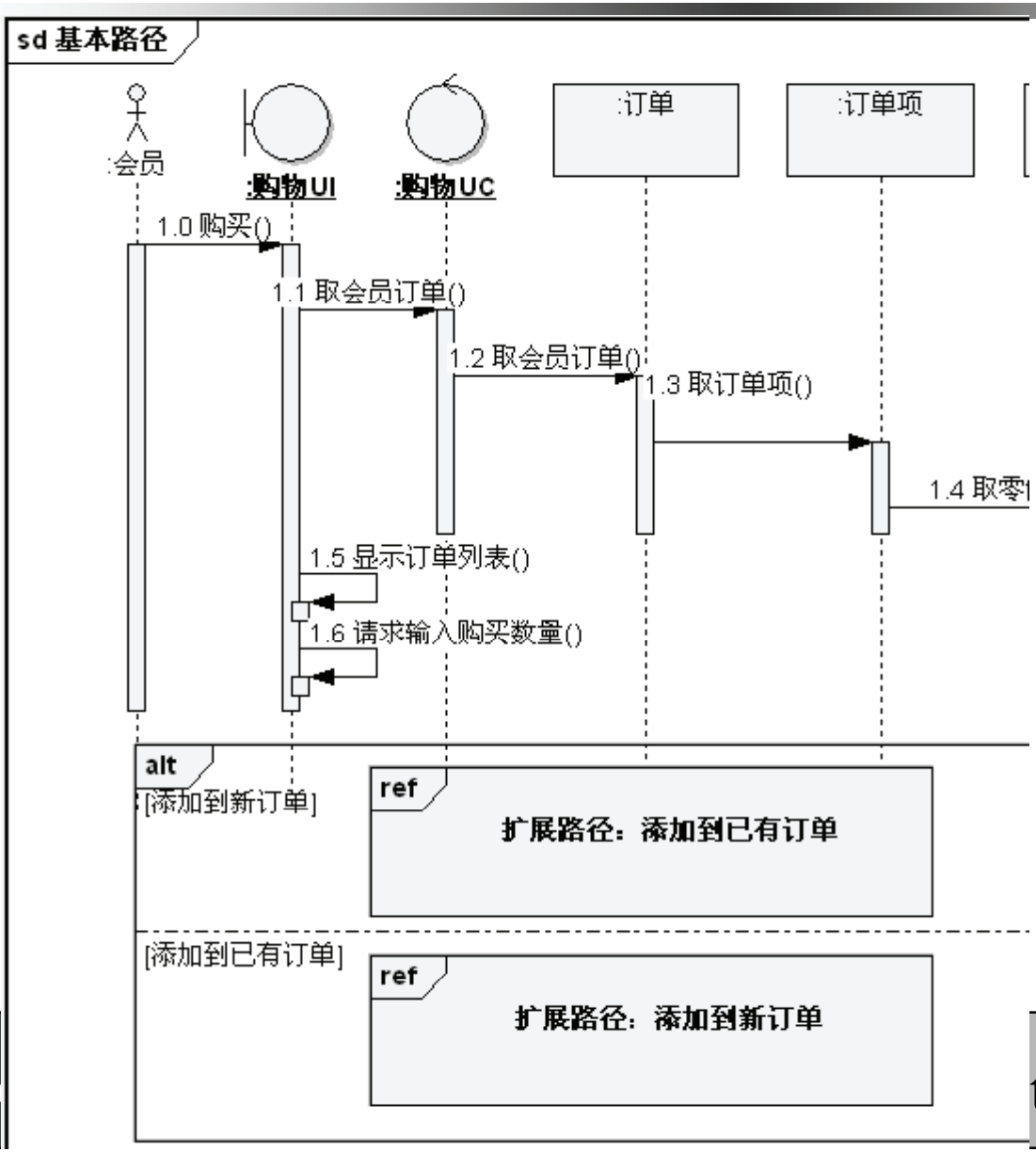
- ❖位置：每个用例下面，对应用例的路径
- ❖基本路径：一张图
- ❖简单的扩展点：可以合并到基本路径图
- ❖复杂扩展点：单独一张图，和基本路径图间链接

序列图绘制要点



简单扩展点：可以合并到基本路径图

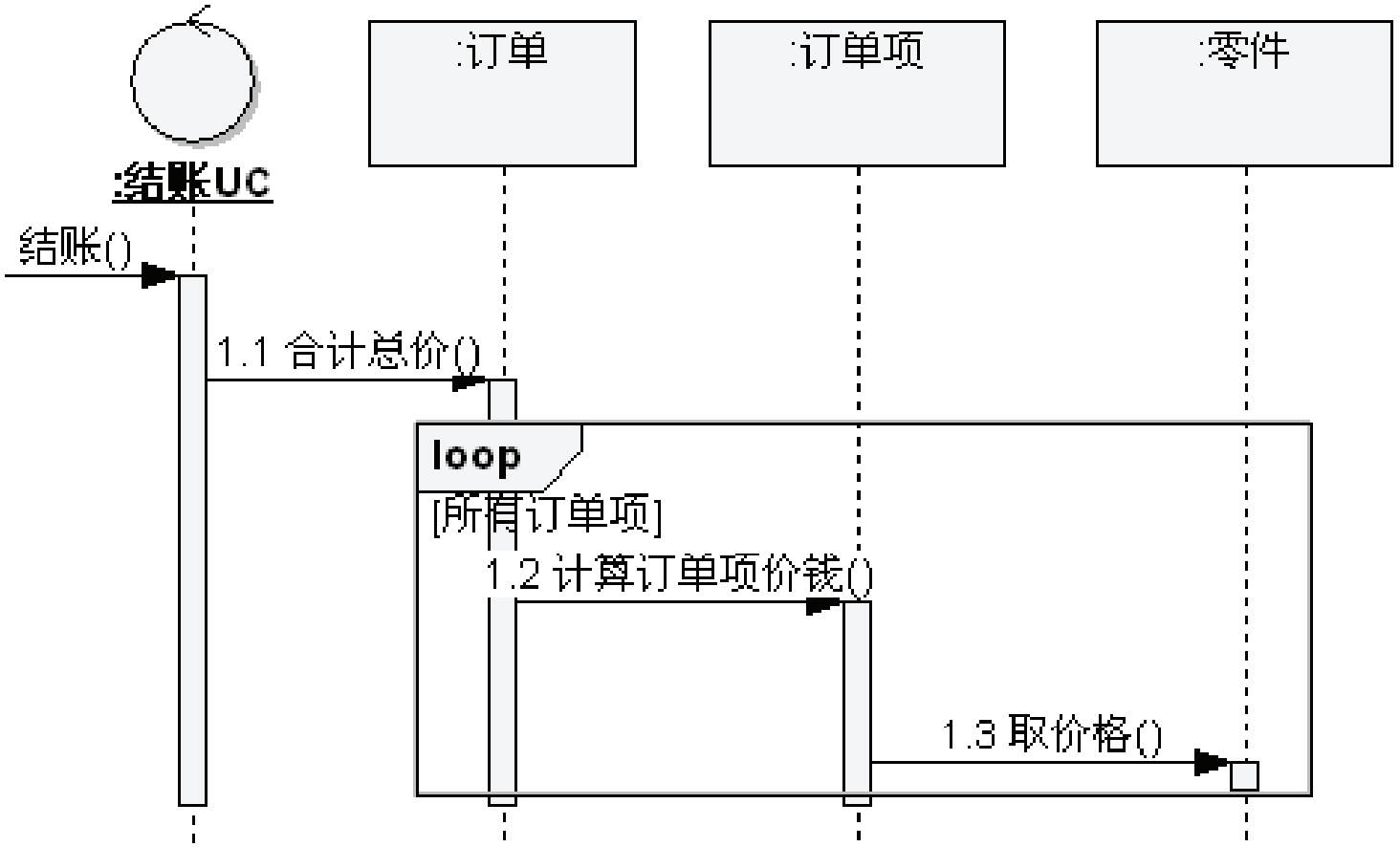
序列图绘制要点



复杂扩展点：单独一张图，在基本路径图引用

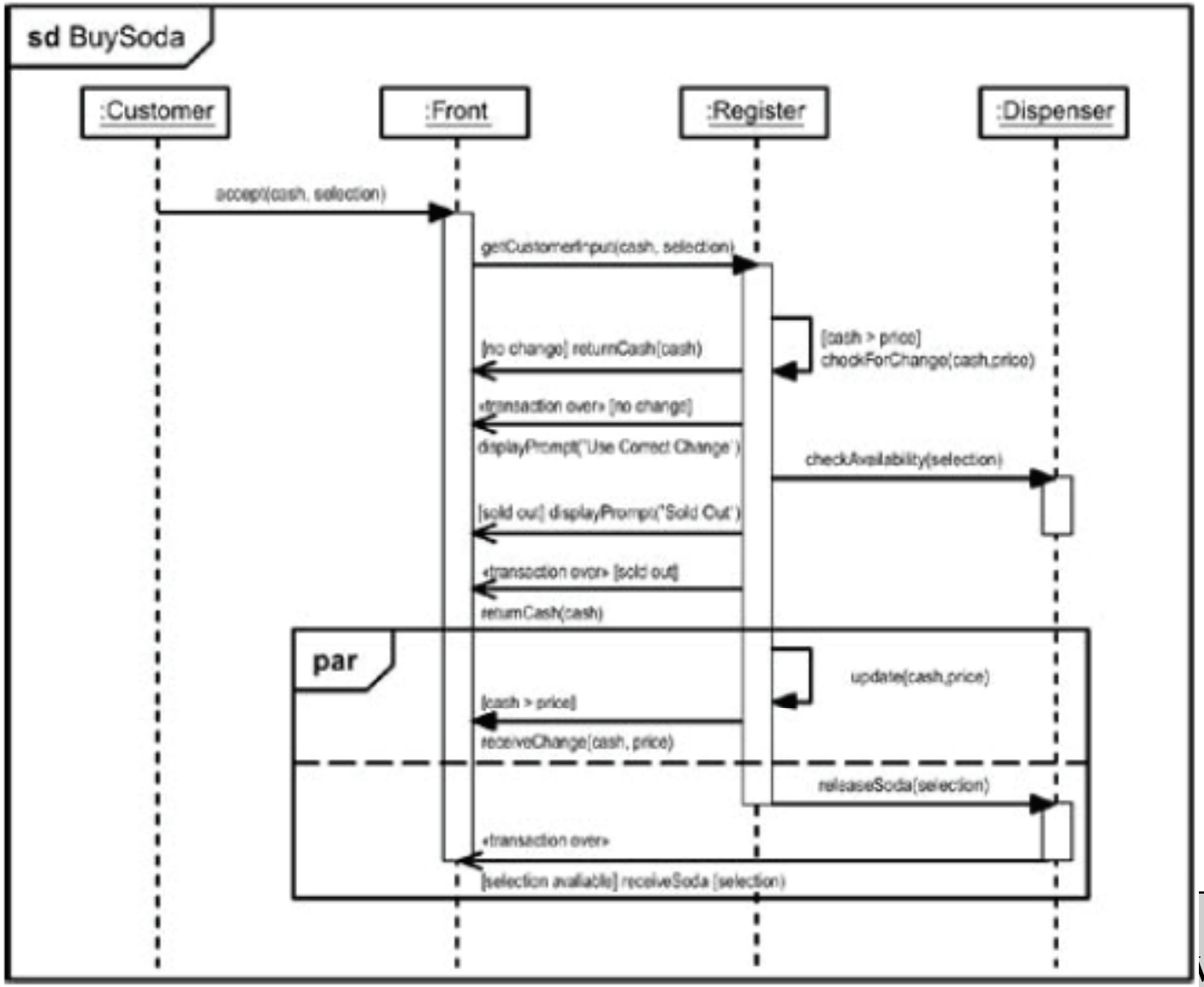
Include、Extend用例也适用

序列图绘制要点



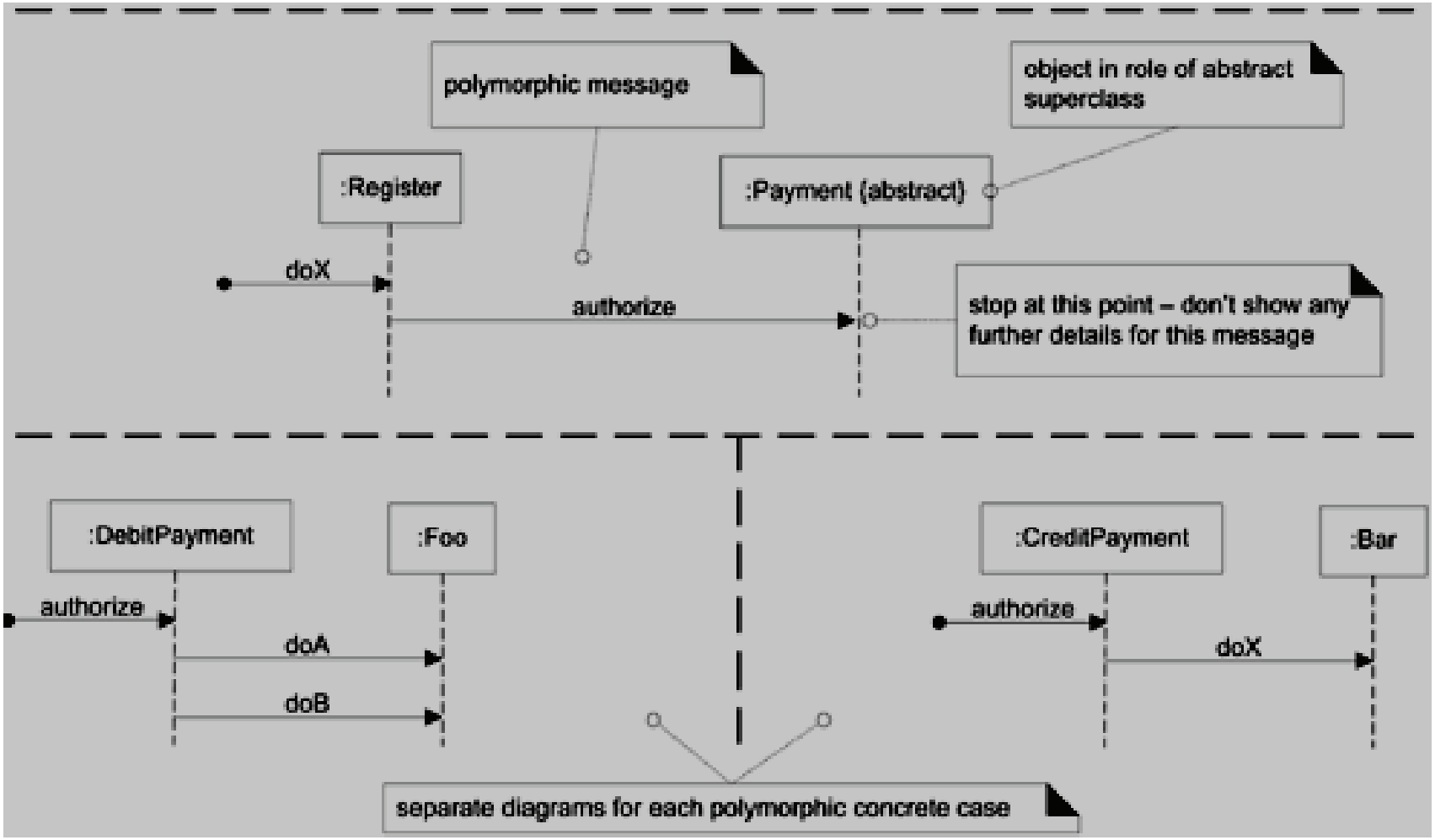
循环

序列图绘制要点


sd BuySoda
:Customer
:Front
:Register
:Dispenser
accept(cash, selection)
getCustomerInput(cash, selection)
[cash > price]
checkForChange(cash,price)
[no change] returnCash(cash)
«transaction over» [no change]
displayPrompt("Use Correct Change")
checkAvailability(selection)
[sold out] displayPrompt("Sold Out")
«transaction over» [sold out]
returnCash(cash)
par
update(cash,price)
[cash > price]
receiveChange(cash, price)
releaseSoda(selection)
«transaction over»
[selection available] receiveSoda [selection]

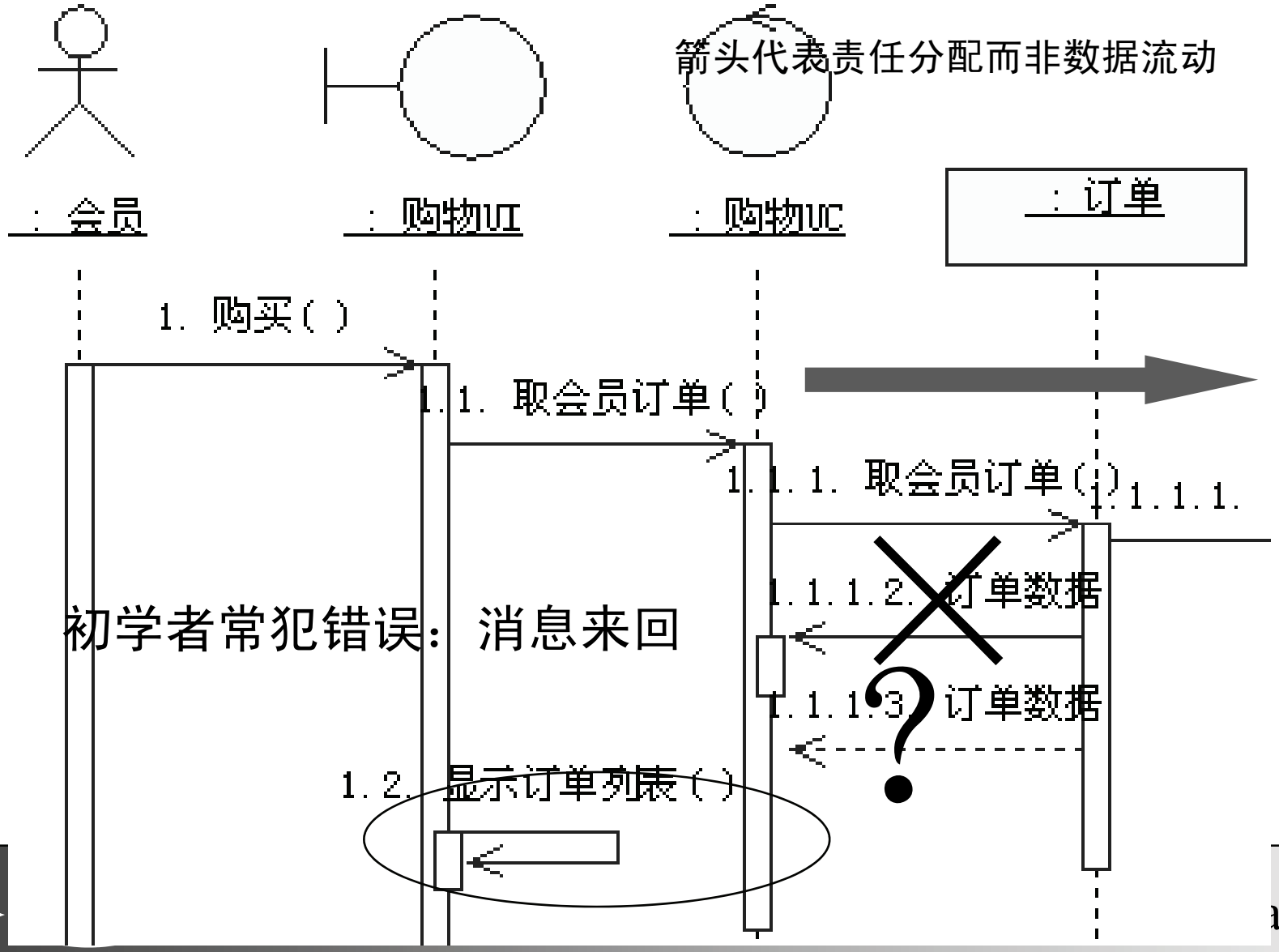
并行

序列图绘制要点

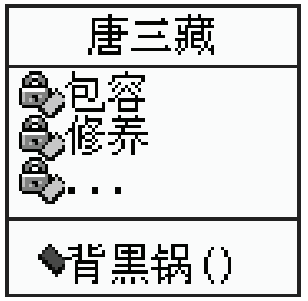

polymorphic message
object in role of abstract superclass
:Register
:Payment {abstract}
doX
authorize
stop at this point – don't show any further details for this message
:DebitPayment
:Foo
:CreditPayment
:Bar
authorize
doA
doB
authorize
doX
separate diagrams for each polymorphic concrete case

多态

序列图绘制要点


箭头代表责任分配而非数据流动
：会员
：购物UI
：购物UC
：订单
1. 购买( )
1.1. 取会员订单( )
1.1.1. 取会员订单( )
1.1.1.1.
1.1.1.2. 订单数据
1.1.1.3. 订单数据
1.2. 显示订单列表( )
初学者常犯错误：消息来回
?

责任分配

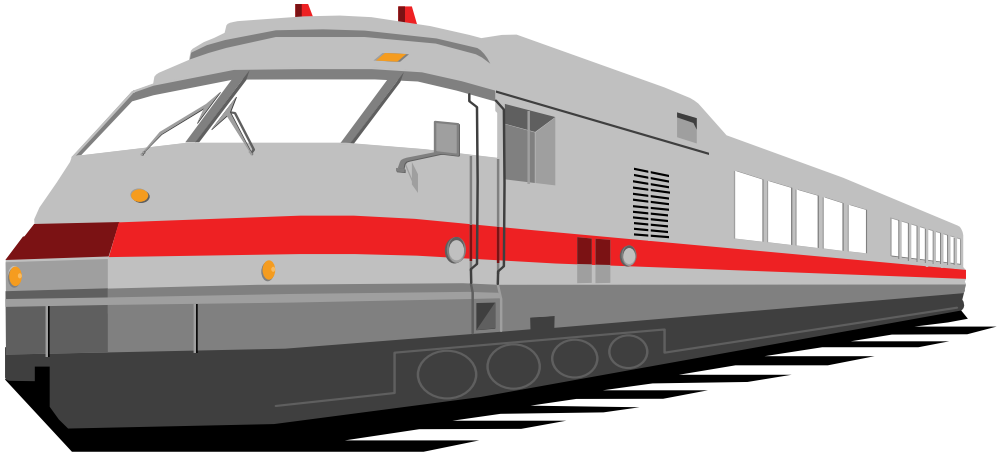


背黑锅我来，送死你去……

低耦合，高内聚

耦合

- 描述设计的组成部分之间的相互依赖

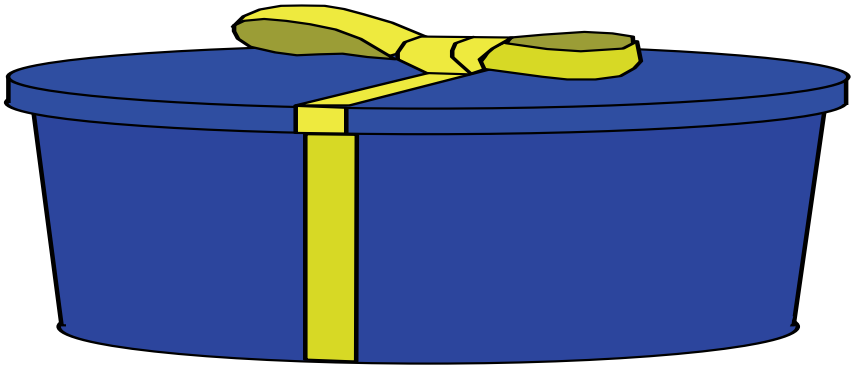


没有耦合，无法一起行动；耦合太高，无法转弯

类间要保持低耦合。目的：复用

内聚

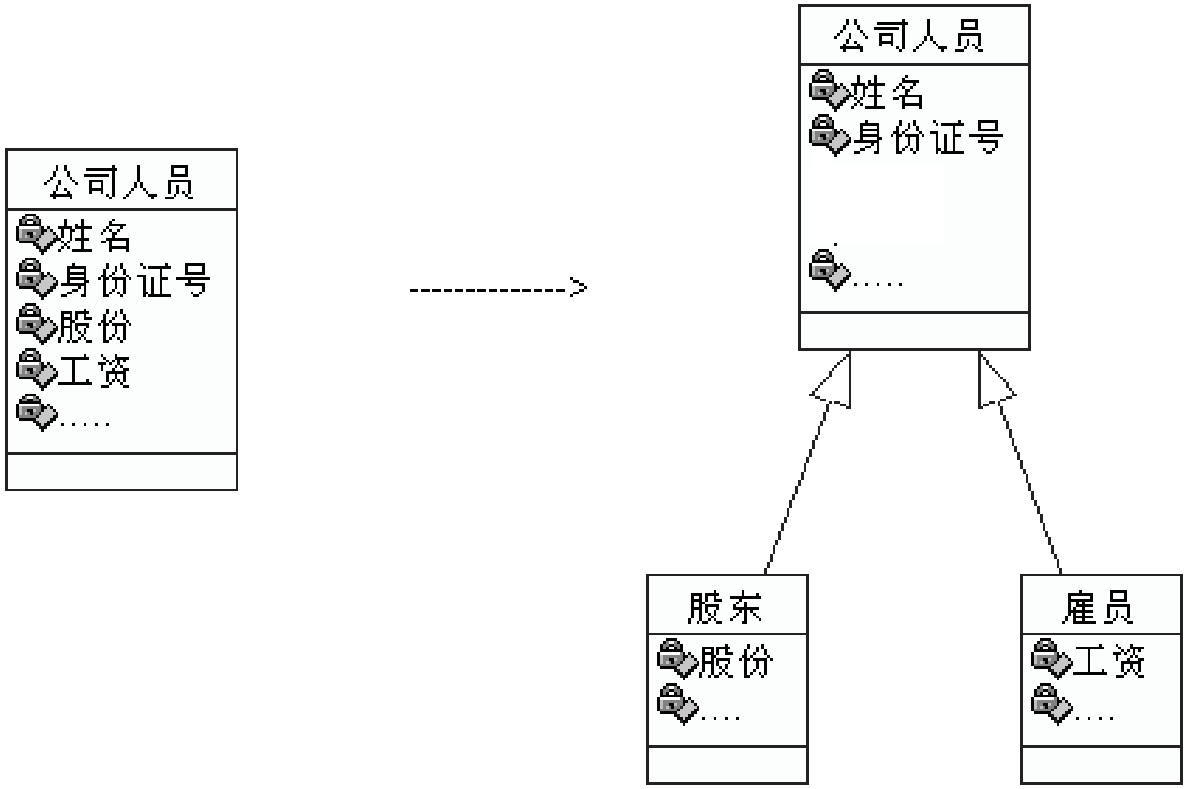
- 描述模块内各元素的紧密结合程度



类内各元素要保持高内聚

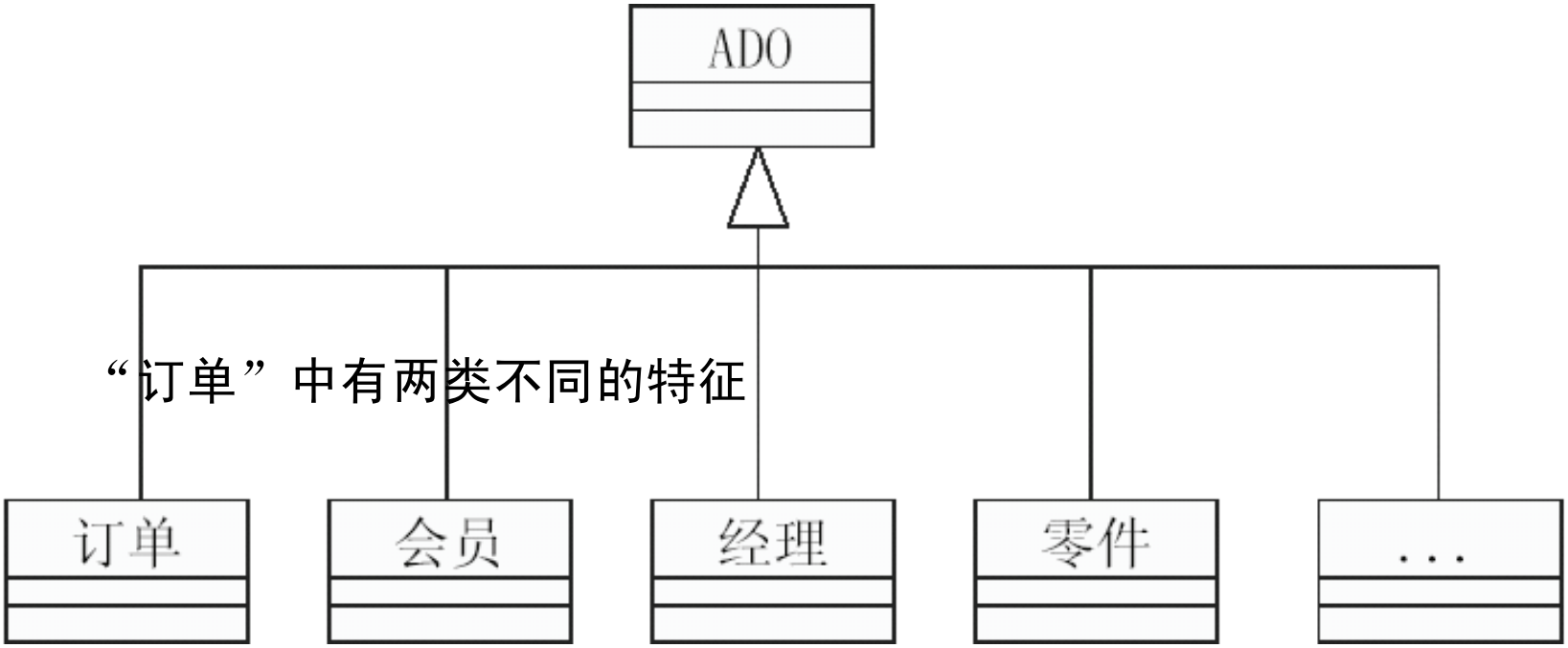
小类，短方法——明确责任

反例（1）



某些属性只对部分对象有意义

反例（2）



不同领域的类之间形成错误的关系

责任分配

- 专家原则——资源决定消息内容
- 老板原则——由老板发送消息给我
- 可视(Demeter)原则——只发消息给朋友

交互原则

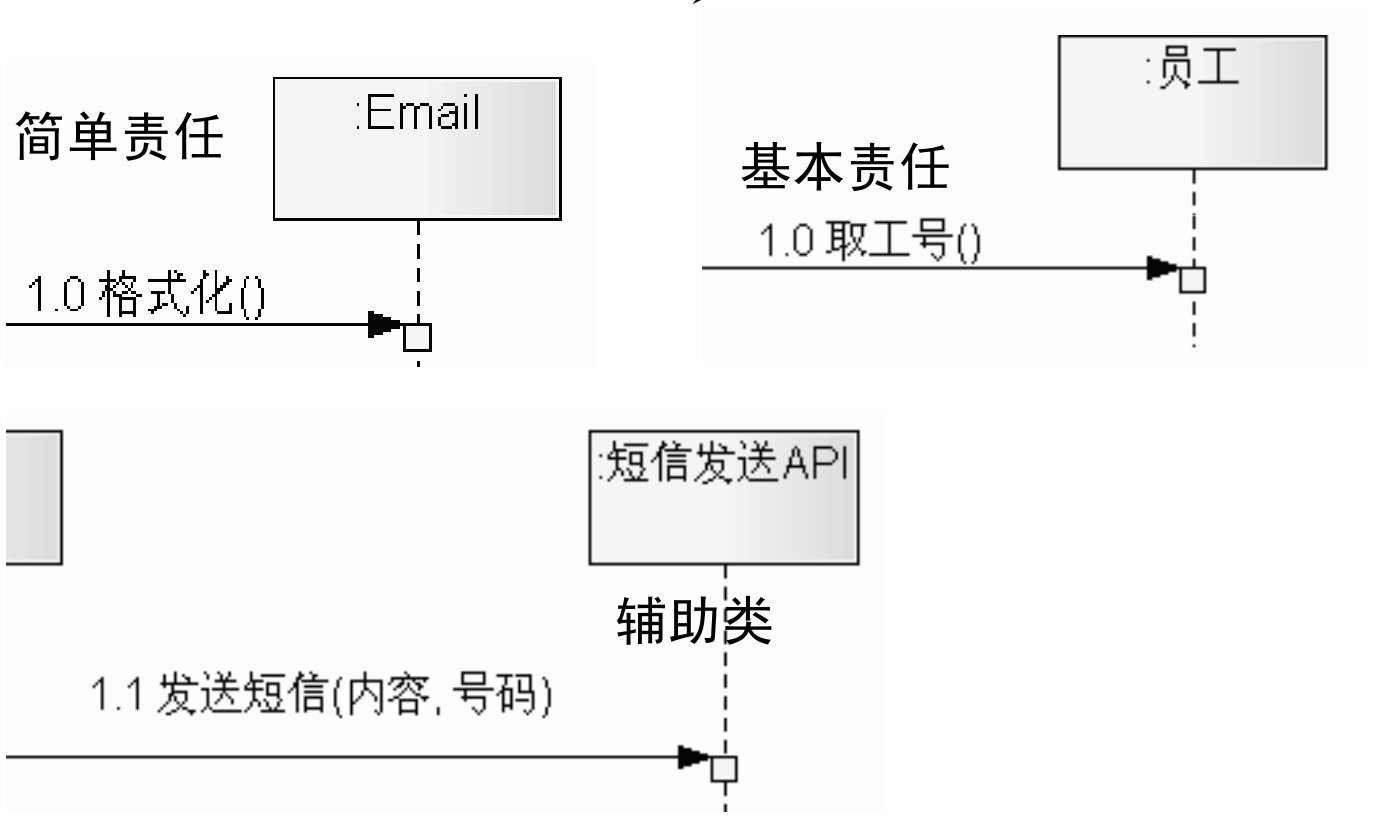
专家原则

- 根据资源分配责任

各尽其才，各施其能

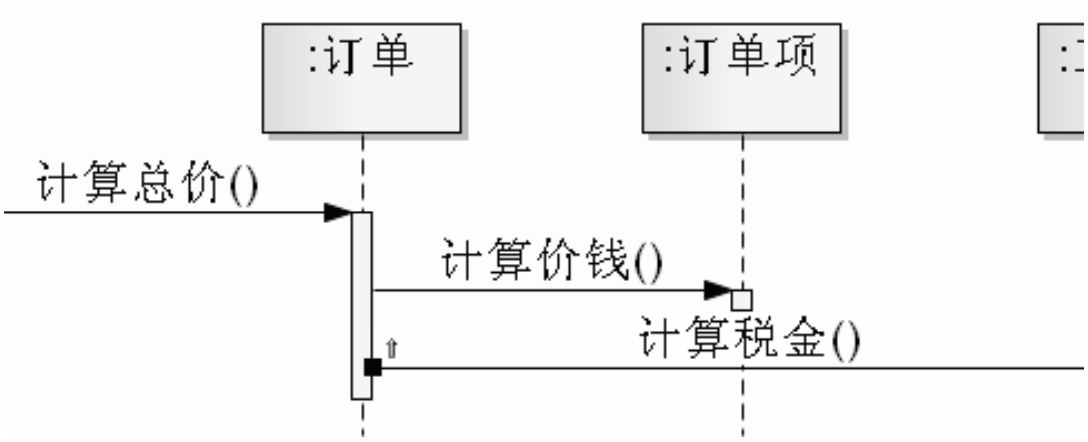
专家原则

考虑到基础设施，独立也只是某个层面上的说法



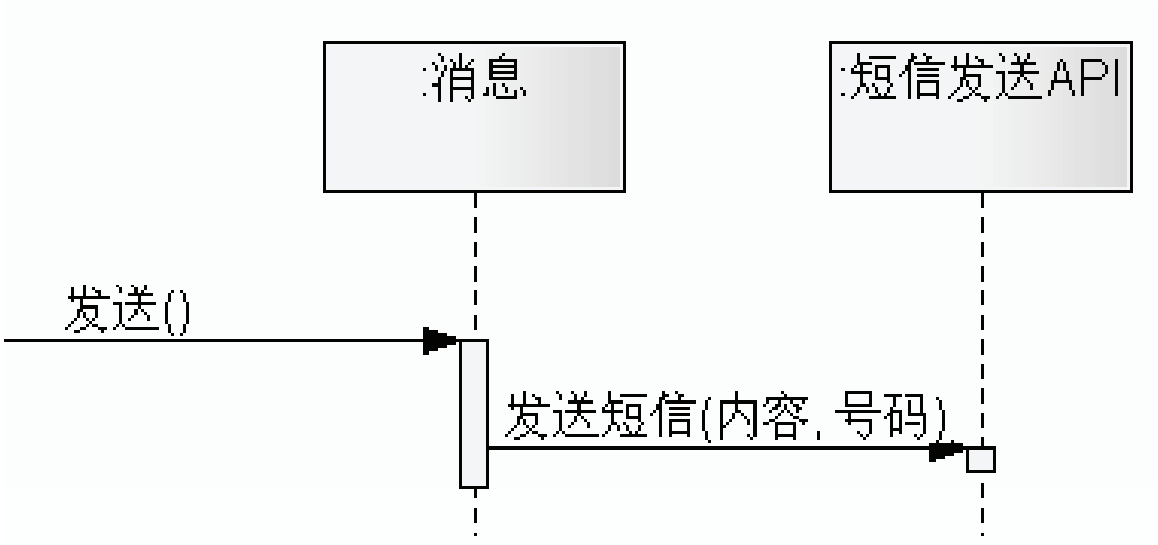
独立完成

专家原则



分解大责任

专家原则



委托给辅助类

专家原则

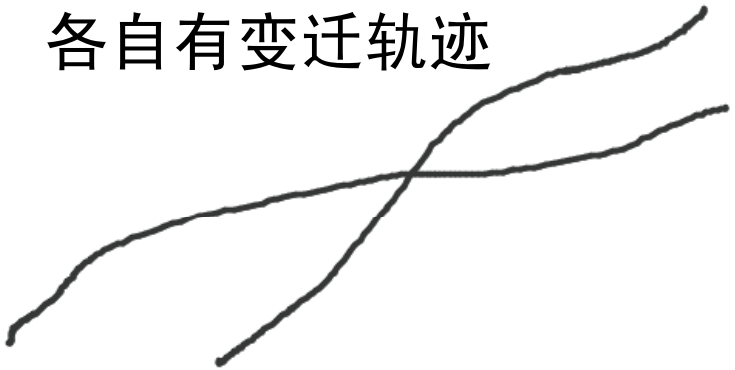
表 1　固体的线胀系数 $\alpha(℃^{-1})$

物　质	$t(℃)$	$\alpha(\times10^{-6})$
铝	25	25
金	25	14.2
银	25	19
铜	25	16.6
钨	25	4.5
铁	25	12.0
铂	25	9.0
黄铜(68 Cu, 32 Zn)	25	18～19
殷钢(36 Ni, 64 Fe)	0～100	0.8～12.8

不同的膨胀系数

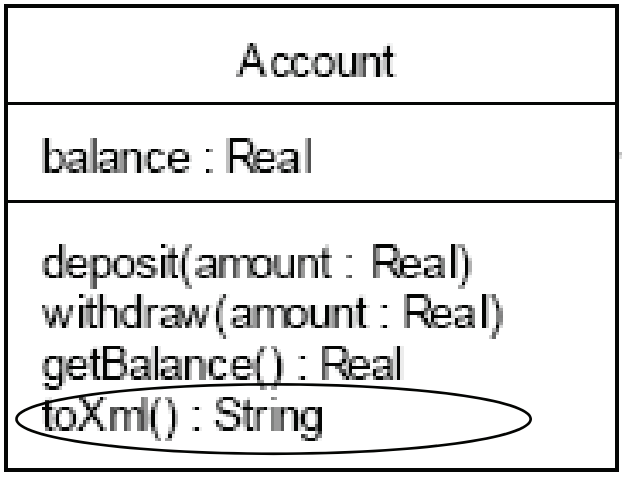
导致结合不能长久

各自有变迁轨迹



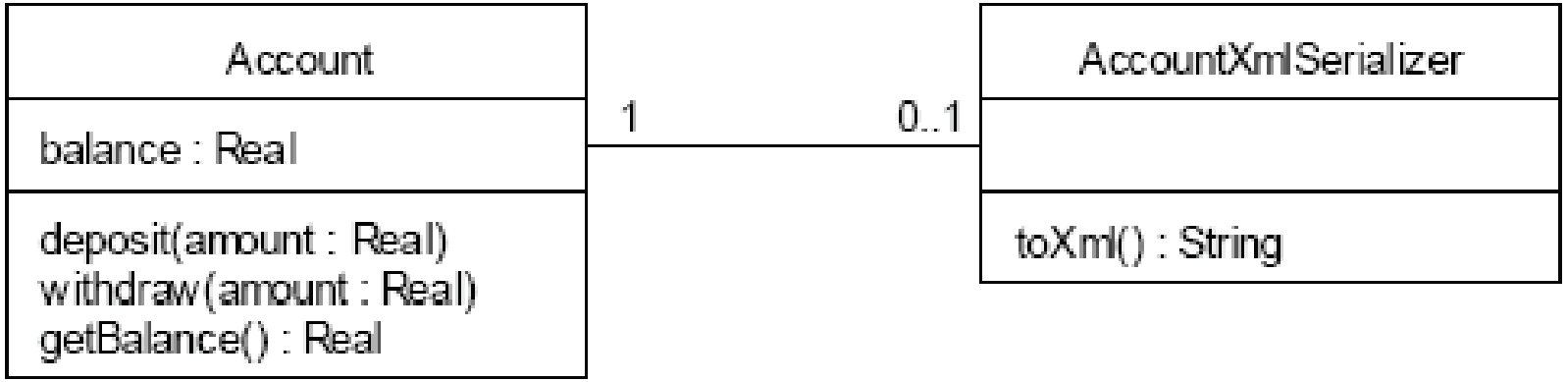
专家要专——SRP（单一责任原则）

专家原则



一个类只有一个变化原因

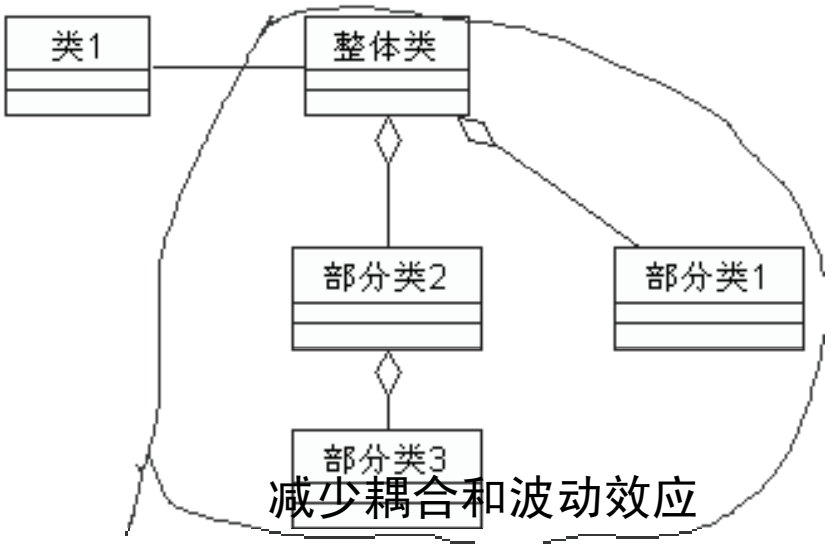
领域知识是瓶颈



SRP

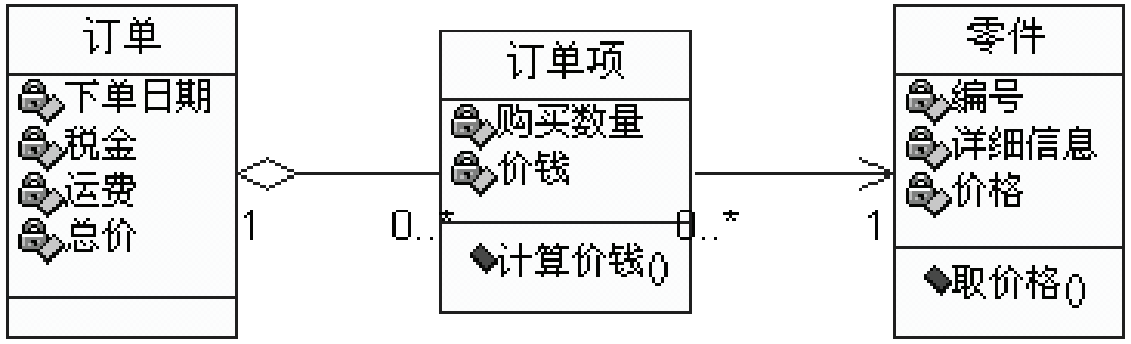
老板原则

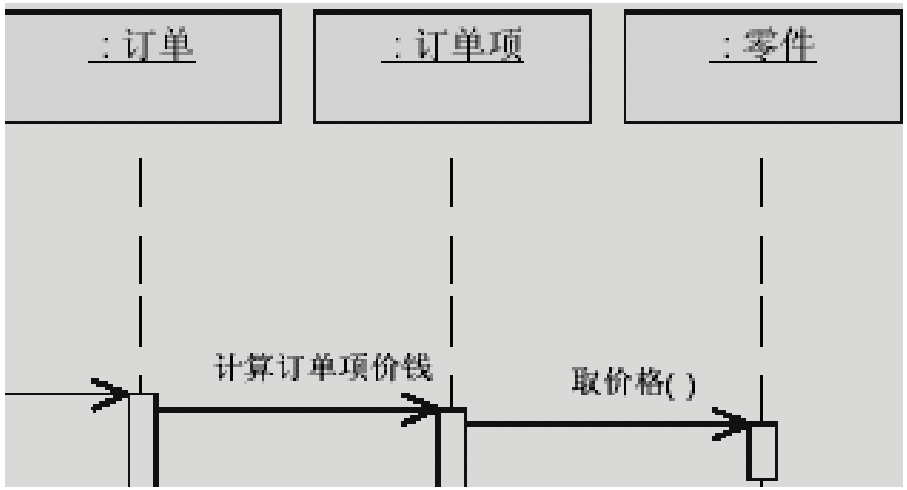
- 由老板传递消息
- 当出现以下情况时，发给A的消息先通过B处理和中转
  - B聚合A（Aggregation）
  - B组合A（ Composition ）



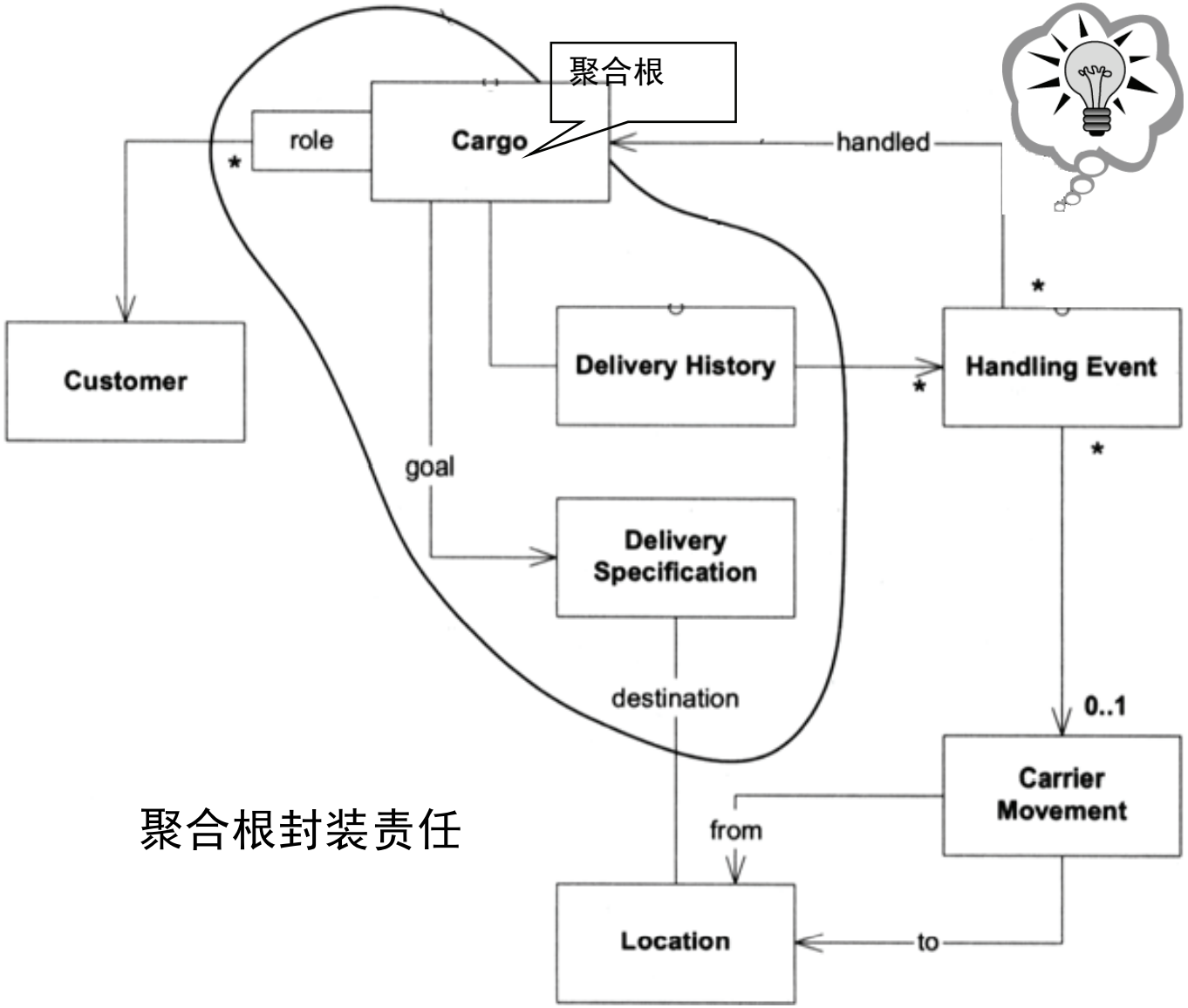
减少耦合和波动效应

老板原则


订单
下单日期
税金
运费
总价
1
0..*
订单项
购买数量
价钱
计算价钱()
0..*
1
零件
编号
详细信息
价格
取价格()

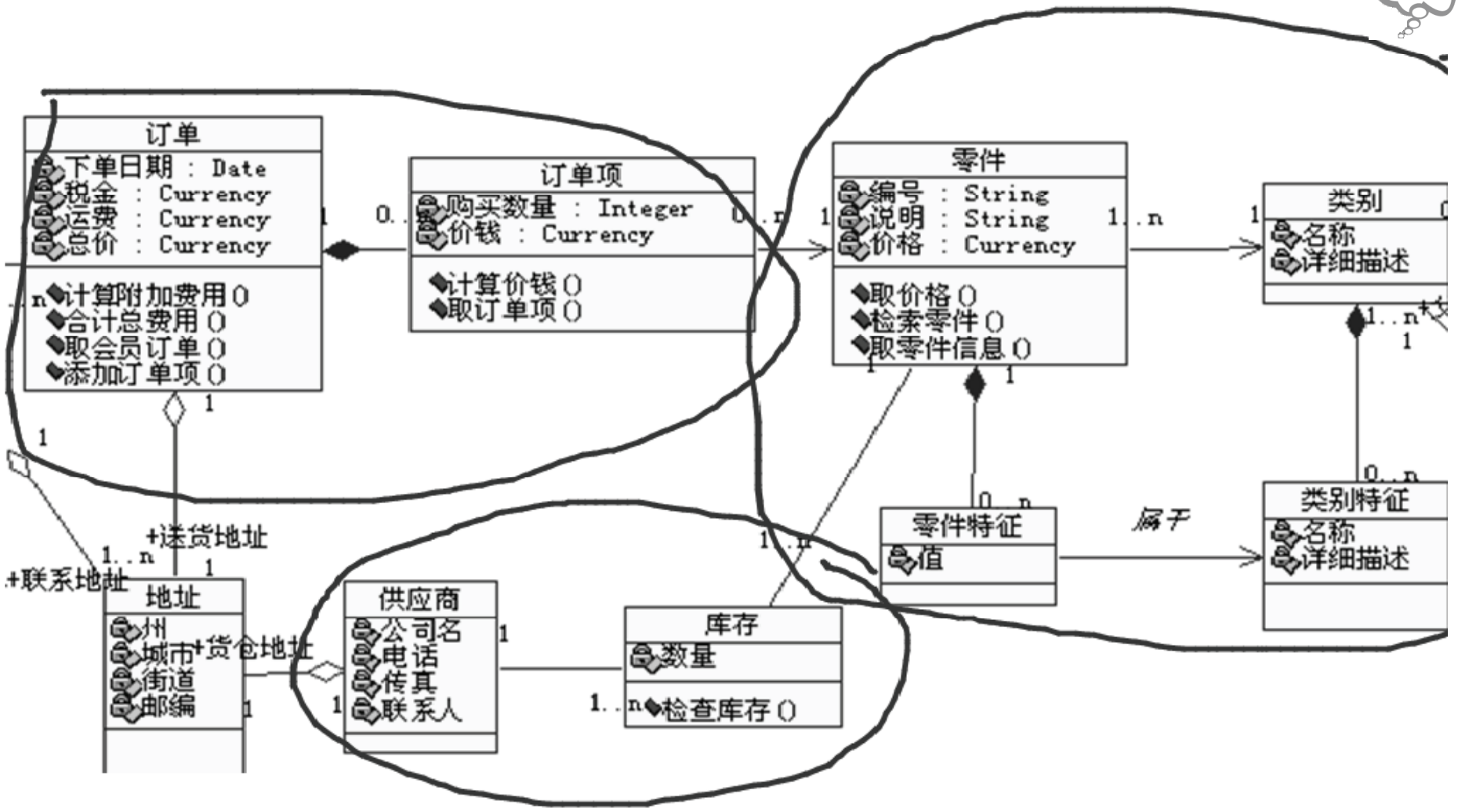

:订单
:订单项
:零件
计算订单项价钱
取价格( )

老板原则

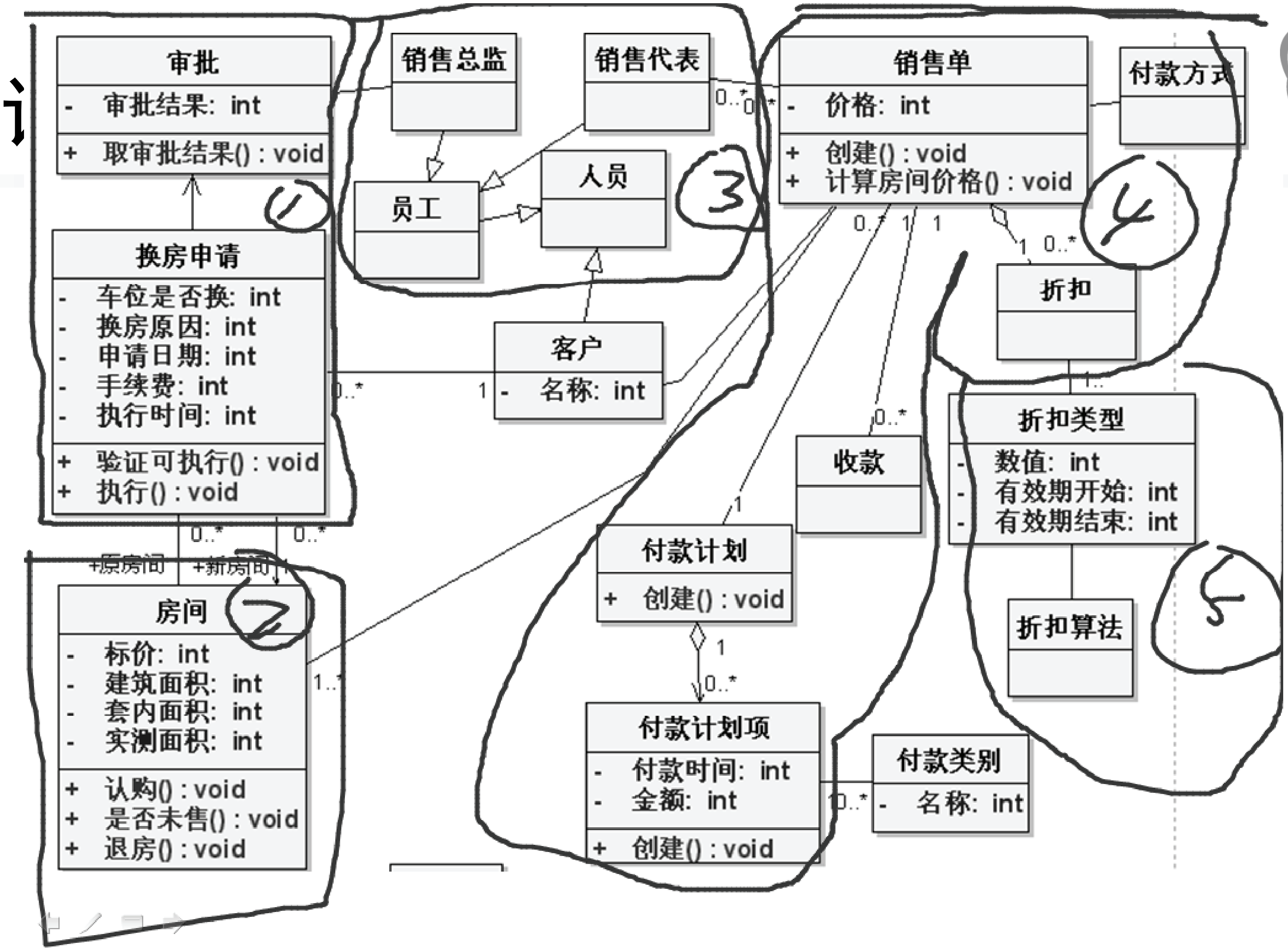


聚合根封装责任

聚合根

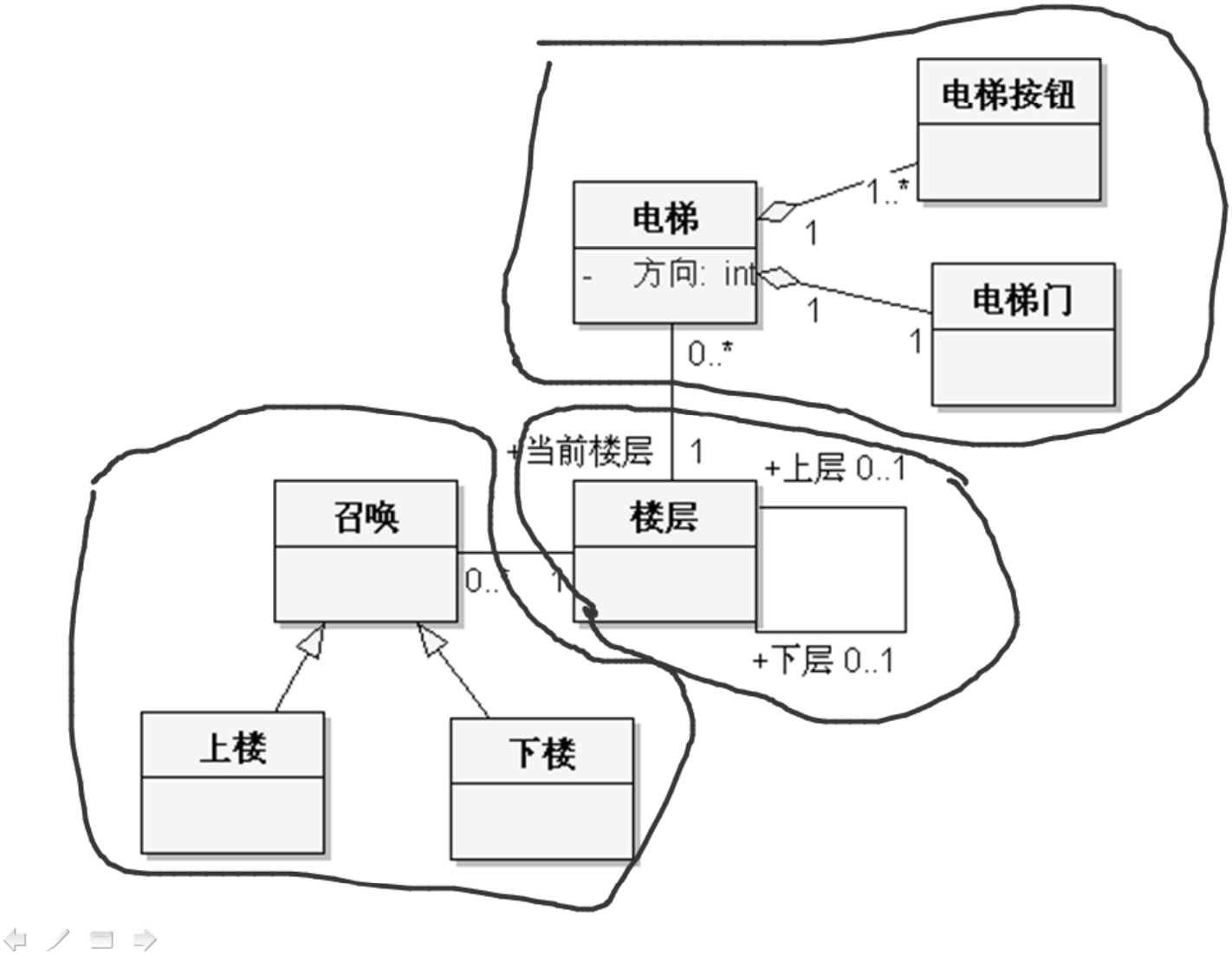

订单
下单日期 : Date
税金 : Currency
运费 : Currency
总价 : Currency
计算附加费用()
合计总费用()
取会员订单()
添加订单项()
订单项
购买数量 : Integer
价钱 : Currency
计算价钱()
取订单项()
零件
编号 : String
说明 : String
价格 : Currency
取价格()
检索零件()
取零件信息()
类别
名称
详细描述
零件特征
值
属于
类别特征
名称
详细描述
地址
州
城市
街道
邮编
+送货地址
+联系地址
+货仓地址
供应商
公司名
电话
传真
联系人
库存
数量
检查库存()

知识的分区


审批
- 审批结果: int
+ 取审批结果() : void
换房申请
- 车位是否换: int
- 换房原因: int
- 申请日期: int
- 手续费: int
- 执行时间: int
+ 验证可执行() : void
+ 执行() : void
房间
- 标价: int
- 建筑面积: int
- 套内面积: int
- 实测面积: int
+ 认购() : void
+ 是否未售() : void
+ 退房() : void
+原房间
+新房间
销售总监
销售代表
员工
人员
客户
- 名称: int
销售单
- 价格: int
+ 创建() : void
+ 计算房间价格() : void
付款方式
折扣
折扣类型
- 数值: int
- 有效期开始: int
- 有效期结束: int
折扣算法
收款
付款计划
+ 创建() : void
付款计划项
- 付款时间: int
- 金额: int
+ 创建() : void
付款类别
- 名称: int

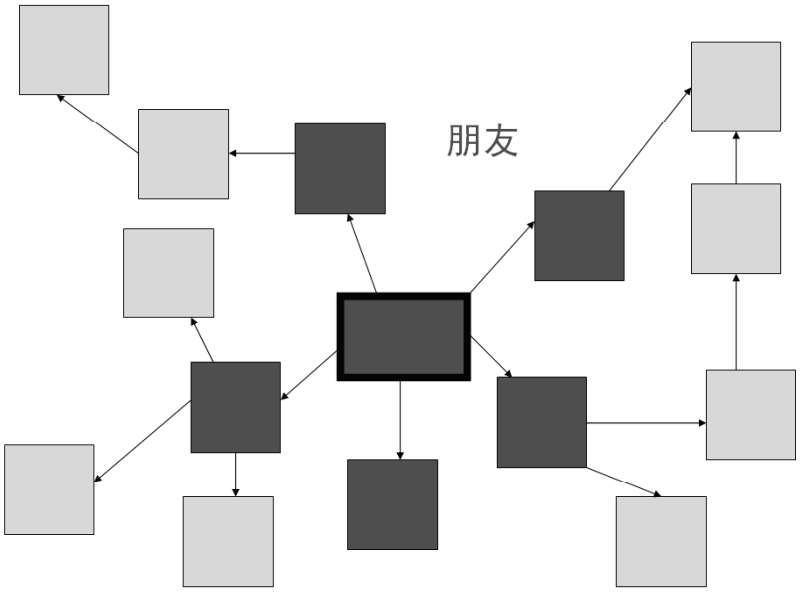
知识的分区

知识



知识的分区

可视原则



不要和陌生人说话

只能发消息给：

- 自己
- 方法参数中的对象
- 属性引用的对象（关联）
- 你创建的对象

Demeter法则

可视原则

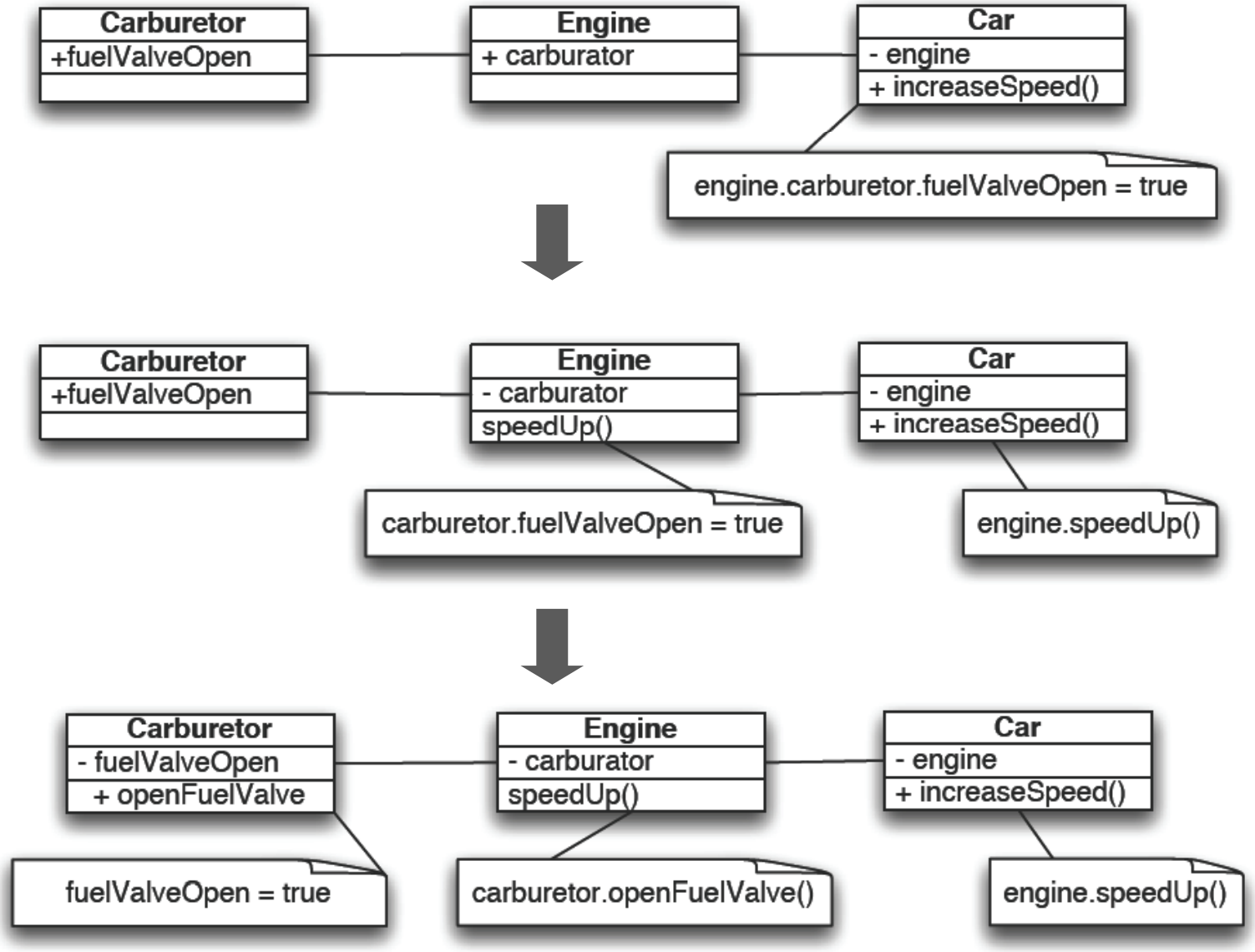
```
public class sample {
  private ObjectA a;
  private int function();
  public void example(ObjectB b) {
  ObjectC c;
  in f = function();

  b.invert();

  a = new.ObjectA();

  a.setActive();

  c.print();

  }
}
```

The Law of Demeter for functions states that any method of an object should call only methods belonging to:

- *itself*
- *any parameters that were passed in to the method*
- *any objects it created*
- *any directly held component object*

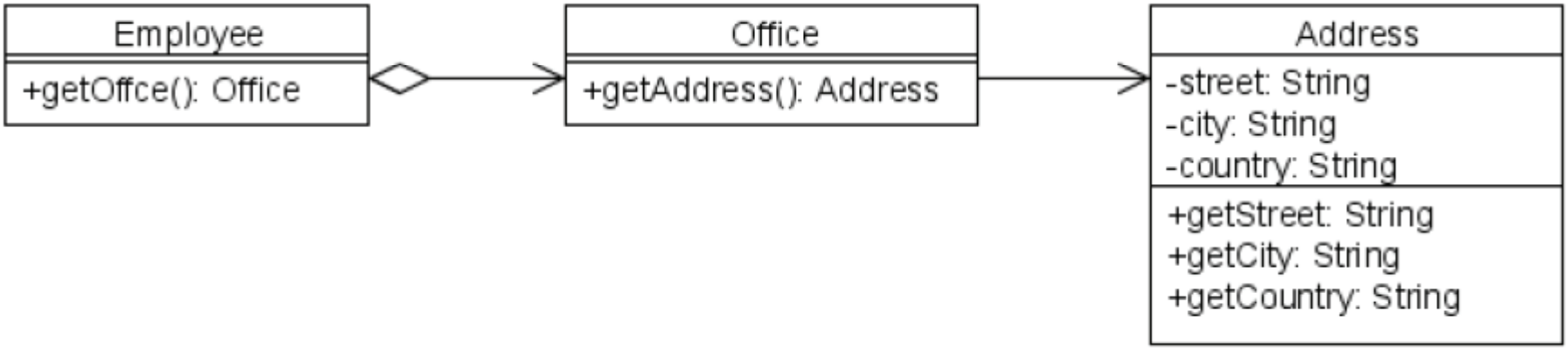
Demeter法则

可视原



Demeter法则

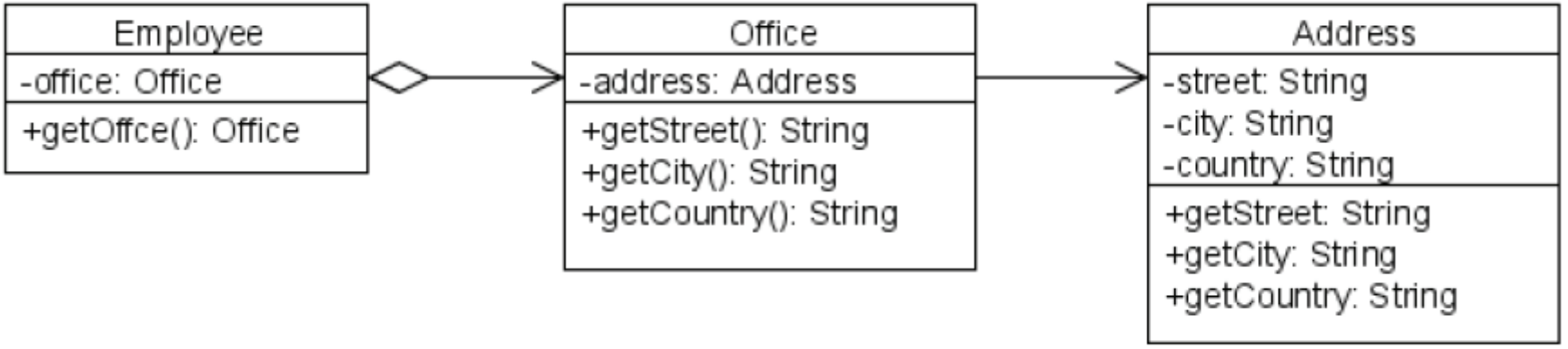
可视原则




```
String employeeStreet = this.office.getAddress().getStreet();
```

Demeter法则

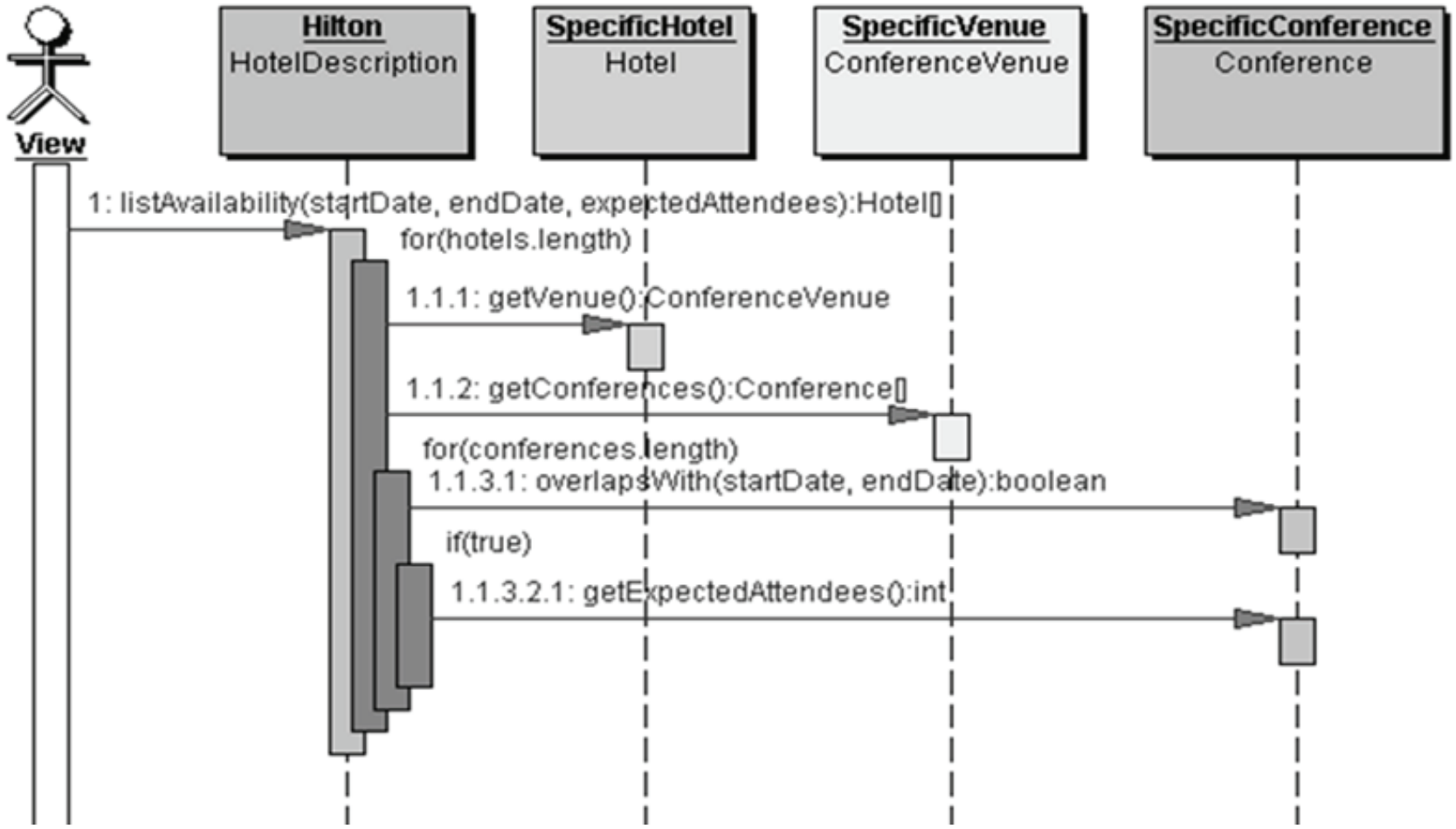
可视原则




```
String employeeStreet = this.office.getStreet();
```

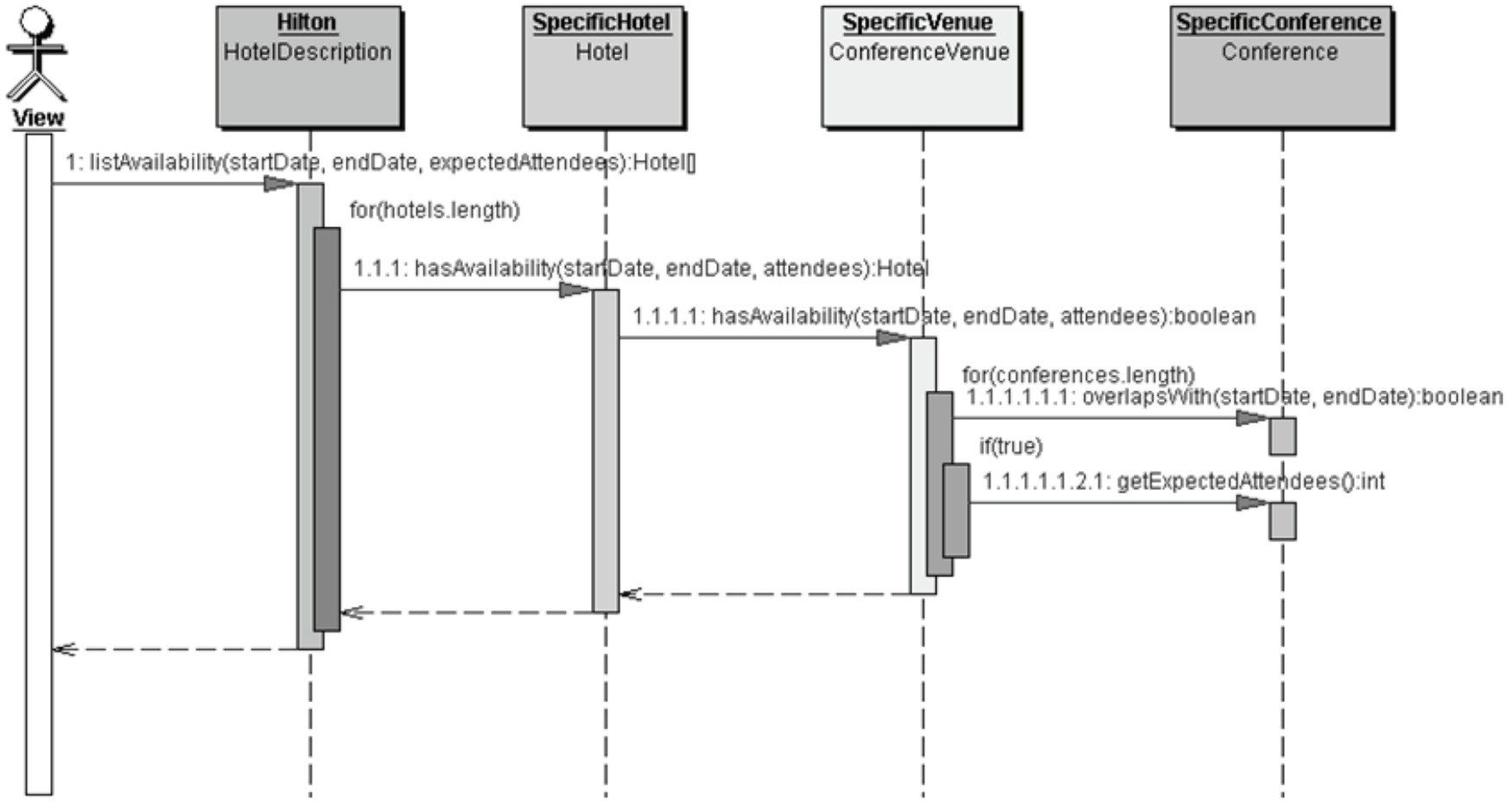
Demeter法则－修改

可视原则


View
Hilton
HotelDescription
SpecificHotel
Hotel
SpecificVenue
ConferenceVenue
SpecificConference
Conference
1: listAvailability(startDate, endDate, expectedAttendees):Hotel[]
for(hotels.length)
1.1.1: getVenue():ConferenceVenue
1.1.2: getConferences():Conference[]
for(conferences.length)
1.1.3.1: overlapsWith(startDate, endDate):boolean
if(true)
1.1.3.2.1: getExpectedAttendees():int

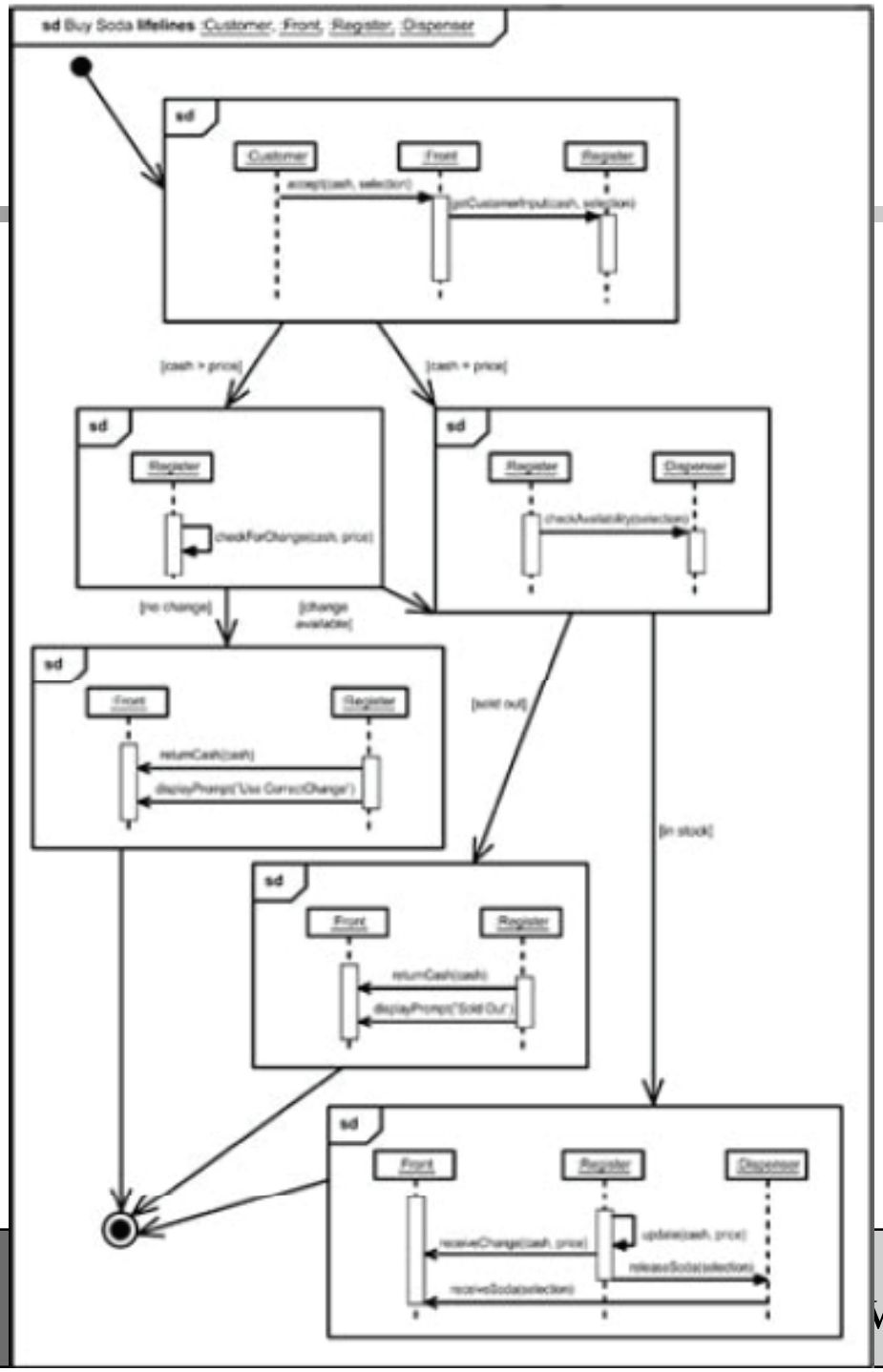
违反可视原则

可视原则

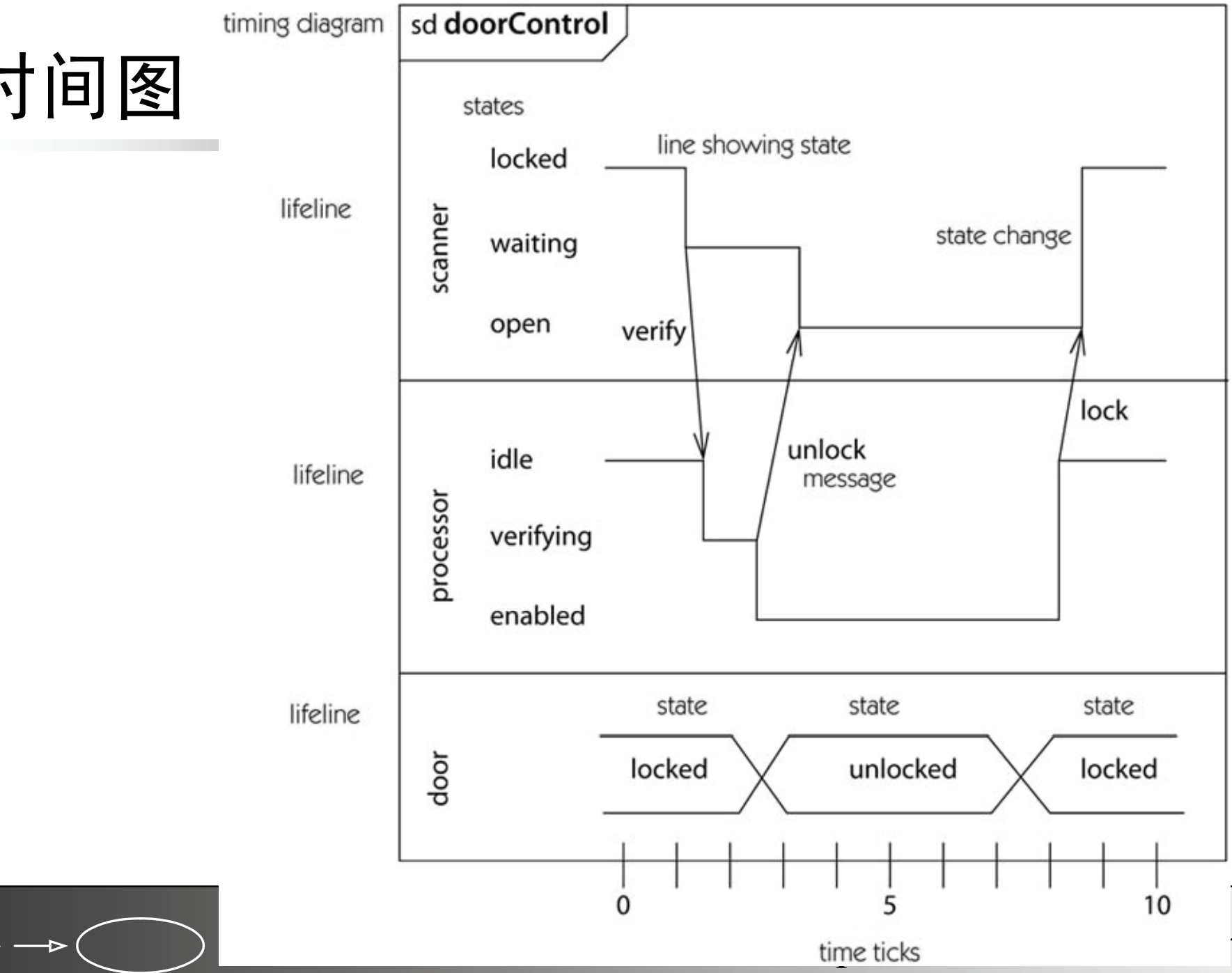

View
Hilton
HotelDescription
SpecificHotel
Hotel
SpecificVenue
ConferenceVenue
SpecificConference
Conference
1: listAvailability(startDate, endDate, expectedAttendees):Hotel[]
for(hotels.length)
1.1.1: hasAvailability(startDate, endDate, attendees):Hotel
1.1.1.1: hasAvailability(startDate, endDate, attendees):boolean
for(conferences.length)
1.1.1.1.1.1: overlapsWith(startDate, endDate):boolean
if(true)
1.1.1.1.1.2.1: getExpectedAttendees():int

遵守可视原则

交互概述图


sd Buy Soda lifelines :Customer, :Front, :Register, :Dispenser
sd
Customer
Front
Register
accept(cash, selection)
getCustomerInput(cash, selection)
[cash > price]
[cash = price]
sd
Register
checkForChange(cash, price)
sd
Register
Dispenser
checkAvailability(selection)
[no change]
[change available]
sd
Front
Register
returnCash(cash)
displayPrompt("Use Correct Change")
[sold out]
[in stock]
sd
Front
Register
returnCash(cash)
displayPrompt("Sold Out")
sd
Front
Register
Dispenser
receiveChange(cash, price)
update(cash, price)
releaseSoda(selection)
receiveSoda(selection)

时间图


timing diagram
sd doorControl
states
line showing state
lifeline
scanner
locked
waiting
open
verify
state change
lifeline
processor
idle
verifying
enabled
unlock
message
lock
lifeline
door
state
state
state
locked
unlocked
locked
0
5
10
time ticks