交互建模

Modelling Interactions

清华大学软件学院 刘璘

interaction diagrams (交互图)

- An **interaction** is a behavior that comprises a set of messages exchanged among a set of objects within a context to accomplish a purpose.

交互是指对象为实现某种目的而彼此传递消息的行为

- Interaction diagrams describe how groups of objects collaborate in some behavior.

交互图描述对象之间如何进行协作

- The UML defines many forms of interaction diagram, of which the most common is: Sequence Diagram.

UML 定义多种交互图：其中应用最广的为顺序图（也称为时序图）

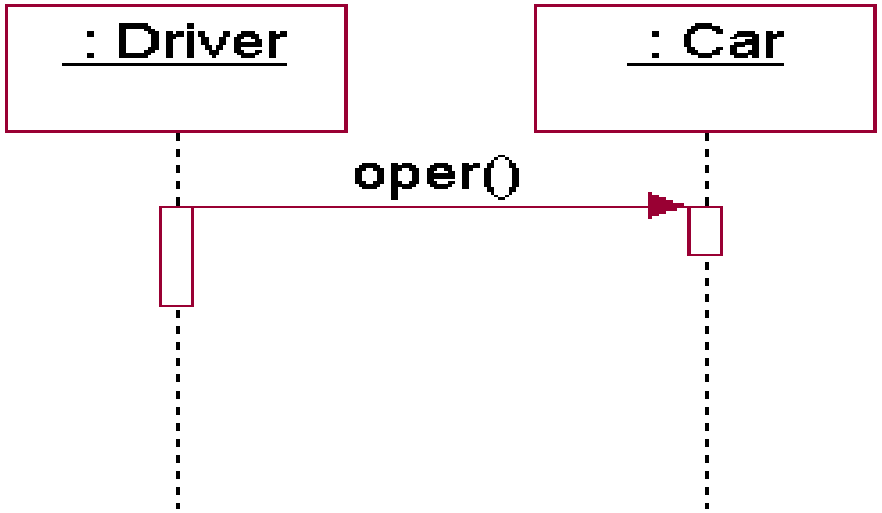
UML sequence diagrams

- **sequence diagram**: an "interaction diagram" that models a single scenario executing in the system
  - perhaps 2nd most used UML diagram (behind class diagram)
- relation of UML diagrams to other exercises:
  - use cases -> sequence diagrams
  - CRC cards -> class diagram

Sequence Diagram(顺序图)

- A **sequence diagram** is a diagram that shows object interactions arranged in **time** sequence. 顺序图按时间次序表示对象间的交互。

- In particular, it shows the **objects** participating in an interaction and the **sequence of messages** exchanged. 主要表示有哪些对象参加交互，以及消息传递的序列。



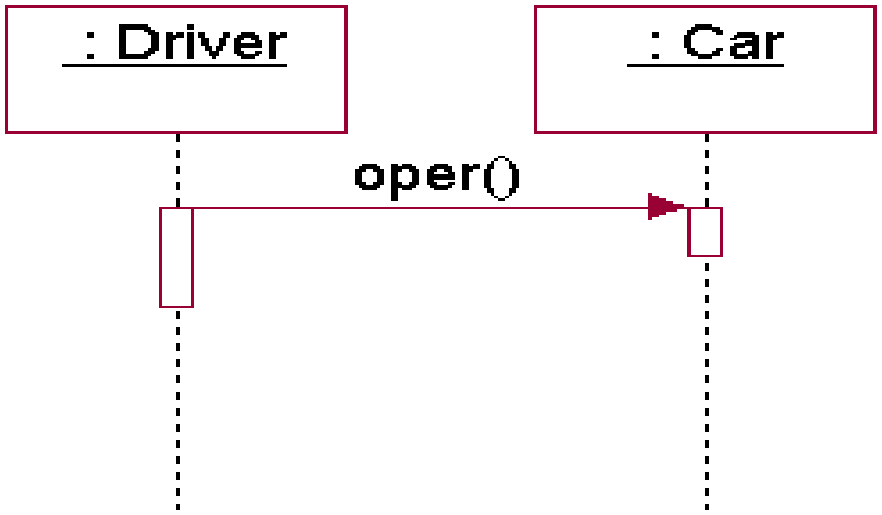
Sequence Diagram(顺序图)

- A **message** is a specification of a communication between objects that conveys information and value.

消息用于描述对象间的交互操作和值传递过程活动

Message types:

- sync　同步消息
- Async　异步消息
- Time-out　超时等待
- Uncommitted / Balking 阻塞



Sequence Diagram and Use Cases 顺序图与用例的关系 I

- A Sequence Diagram (SD) captures the behavior of a single scenario. There will be a SD for each Use Case in the system.

  顺序图表达单个情景实例的行为。每个用例对应一个顺序图。

- We draw SDs in order to document how objects collaborate to produce the functionality of each use case.

  顺序图表达对象间如何协作完成用例所描述的功能。

- The SDs show the data and messages which pass across the system boundary and the messages being sent from one object to another in order to achieve the overall functionality of the Use Case.

  顺序图用于表示为了完成用例而在系统边界输入输出的数据以及消息及对象间的消息传递。

1. The customer requests a funds transfer.
2. The system asks the user to identify the accounts between which funds are to be transferred and the transfer amount.
3. Customer selects the account to transfer funds from, the account to transfer to, and then indicates the amount of funds to transfer.
4. The system checks the account from which funds are to be transferred and confirms that sufficient funds are available.
5. The amount is debited to the account from which funds are to be transferred and credited to the account previously selected by the customer.

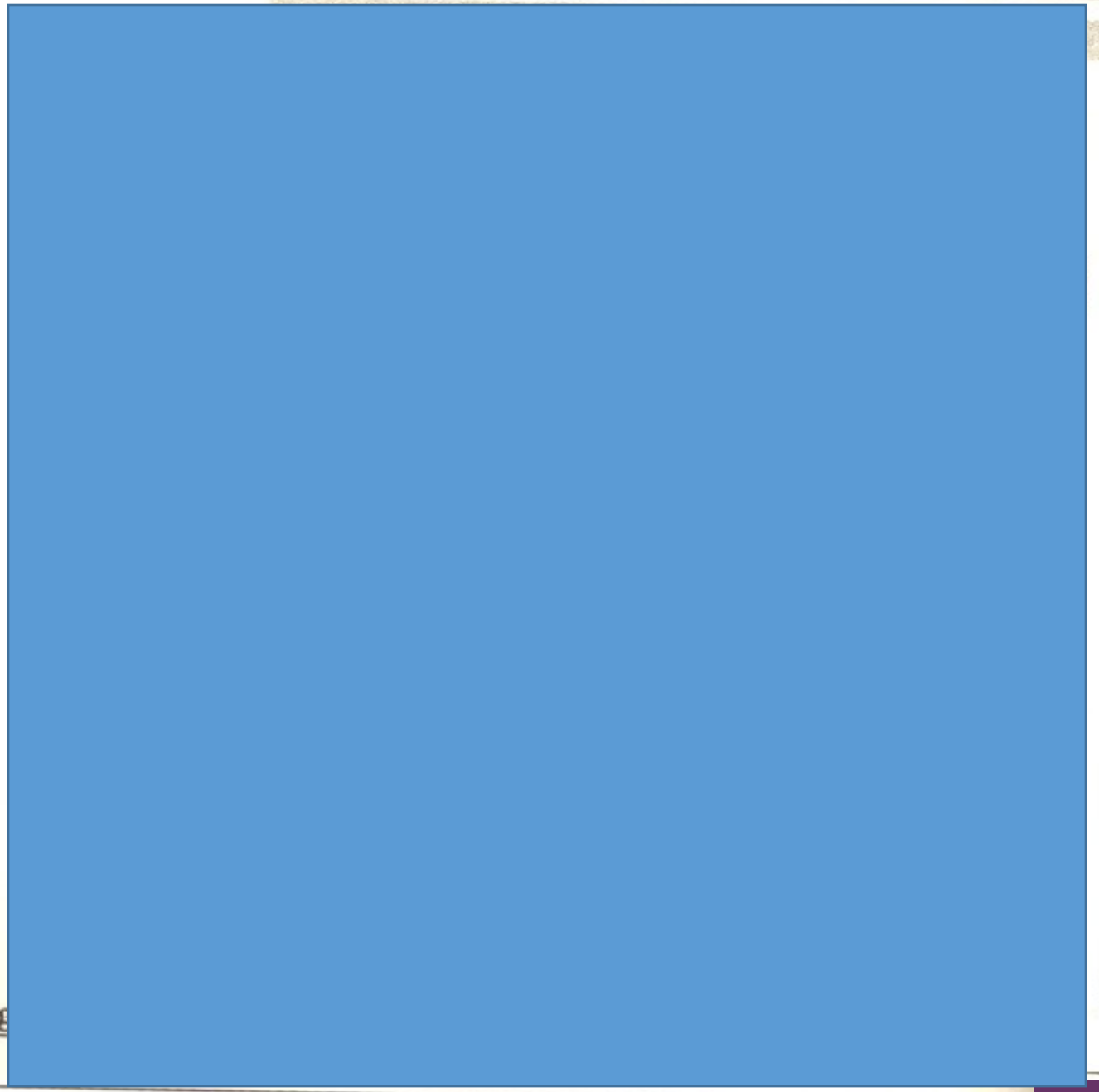


Figure 7-5 Transfer funds sequenc

Sequence Diagram and Use Cases 顺序图与用例的关系 II

- A Sequence Diagram can be seen as an expansion of a use case to the lowest possible level of detail.

  顺序图可帮助分析人员对用例图进行扩展、细化和补遗

- Sequence diagrams can be drawn at different levels of details and also to meet different purposes at several stages in the development life-cycle.

  顺序图可用于开发周期的不同阶段，服务于不同目的，描述不同粒度的行为

- Analysis Sequence Diagrams normally **do not**
  - include design objects 分析阶段的顺序图不包含设计对象
  - Specify message signatures in any detail 分析阶段的顺序图不关注消息参数

Drawing Sequence Diagram 绘制顺序图

- The name of the class and the name of the instance of that class (object), if required, at the top of the column that represents it.

  在顺序图顶端写出类名和实例（对象）名

- Including the arrows marked Return makes it easier to follow the flow of control from object to object.

  在图中包含表示返回信息的箭头便于跟踪对象间的控制流

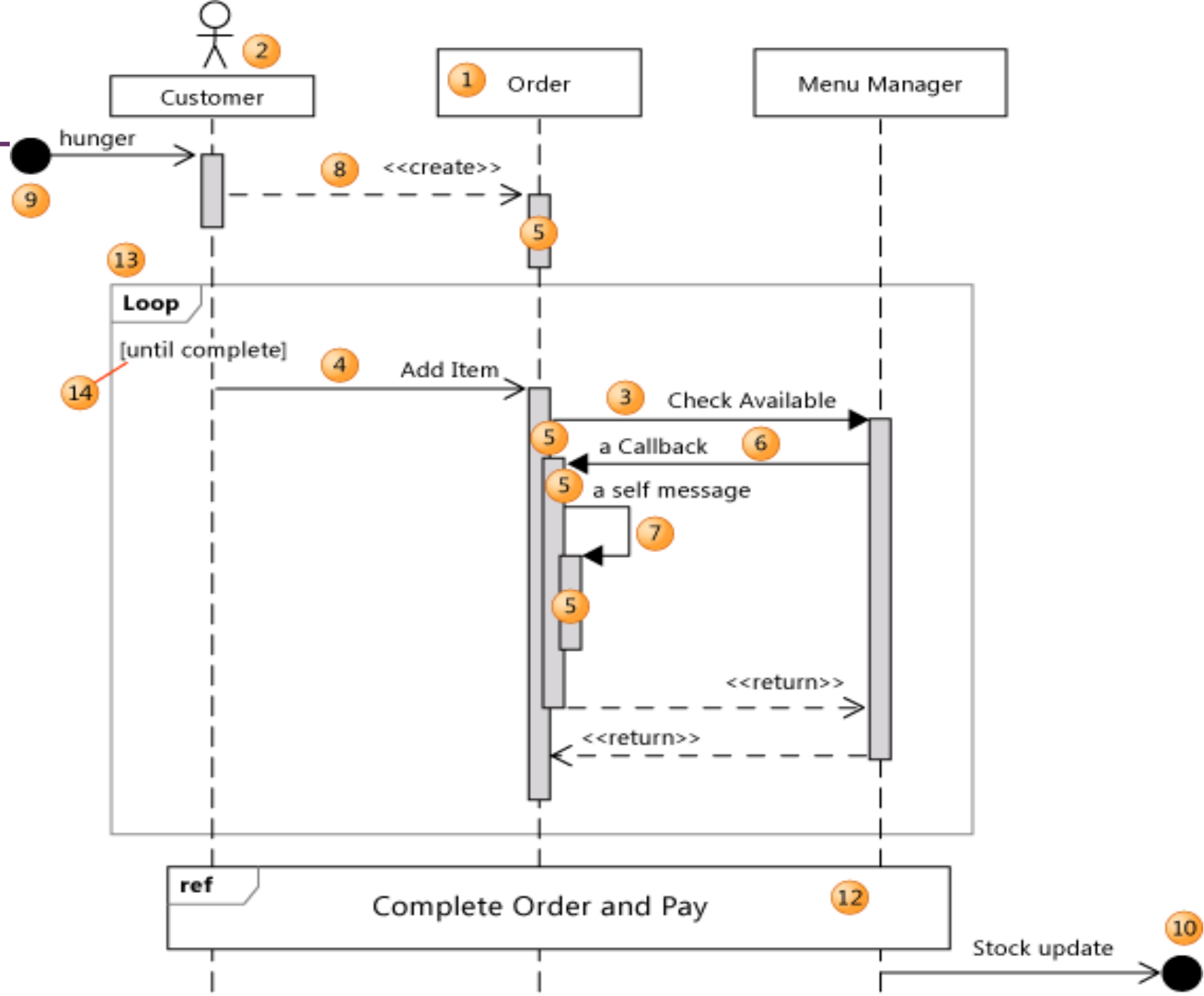
- The vertical line for each object represents its lifeline.

  每个对象下面的竖线表示该对象的生命线。

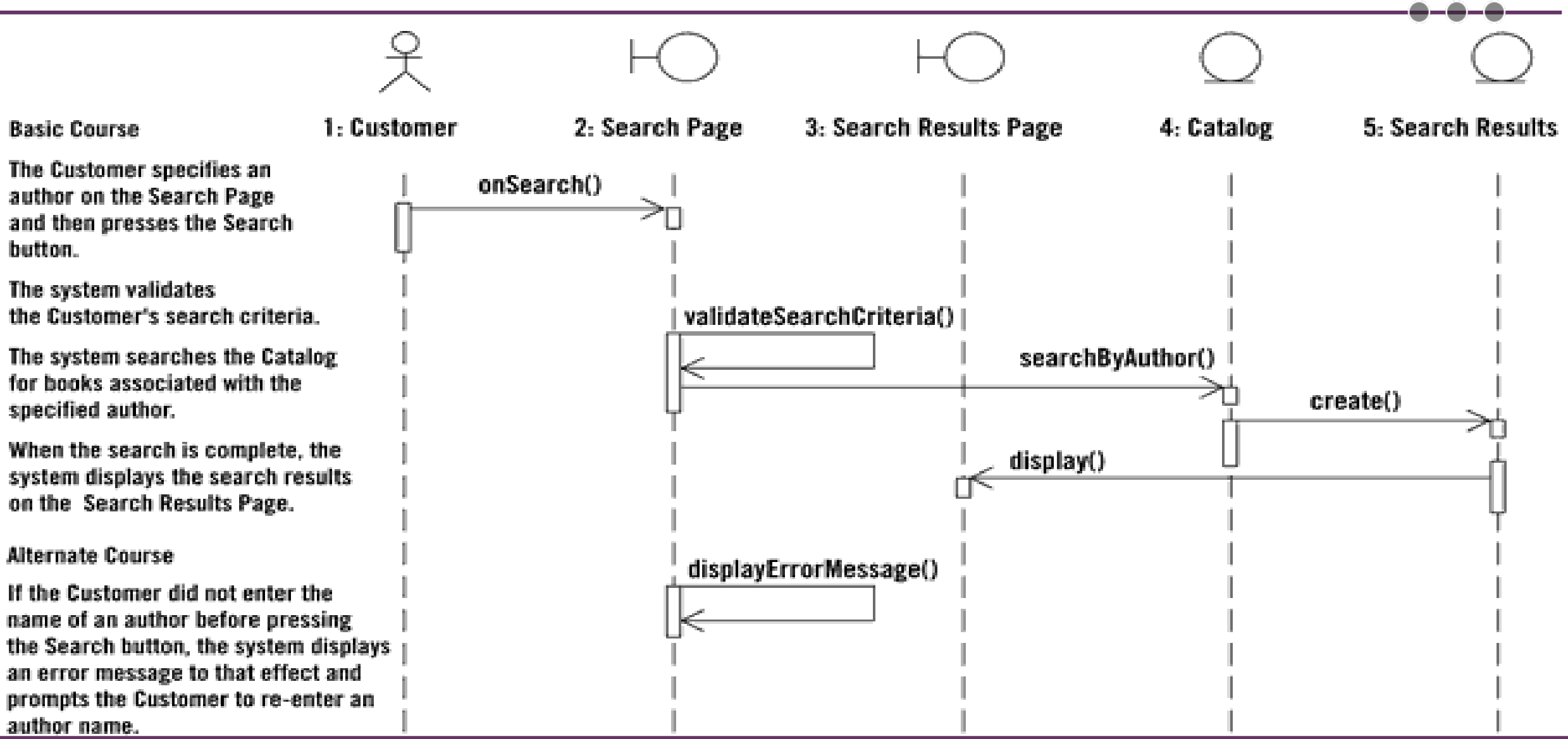
- The thick bar represents its activation. 生命线上的矩形框表示对象的激活期
  - i.e. when it is doing something 即正在执行某项操作

Key parts of a SD

- **participant**: an object or entity that acts in the sequence diagram
  - sequence diagram starts with an unattached "found message" arrow
- **message**: communication between participant objects
- the axes in a sequence diagram:
  - **horizontal**: which object/participant is acting
  - **vertical**: time (down -> forward in time)

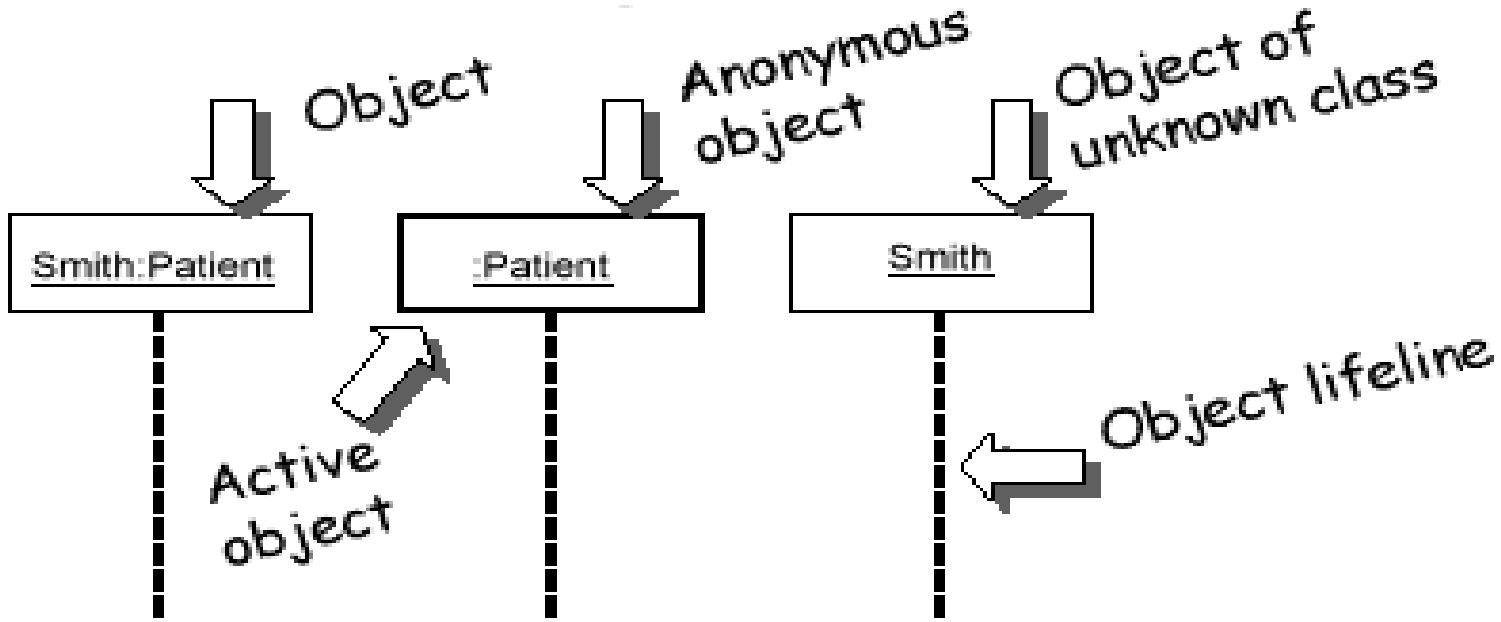


Sequence diagram from use case



Representing objects

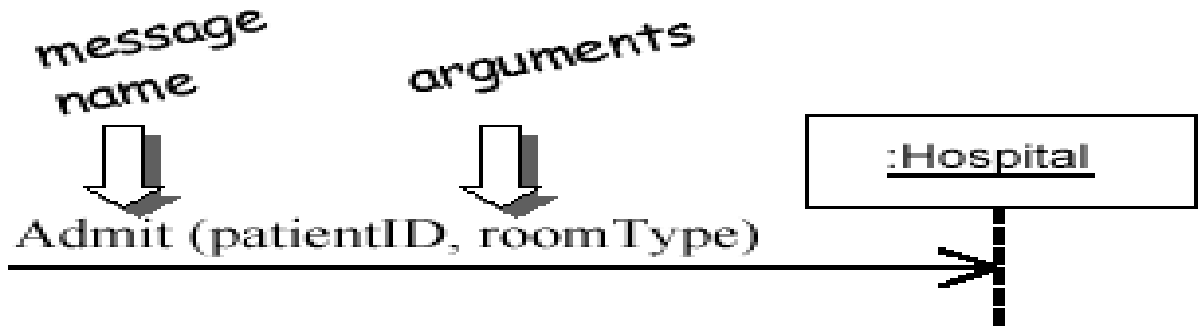
- squares with object type, optionally preceded by object name and colon
  - write object's name if it clarifies the diagram
  - object's "life line" represented by dashed vert. line



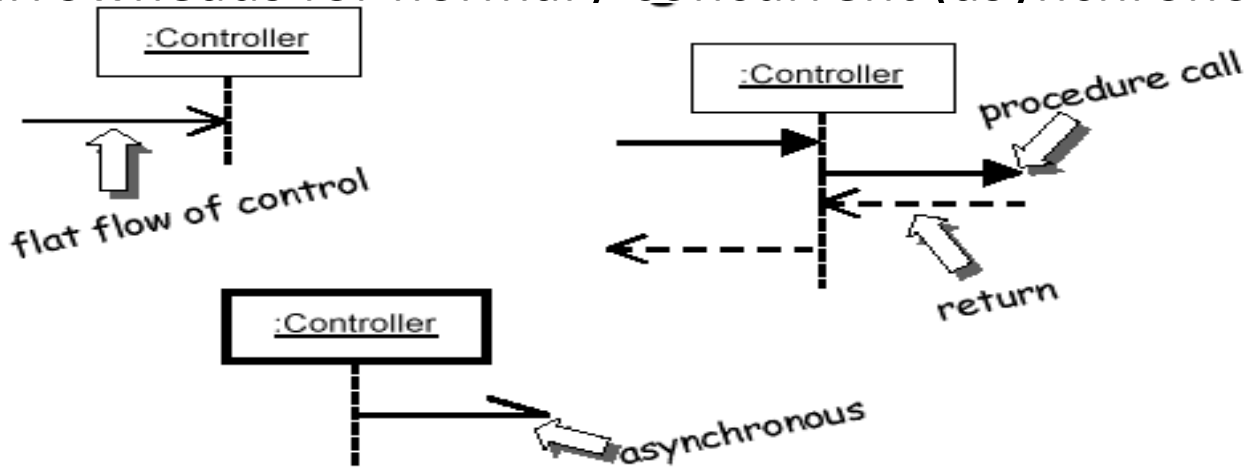
Name syntax: <objectname>:<classname>

Messages between objects

- message (method call) indicated by horizontal arrow to other object
  - write message name and arguments above arrow

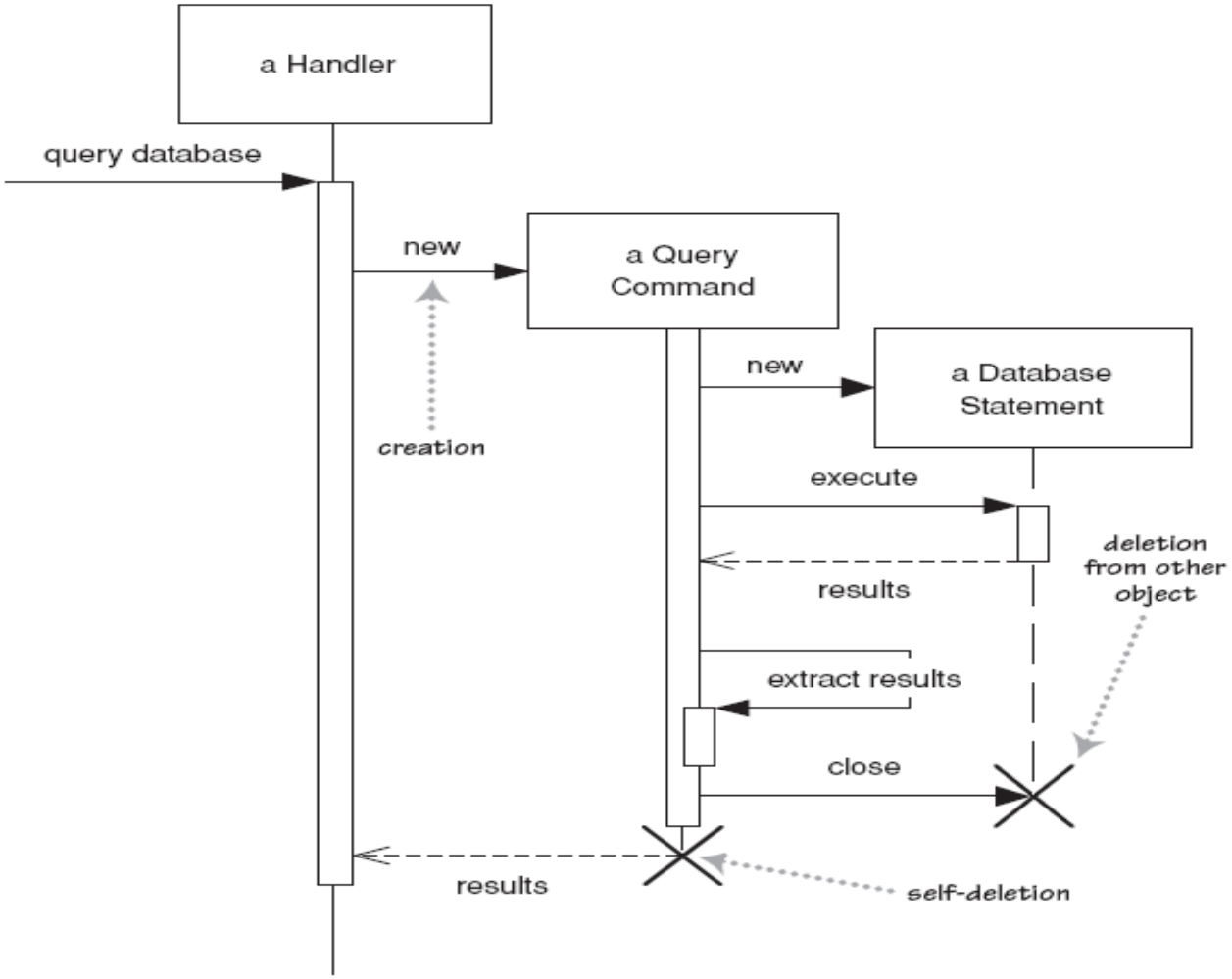


  - dashed arrow back indicates return
  - different arrowheads for normal / concurrent (asynchronous) methods



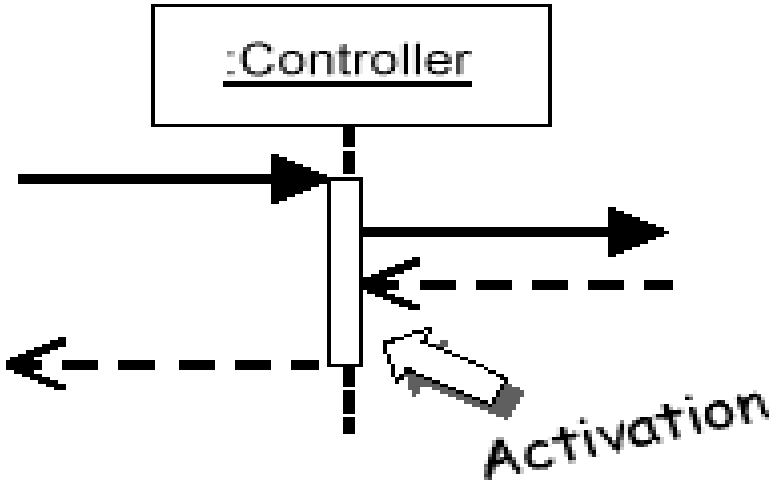
Lifetime of objects

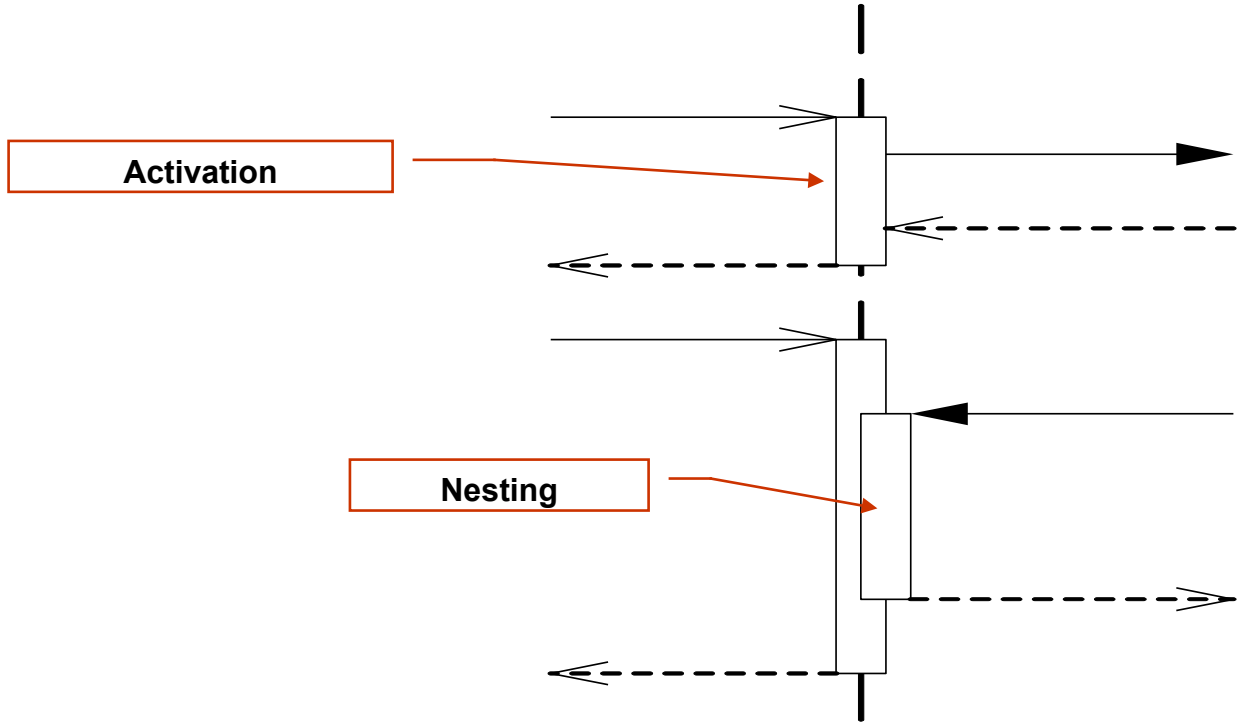
- creation: arrow with 'new' written above it
  - notice that an object created after the start of the scenario appears lower than the others
- deletion: an X at bottom of object's lifeline
  - Java doesn't explicitly delete objects; they fall out of scope and are garbage-collected



Indicating method calls

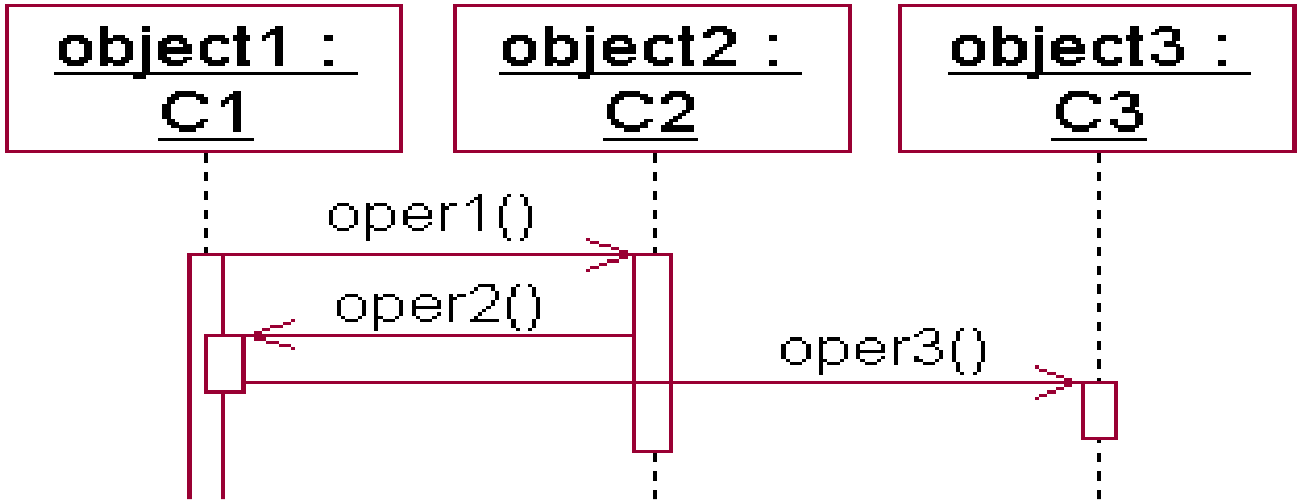
- **activation**: thick box over object's life line; drawn when object's method is on the stack
  - either that object is running its code, or it is on the stack waiting for another object's method to finish
  - nest to indicate recursion





focus of control 的嵌套

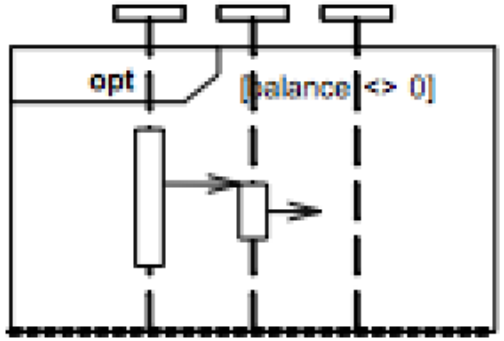
- 嵌套的 FOC 可以更精确地说明消息的开始和结束位置。
- 图例：

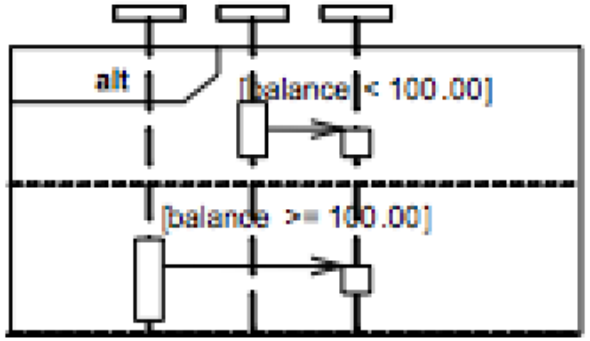


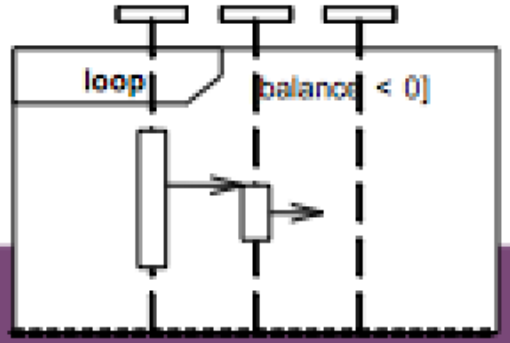
activation(激活期) 表示对象执行一个动作的期间，也即对象激活的时间段。
An activation represents the period during which an object performs an operation either directly or through a subordinate operation.
FOC 和 **activation** 是同一个概念。

Indicating selection and loops

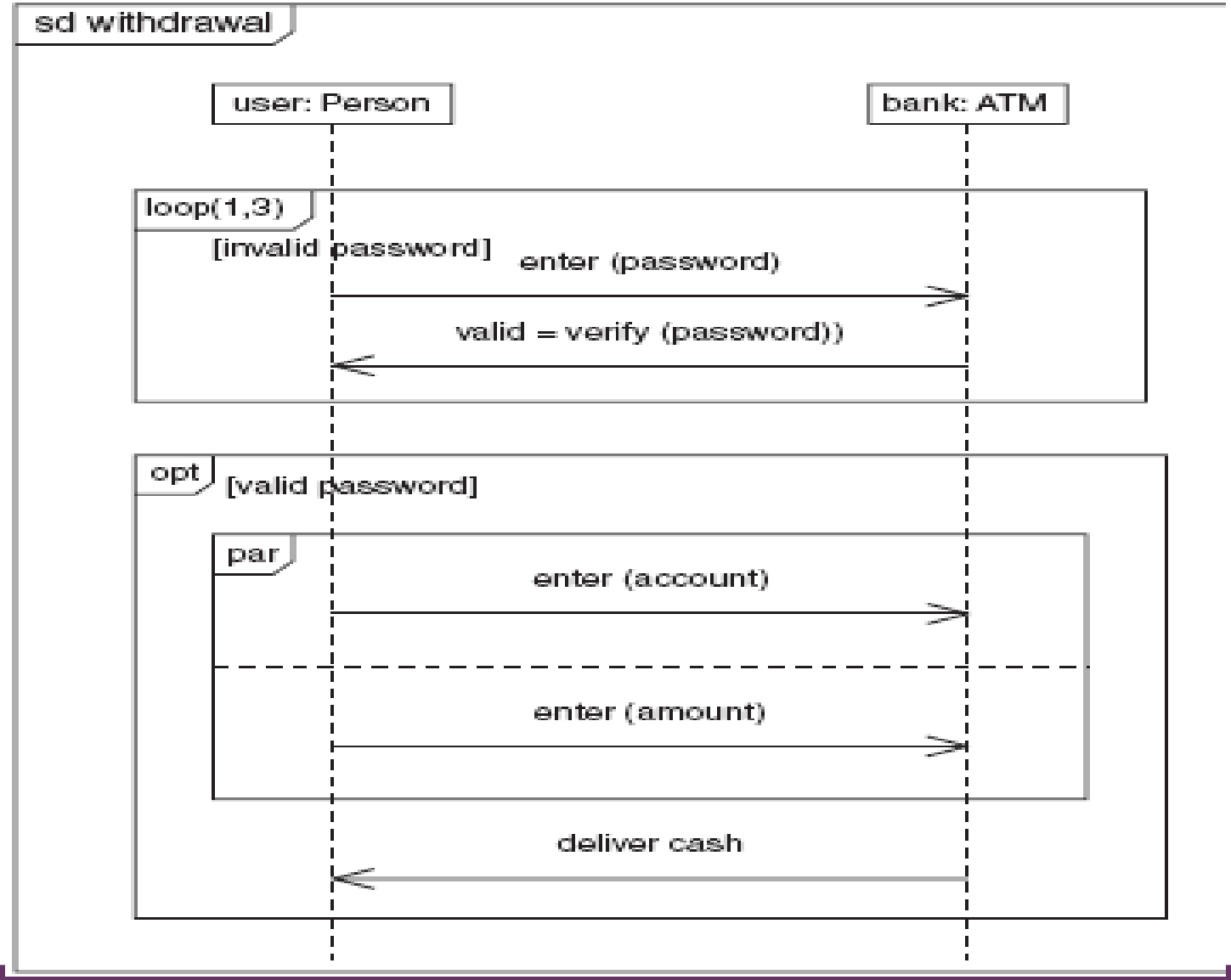
- frame: box around part of a sequence diagram to indicate selection or loop
  - if -> (opt) [condition]
  - if/else -> (alt) [condition], separated by horiz. dashed line
  - loop -> (loop) [condition or items to loop over]

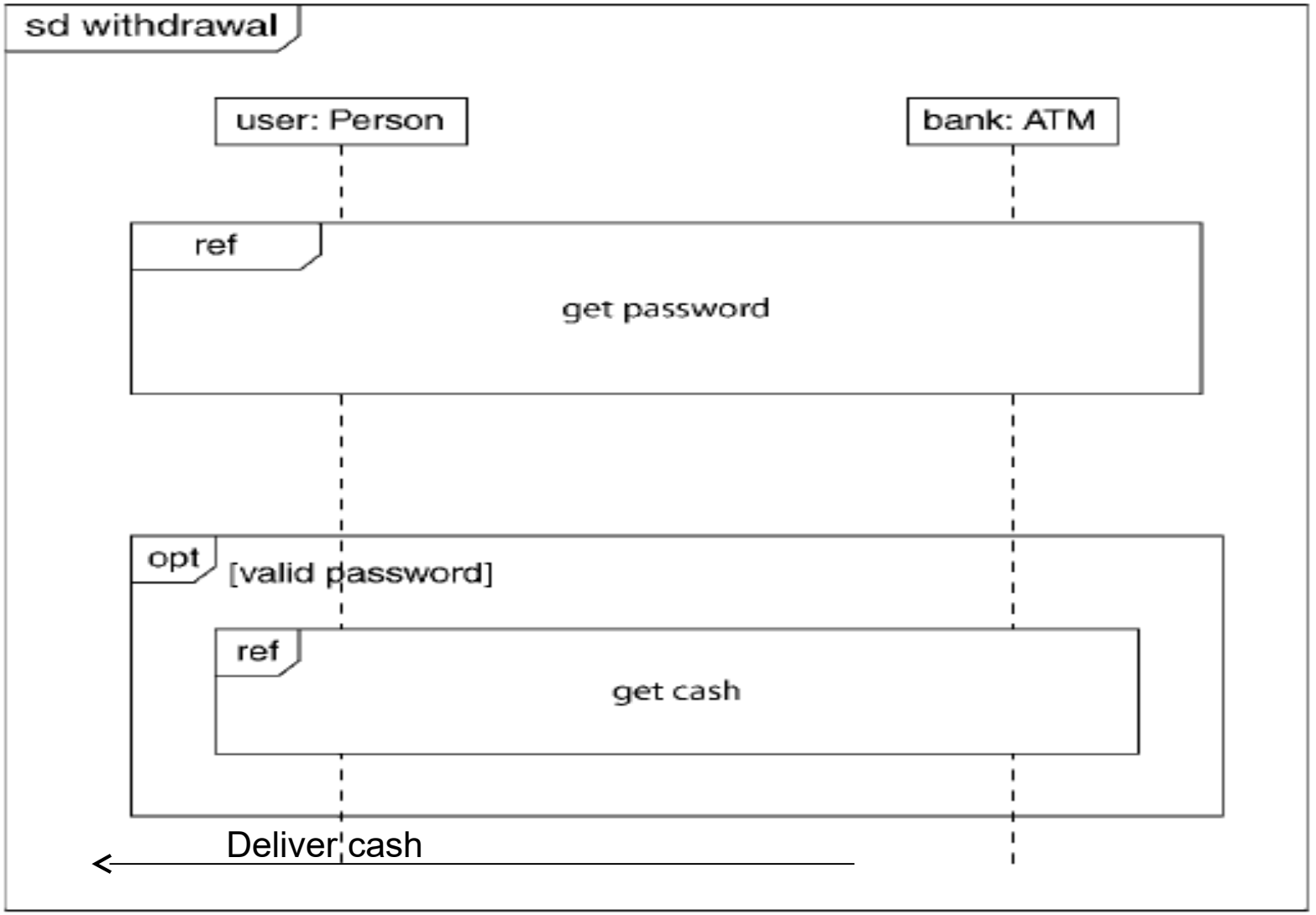


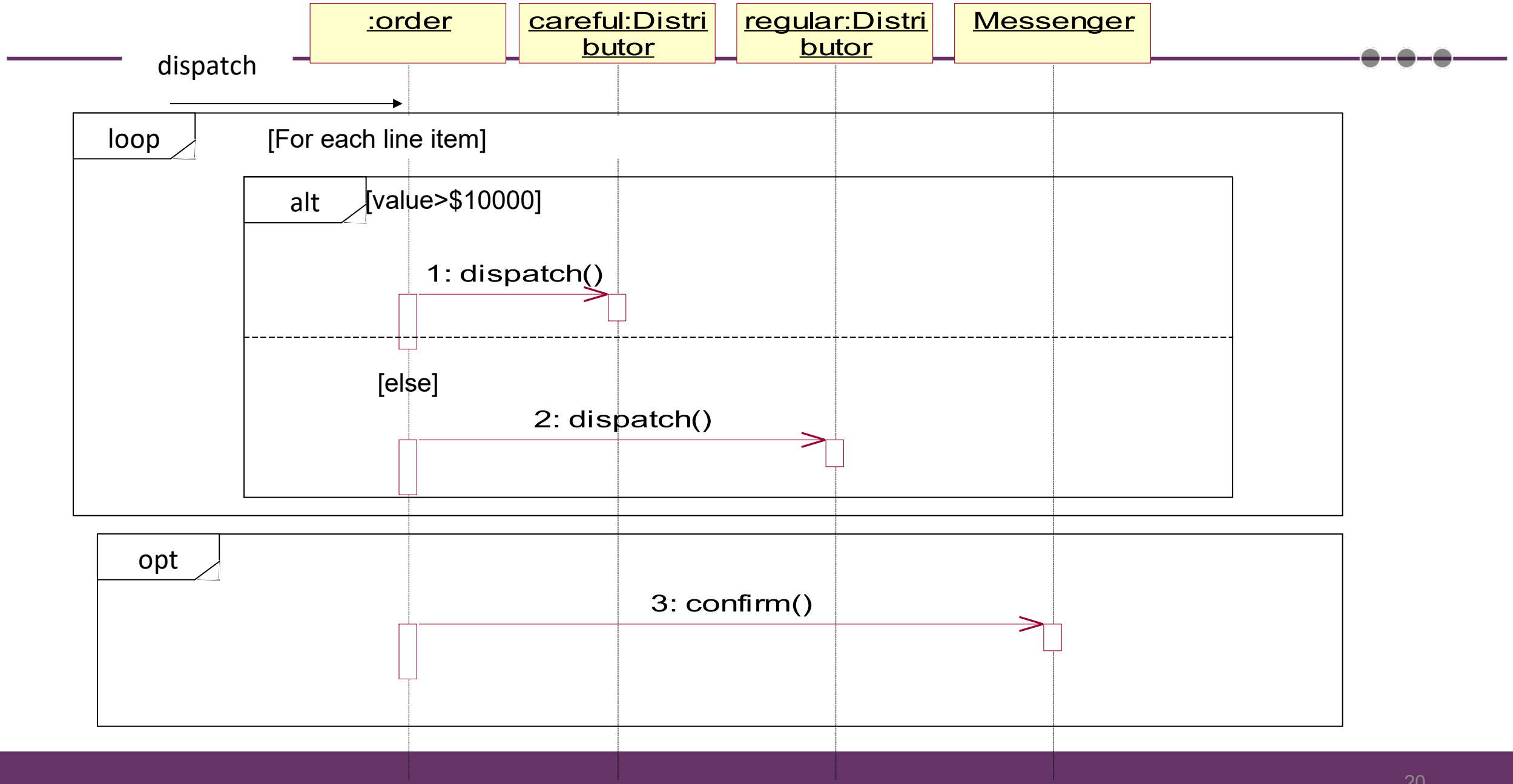




Another Example

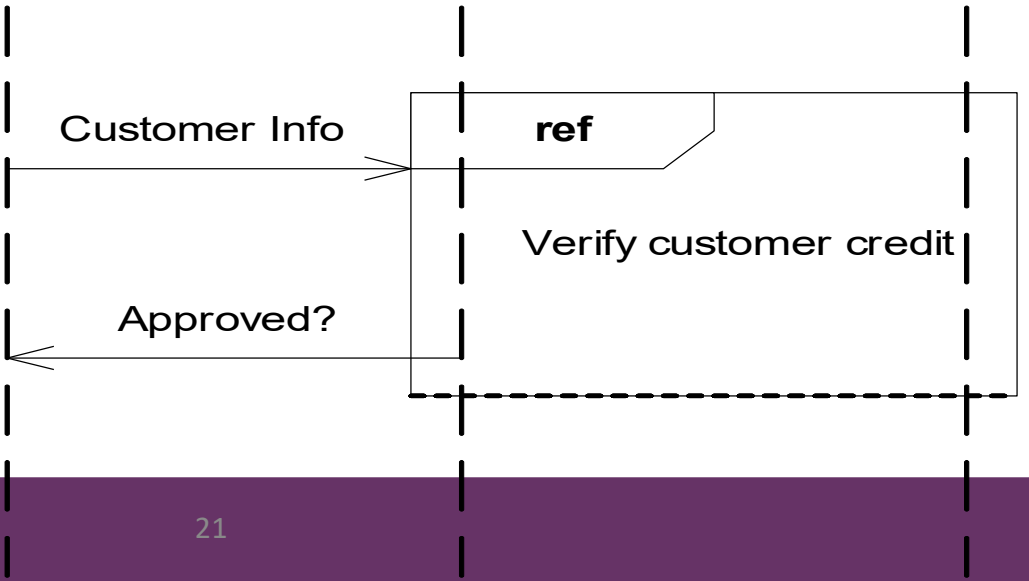


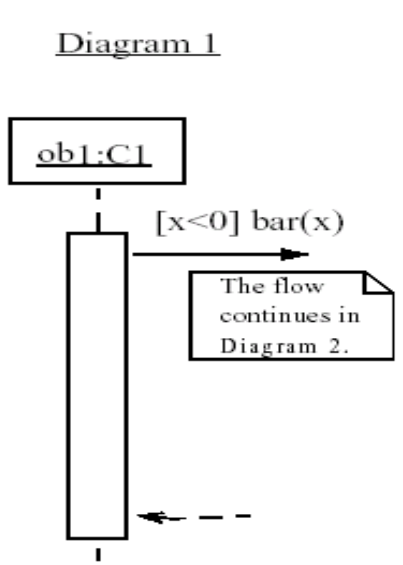

sd withdrawal
user: Person
bank: ATM
ref
get password
opt
[valid password]
ref
get cash
Deliver cash

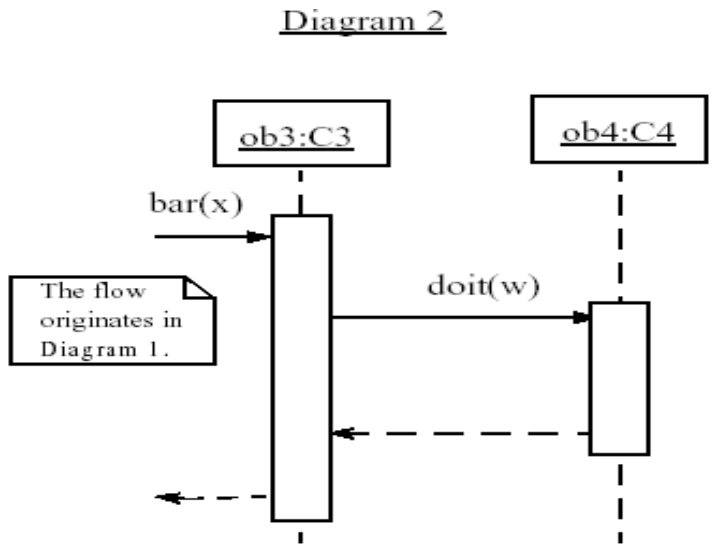

:order
careful:Distributor
regular:Distributor
Messenger
dispatch
loop
[For each line item]
alt
[value>$10000]
1: dispatch()
[else]
2: dispatch()
opt
3: confirm()

linking sequence diagrams

- if one sequence diagram is too large or refers to another diagram, indicate it with either:
  - an unfinished arrow and comment
  - a "ref" frame that names the other diagram
  - when would this occur in our system?



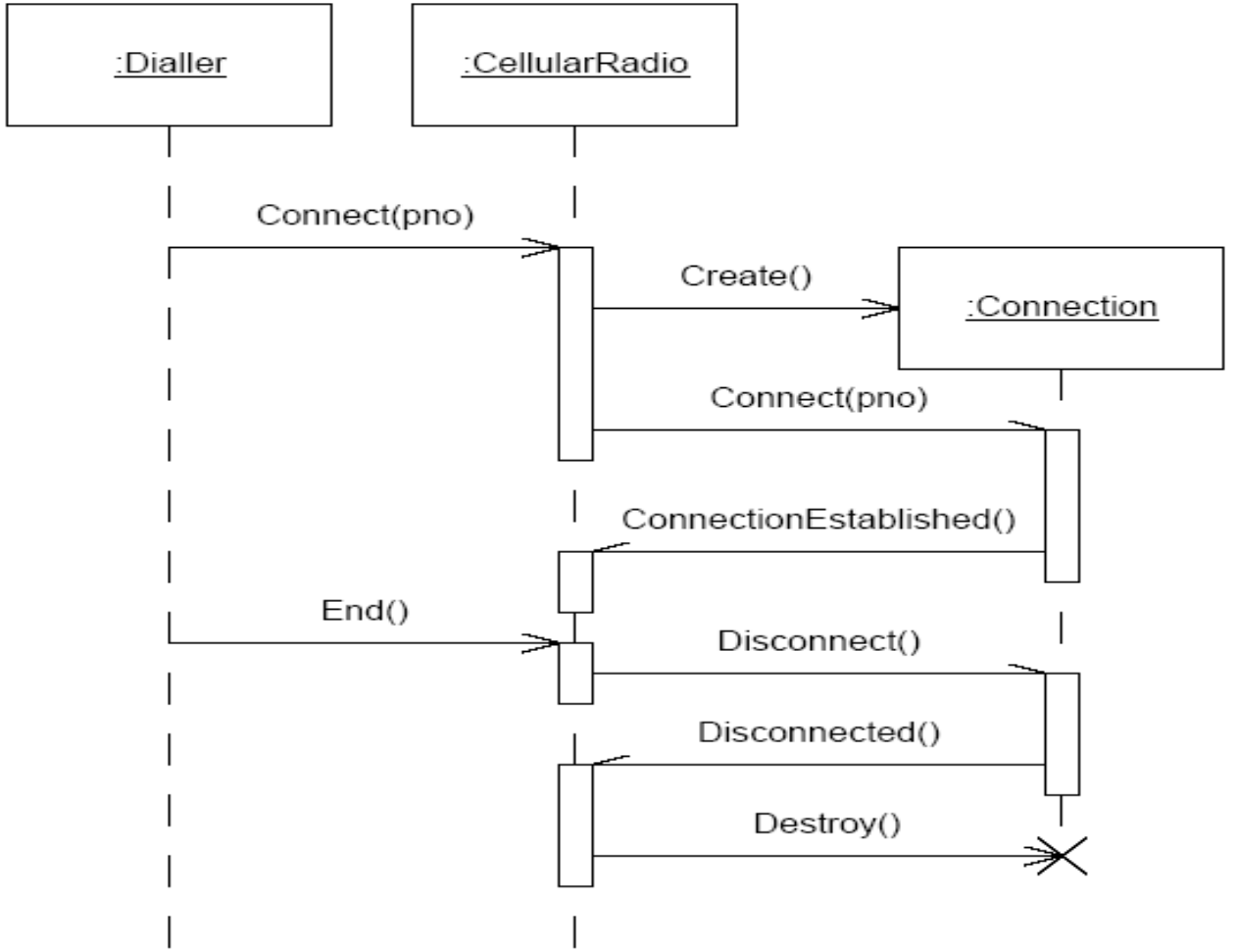




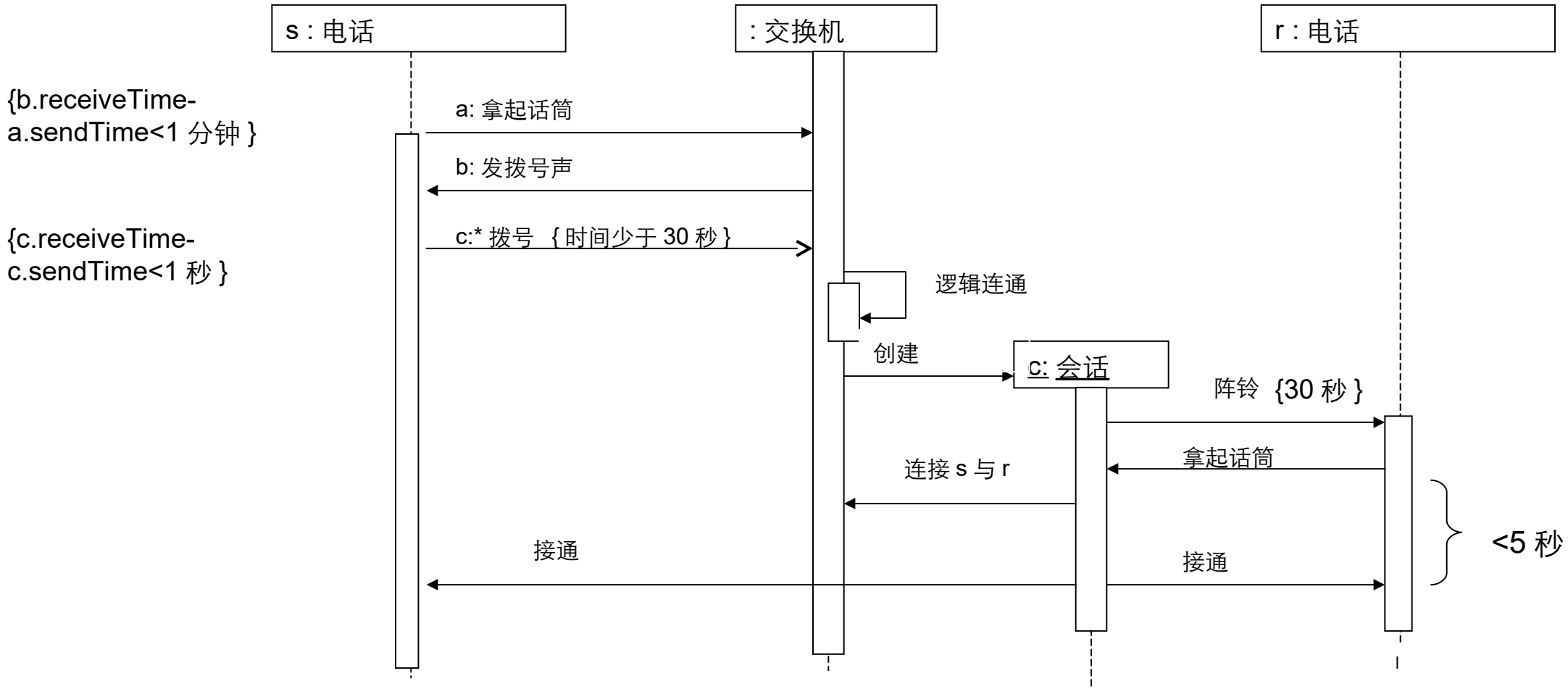
Flawed sequence diagram 1

- What's wrong with this sequence diagram?

(Look at the UML syntax and the viability of the scenario.)



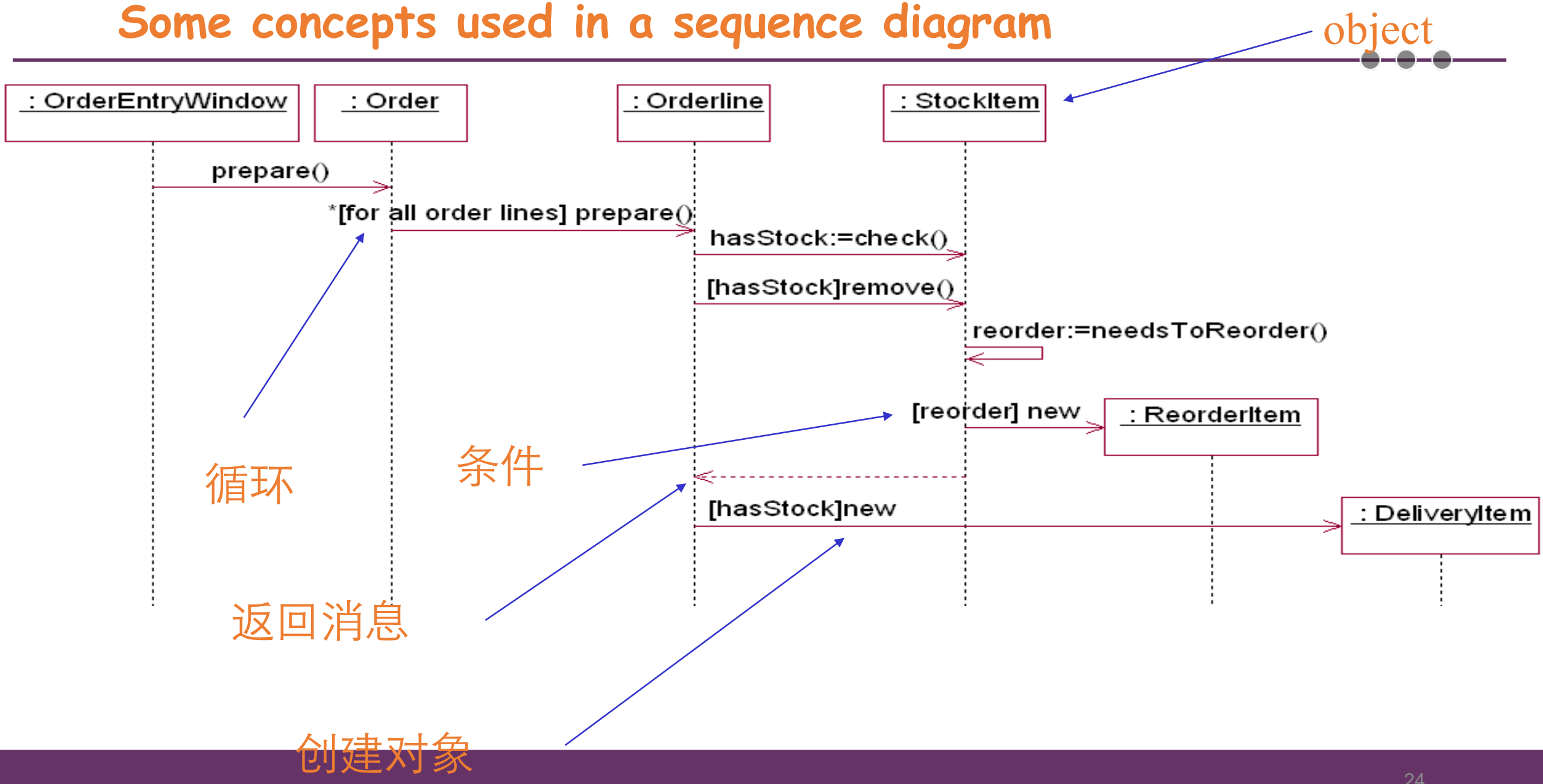
Possible right answer



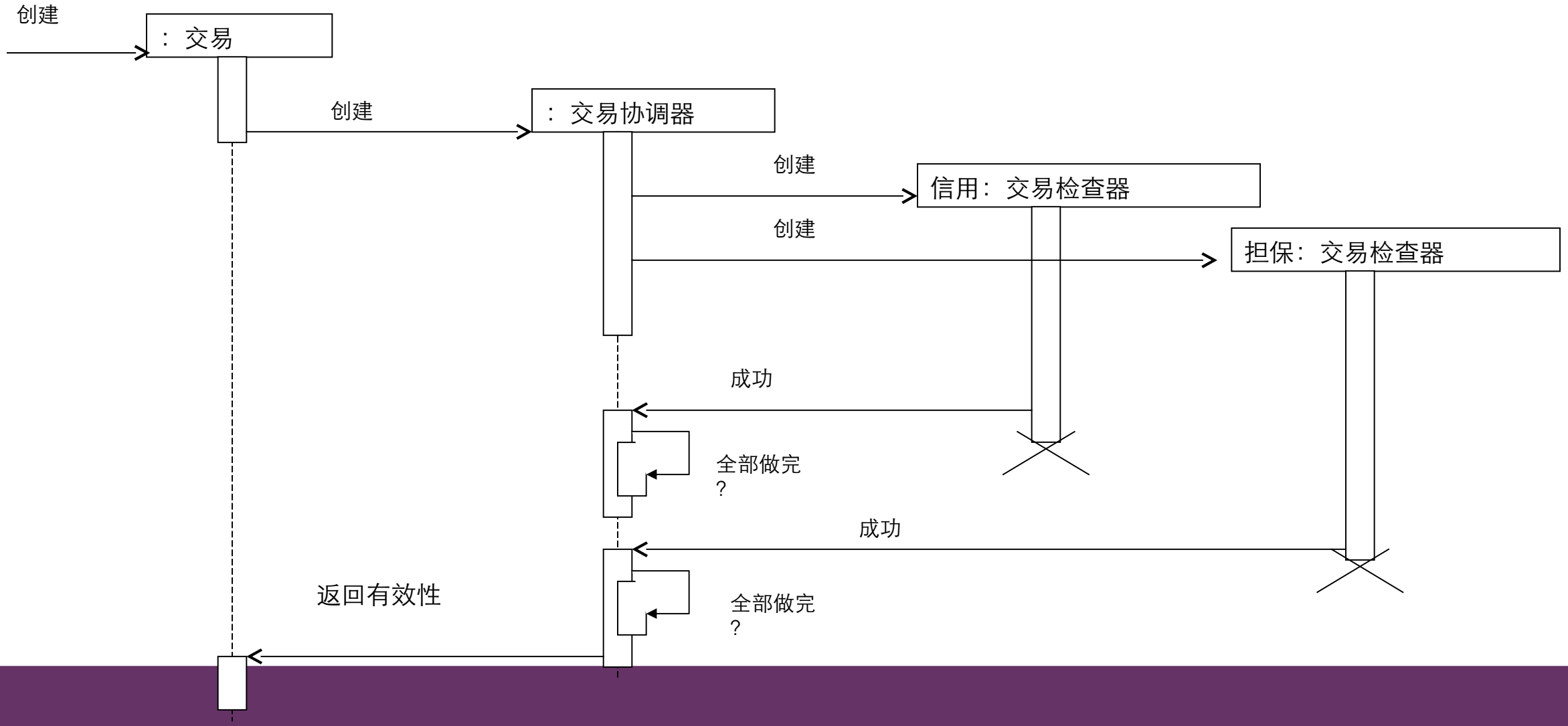
问题：时间超过 30 秒的情况没说明

会话对象没有说明计费等情况

Some concepts used in a sequence diagram

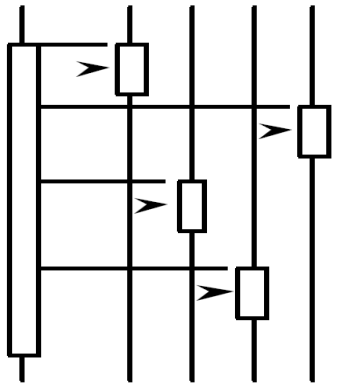


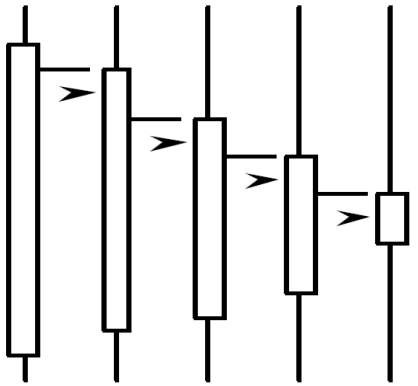
例题 银行系统的交易验证



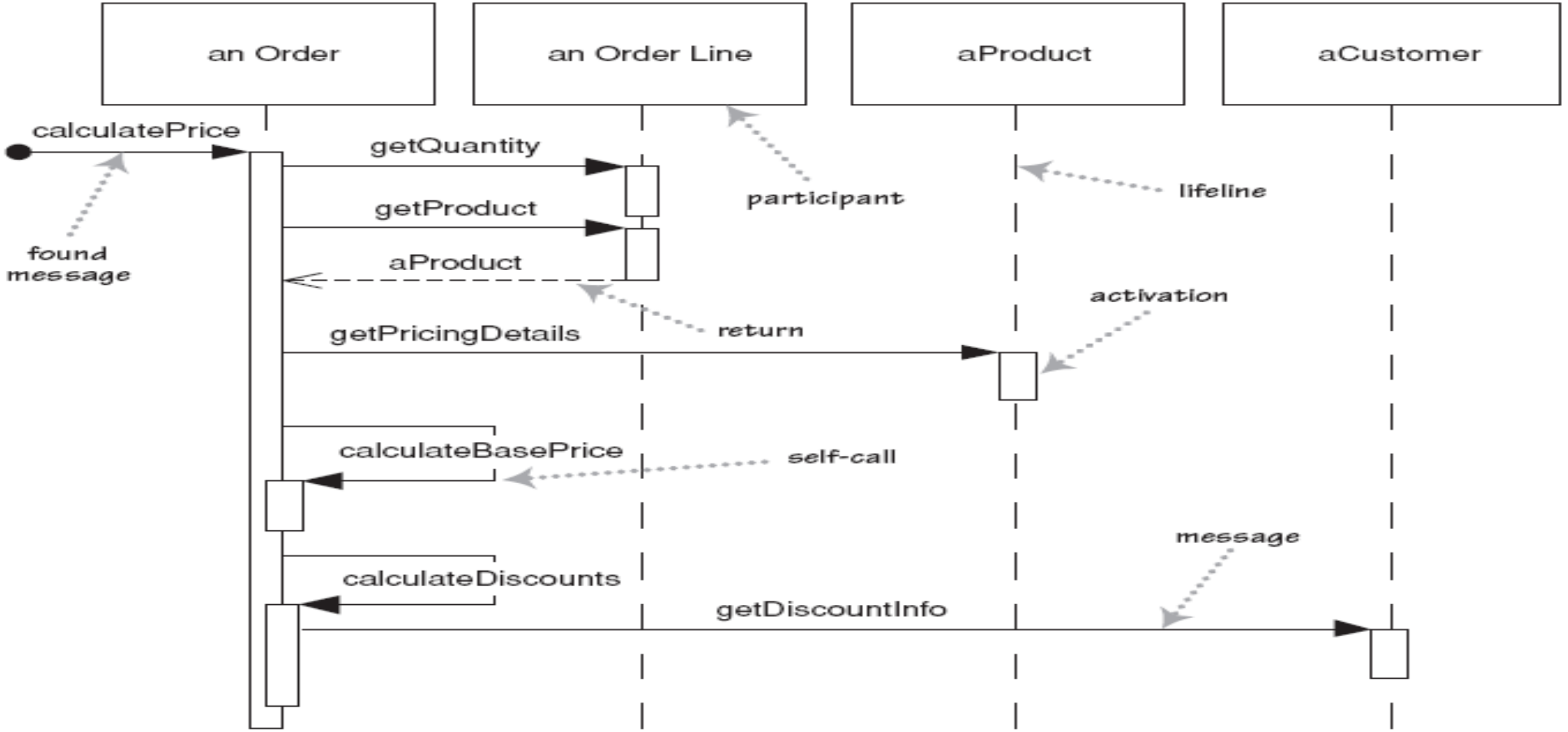
(De)centralized system control

- What can you say about the control flow of each of the following systems?
  - centralized?
  - distributed?

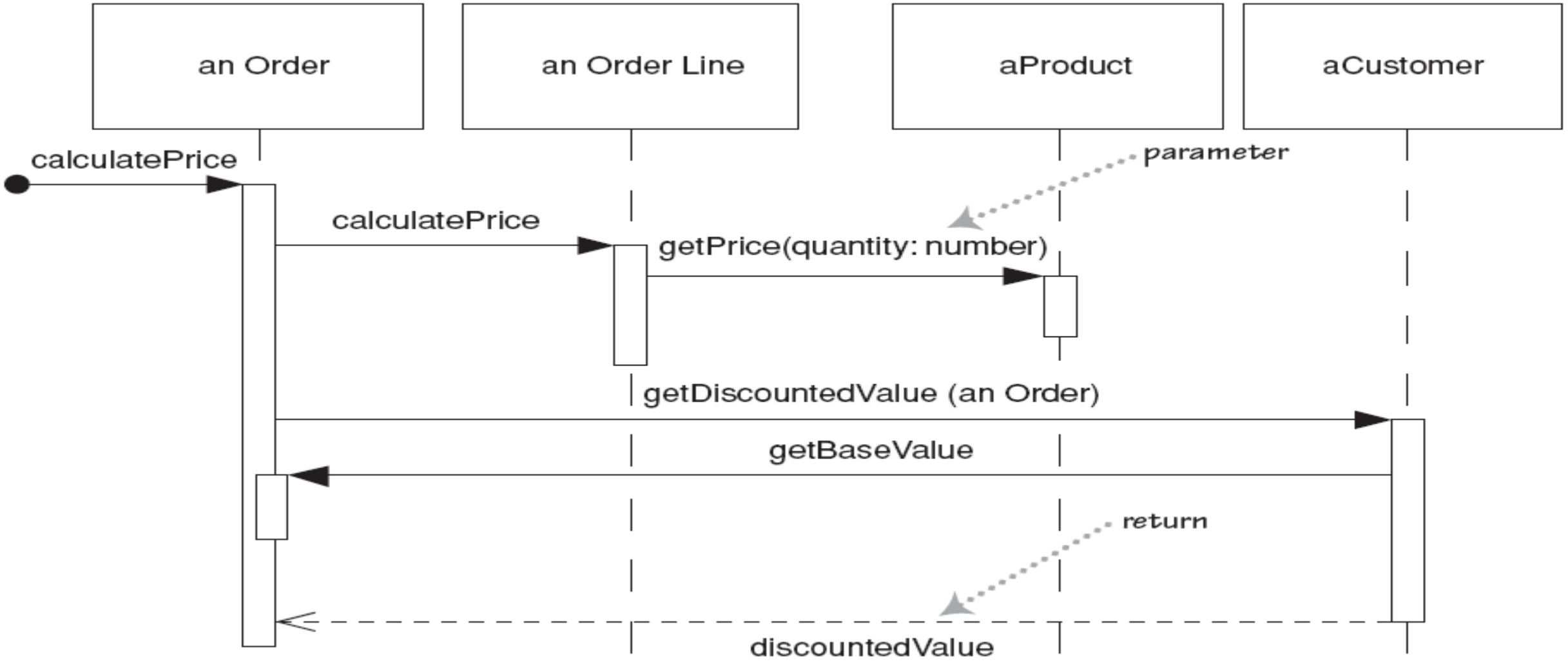




Example 1: A sequence diagram for centralized control
集中控制的计价系统顺序图



Example 2: A sequence diagram for distributed control
分布控制的计价系统顺序图



Why not just code it?

- Sequence diagrams can be somewhat close to the code level. So why not just code up that algorithm rather than drawing it as a sequence diagram?

- a good sequence diagram is still a bit above the level of the real code (not EVERY line of code is drawn on diagram)
- sequence diagrams are language-agnostic (can be implemented in many different languages
- non-coders can do sequence diagrams
- easier to do sequence diagrams as a team
- can see many objects/classes at a time on same page (visual bandwidth)

Sequence diagram exercise 1

- Let's do a sequence diagram for the following casual use case, *Start New Poker Round* :

  The scenario begins when the player chooses to start a new round in the UI. The UI asks whether any new players want to join the round; if so, the new players are added using the UI.

  All players' hands are emptied into the deck, which is then shuffled. The player left of the dealer supplies an ante bet of the proper amount. Next each player is dealt a hand of two cards from the deck in a round-robin fashion; one card to each player, then the second card.

  If the player left of the dealer doesn't have enough money to ante, he/she is removed from the game, and the next player supplies the ante. If that player also cannot afford the ante, this cycle continues until such a player is found or all players are removed.

Sequence diagram exercise 2

- Let's do a sequence diagram for the following casual use case, *Add Calendar Appointment* :

The scenario begins when the user chooses to add a new appointment in the UI. The UI notices which part of the calendar is active and pops up an Add Appointment window for that date and time.

The user enters the necessary information about the appointment's name, location, start and end times. The UI will prevent the user from entering an appointment that has invalid information, such as an empty name or negative duration. The calendar records the new appointment in the user's list of appointments. Any reminder selected by the user is added to the list of reminders.

If the user already has an appointment at that time, the user is shown a warning message and asked to choose an available time or replace the previous appointment. If the user enters an appointment with the same name and duration as an existing group meeting, the calendar asks the user whether he/she intended to join that group meeting instead. If so, the user is added to that group meeting's list of participants.

例：一些消息的例子

2: display (x, y)	简单消息
1.3.1: p:=find(specs)	嵌套消息，消息带返回值
4 [x<0] : invert (x, color)	条件消息
3.1*: update ( )	循环消息
A3,B4/ C2: copy(a,b)	线程间同步

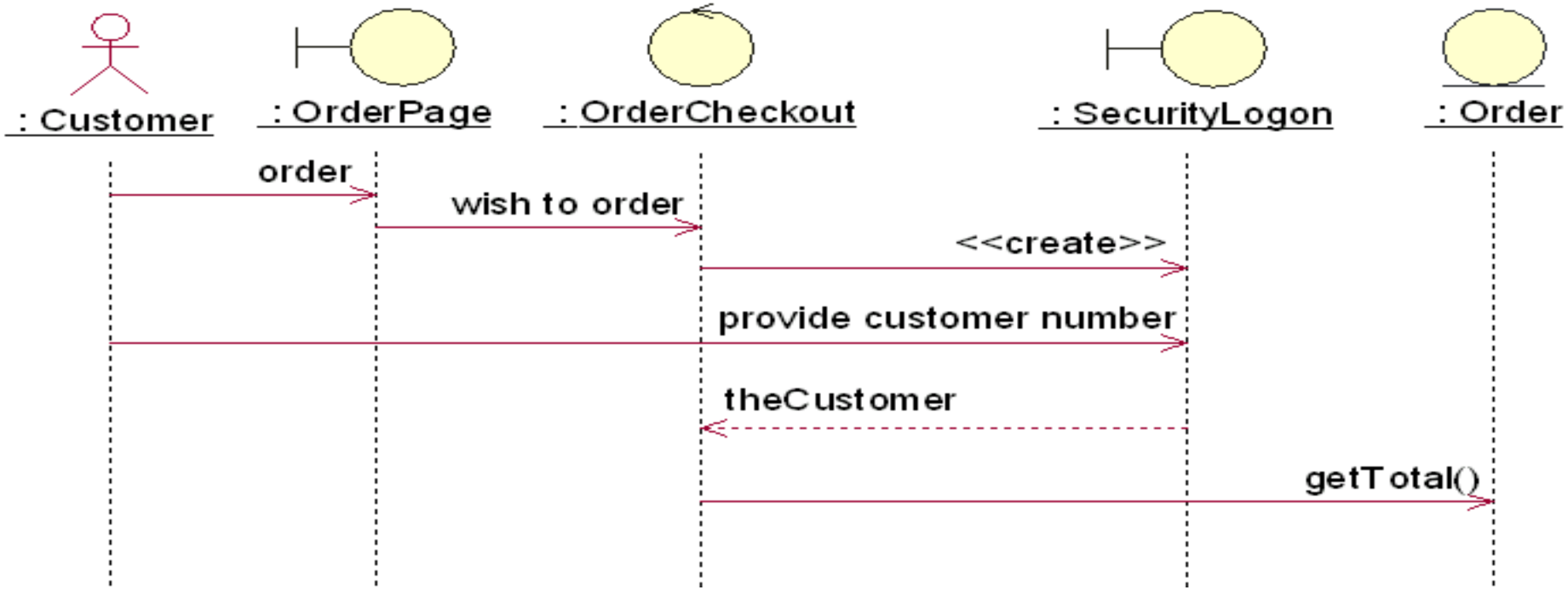
顺序图建模风格

- 建模风格 1： 把注意力集中于关键的交互。
  - 创建模型时要把注意力集中于系统的关键方面，而不要包括无关的细节。
  - 例如：如果顺序图是用于描述业务逻辑的，就没必要包括对象和数据库之间的详细交互，像 save() 和 delete() 这样的消息可能就足够了。或者简单地假定持久性 (persistence) 会适当地被处理，而不用考虑持久性方面的细节问题。

建模风格 2：对于参数，优先考虑使用参数名而不是参数类型。

- 例如，消息 addDeposit(amount, target) 比 addDeposit(Currency, Account) 传递了更多的信息。
- 在消息中只使用类型信息不能传递足够的信息。
- 参数的类型信息用 UML 类图捕获更好。
- 建模风格只是建议，不是规定。如果只是需要表示“有某些东西要传递”这个事实，而不需要提供更多的细节，则可以指明参数的类型作为占位符。

- 建模风格 3： 不要对明显的返回值建模。
  - 例：创建安全登录对象的行为会导致生成一个顾客对象，这个是不明显的；而向订单对象发送请求其总数的消息，其返回值是显然的。



- 建模风格 4： 可以把返回值建模为方法调用的一部分。

顺序图常见问题分析

- **顺序图中消息的循环发送**
  - 在消息名字前加循环条件

例：

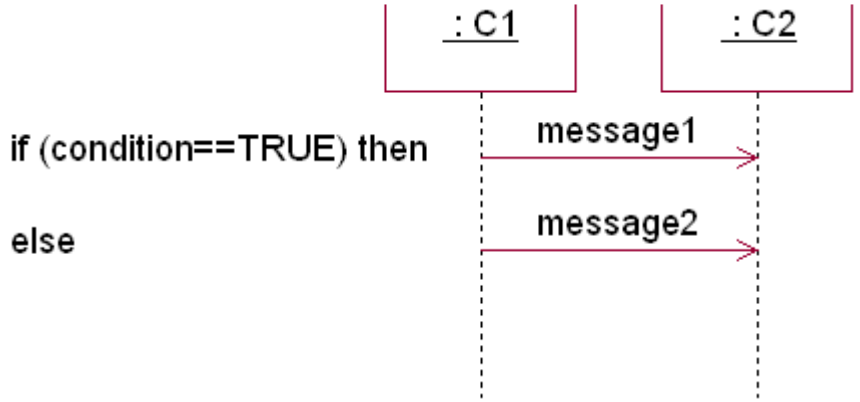
1.1 *[for all order lines]: message1()

2.1 *[i:=1..n]: message2()

- **顺序图中消息的条件发送**

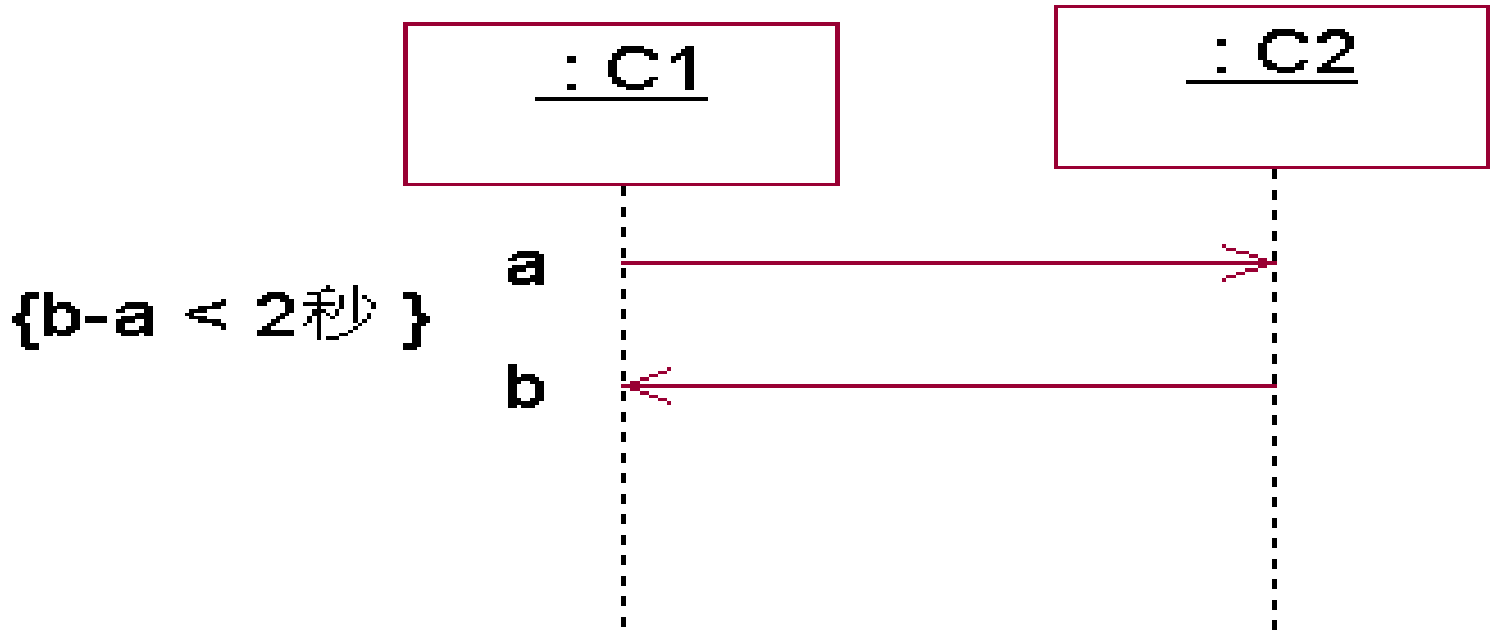
1. 在消息名字前加条件子句;
2. 使用文字说明;
3. 分成多个 Sequence Diagram 。

例:



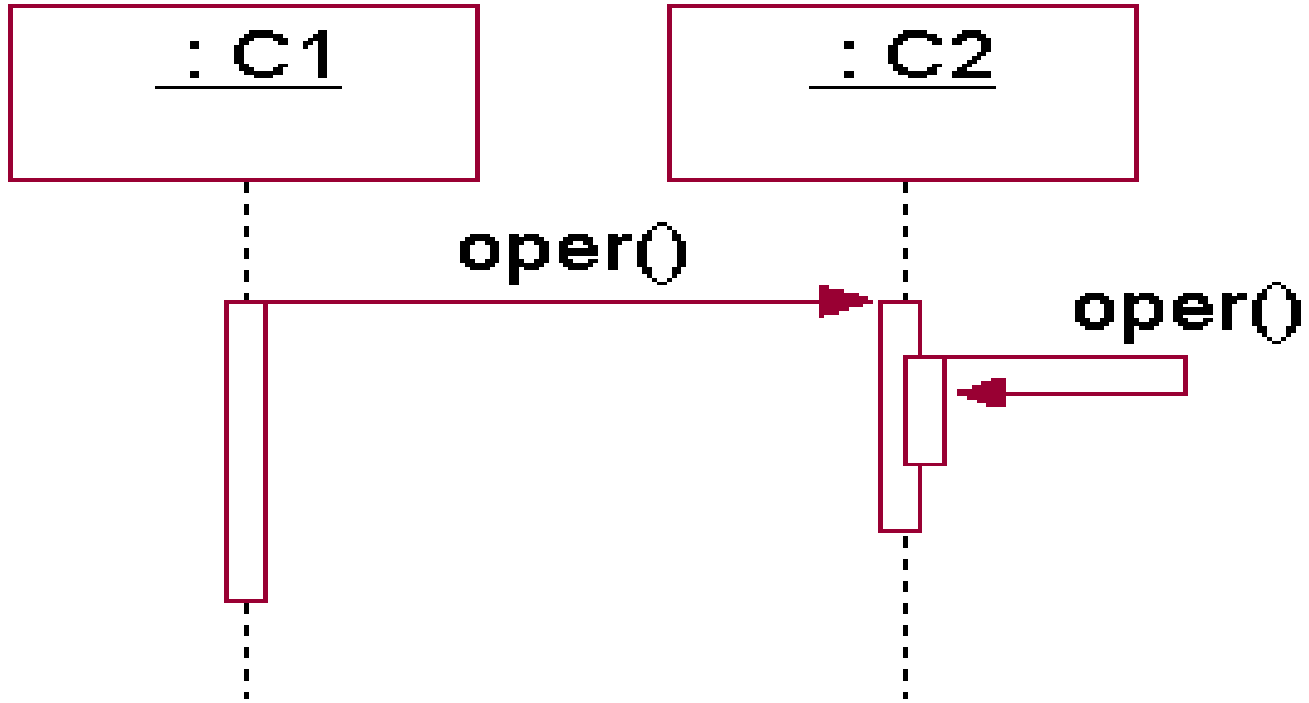
- **顺序图中时间约束的表示**
  - 用 constraint(约束) 来表示。

例 :

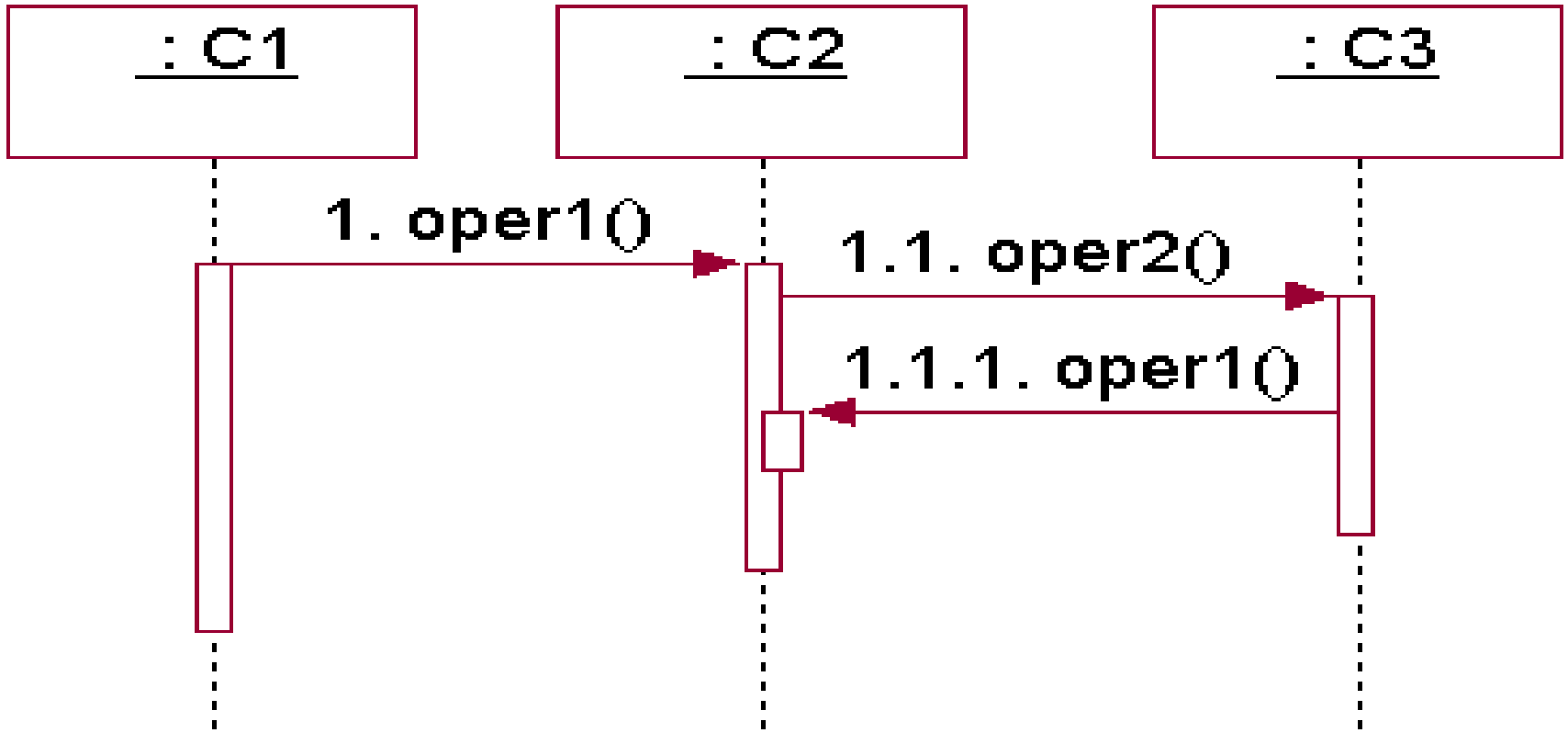


- **顺序图中递归的表示**
  - 利用嵌套的 FOC 表示

例 1. 单个对象自身的递归。



例 2. 多个对象间相互递归调用的表示。



Collaboration/Communication (协作 / 通信图)

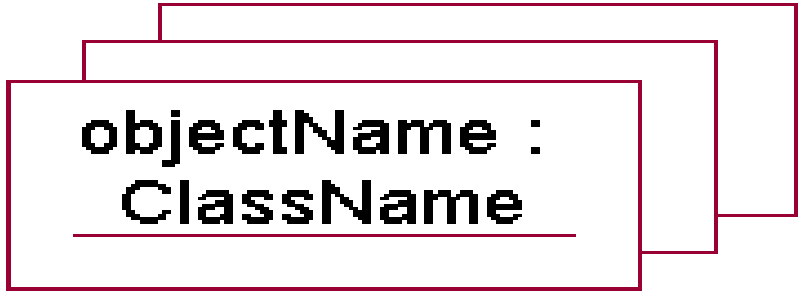
- 协作图包含一组对象和链 (link)，用于描述系统的行为是如何由系统的成员协作实现的。

- A **collaboration diagram** is a diagram that shows interactions organized around roles—that is, slots for instances and their links within a **collaboration**.

- 协作图中的一些主要元素：
  - **Object**(包括 actor 实例，多对象，主动对象)
  - **Message**
  - **Link**(链)

multiobject (协作图中的多对象)

- A **multiobject** is used within a collaboration to show operations that address the entire set of objects as a unit rather than a single object in it.

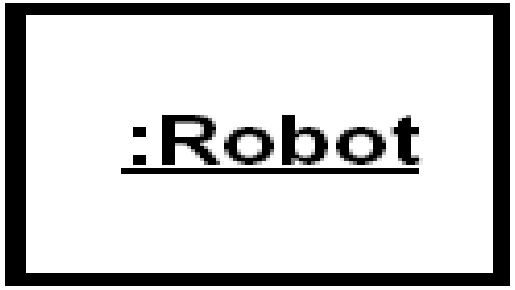


Active Object

- **主动对象**是一组属性和一组方法的封装体，其中至少有一个方法不需要接收消息就能主动执行 (称作主动方法) 。
- An object that owns a thread of control and can initiate control activity 。
- 除含有主动服务外，主动对象的其它方面与被动对象没有什么不同。
- 目前并无商品化的 OOPL 能支持主动对象的概念，需要程序员在现有的语言条件下设法把它实现成一个主动成分。

主动对象的表示

(1) UML 中的表示： 加粗的边框



(2) Rose 中的表示

link (协作图中的链)

- 协作图中用**链 (link)** 来连接对象，而消息显示在链的旁边。
  - 链是 **association(关联)** 的 **instance(实例)**
- 一个链上可以有多个消息。
- 链的两端**不能**有**多重性 (multiplicity)** 标记。

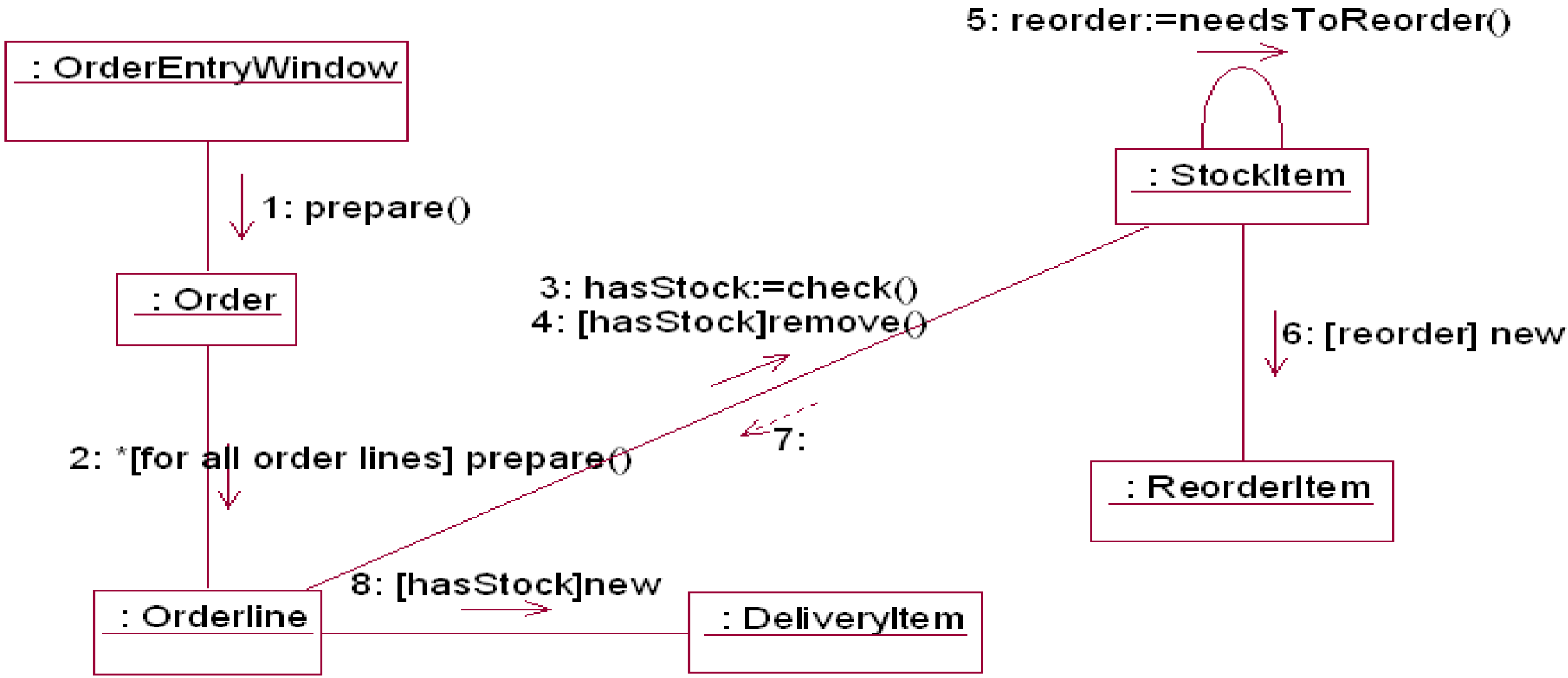
协作图中重复和条件分支的表示

- 和顺序图中的表示方法类似
  - 用 *[i:=1..n] 或 * 表示要重复发送的消息。
  - 用类似 [x > 0] 的条件语句表示消息的分支。
  - UML 并没有规定 [] 中的表达式的格式，因此可以采用伪代码的格式或某种程序设计语言的语法格式。

建立 collaboration 图的步骤

1. 确定交互过程的上下文 (context)；
2. 识别参与交互过程的对象；
3. 如果需要，为每个对象设置初始特性；
4. 确定对象之间的链 (link)，以及沿着链的消息；
5. 从引发这个交互过程的初始消息开始，将随后的每个消息附到相应的链上；
6. 如果需要表示消息的嵌套，则用 Dewey 十进制表示法；
7. 如果需要说明时间约束，则在消息旁边加上约束说明；
8. 如果需要，可以为每个消息附上前置条件和后置条件。

Collaboration 图的例子：由顺序图转换来的协作图



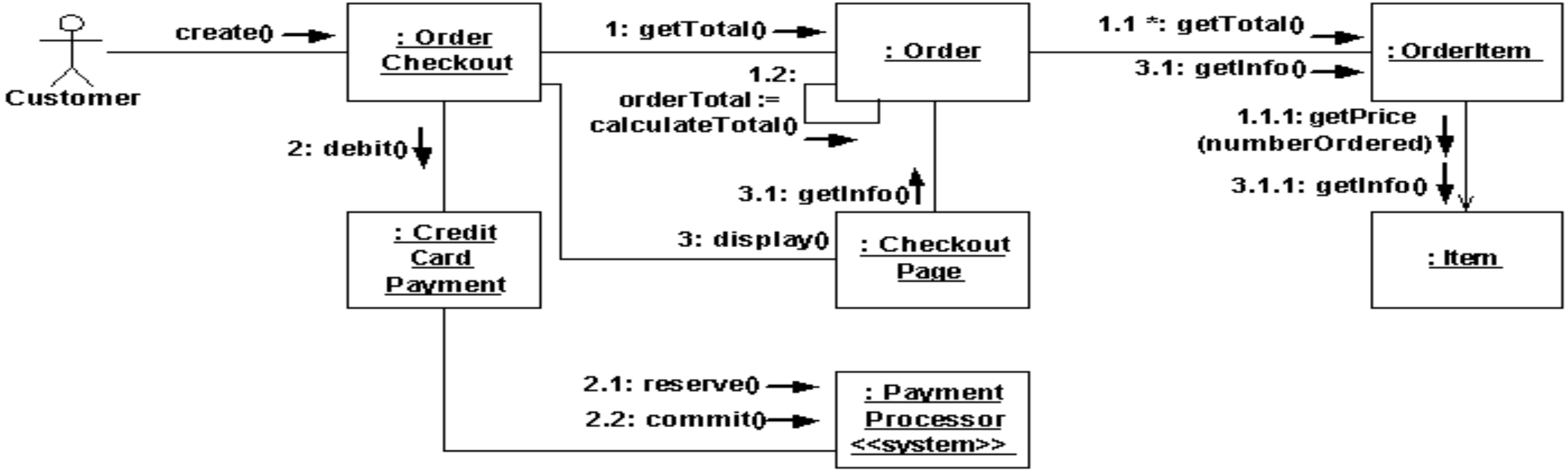
协作图建模风格

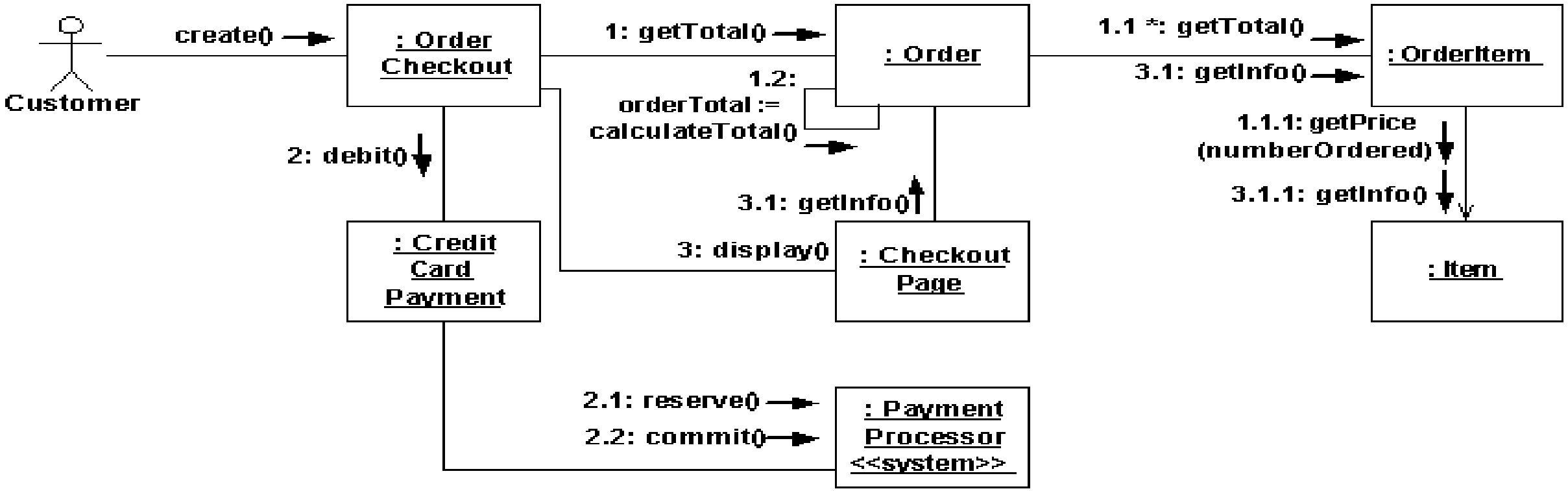
类似于顺序图：

- 把注意力集中于关键的交互。
- 对于参数，优先考虑使用参数名而不是参数类型。
- 不要对明显的返回值建模。
- 可以把返回值建模为方法调用的一部分。

其他的建模风格:

- 建模风格 5： 为每个消息画出箭头。
  - 便于可视化地确定流向一个给定对象的消息数量，以此判断该对象所涉及的潜在耦合情况，这通常是在对设计进行重构时要考虑的一个重要因素。
  - 例子:




Customer
create()
: Order Checkout
1: getTotal()
: Order
1.1 *: getTotal()
: OrderItem
3.1: getInfo()
1.2: orderTotal := calculateTotal()
1.1.1: getPrice (numberOrdered)
3.1.1: getInfo()
2: debit()
3.1: getInfo()
: Credit Card Payment
3: display()
: Checkout Page
: Item
2.1: reserve()
2.2: commit()
: Payment Processor <<system>>

- 建模风格 6： 注明导航性 (Navigability) 时要慎重。
  - 可以在协作图中对导航性建模，但这不是很常见，因为这太容易和消息流混淆。
  - 用 UML 类图来描绘导航性更好。
  - 例子： Slide 49

顺序图和协作图的比较

- 顺序图强调消息的时间顺序，协作图强调参加交互的对象的组织，两者可以相互转换。
- 顺序图不同于协作图的两个特征：
  - 顺序图有对象生命线
  - 顺序图有控制焦点
- 协作图不同于顺序图的两个特征：
  - 协作图有路径
  - 协作图必须有消息顺序号

- 和协作图相比，顺序图：
  - Take more space
  - Easier to follow algorithms
  - Easier to depict lifetimes
  - Easier to show multithreading in one object
  - More difficult to visualizing multiple concurrent flows of control
  - Do not show association links
- 顺序图可以表示某些协作图无法表示的信息；同样，协作图也可以表示某些顺序图无法表示的信息。

工具支持

1. 模型以动画的方式执行；
2. 正向工程和逆向工程；
3. Rose 中的顺序图和协作图
   - 不支持 UML1.5 中的全部符号；
   - 顺序图和协作图相互转换时不转换 text box 元素。

常见问题分析

1. Sequence 图中的对象如何确定？与 Class 图是否是同时生成并交互修改？
2. 交互图中的多态性问题：如果对象具有多态性，发送对象不可能事先知道目标对象属于哪个类，因此在交互图中如何确定目标对象所属的类?

答：多态性属于运行时问题，这个类是目标对象有可能所属超类 (一般类，父类) 。

总结

- 基本概念：交互图，顺序图，协作图， Lifeline (生命线)， Focus of control(控制焦点)， activation(激活期)，消息的类型和语法格式，active object(主动对象)， multiobject(多对象)， Link(链)
- 交互图的应用

思考题

- 如何根据用例图、交互图、类图、活动图、状态图等来生成测试用例?
- 例： Aynur Abdurazik and Jeff Offutt, Using UML Collaboration Diagram for Static Checking and Test Generation.
  - <<UML 2000>> 国际会议论文，
  - 这篇论文给出了一个根据 UML 的协作图 (设计阶段的结果) 对系统进行静态分析和动态执行路径检查的方法。

主要想法:

- 由于协作图包含了对象间所传递的消息及消息的顺序，协作图提供了设计层的数据流和控制流信息。因此，利用协作图可以

  (1) 在代码生成前利用数据流和控制流信息产生测试数据;

  (2) 对协作图本身进行静态检查;

  (3) 对代码进行静态检查。

撰写需求文档

Writing and Reviewing Requirements

清华大学软件学院 刘璘

Software Requirements Specification （SRS）

- Is a contractually binding document stating clearly what the software will do, and when necessary, what it won't do.
- Describes functional requirements in terms of input, outputs, and transformations from input to outputs.
- Describes non-functional requirements that have been negotiated and agreed upon by the stakeholders.
- Is supported by the requirements definition document, containing clear definitions of all terms in the SRS.
- Is an active, living, specification

Styles of SRSs

- Narrative natural language text (typical with marketing)
- From a use-case model:
- A use-case model is expressed as a directed graph that can be traversed. If requirements are placed in the model as use cases, then a traversal can generate a full set of requirements.
- Generated from a requirements database:
- Commercial requirements databases have built-in facilities to generate filtered requirement specifications, e.g., a specification for a particular product from a product-line specification database.
- Generated from a hybrid model: Feature model and a use-case model

The last two are often done with the help of templates

Overview of Methods for producing Different Styles of SRSs

Method	When to Use	Suitable For	Effort	Resources Needed
Narrative Text	Small projects	Informal specifications for small products	Medium	Analyst with good writing skills
Generate from a Use Case Model	Medium to large projects	Formal specifications, product lines	Medium	Analysts with modeling skill, good process standards, modeling tool(s)
Generated from a Requirements Database	Medium to Large projects	Formal specifications, product lines	Light	Requirements database, analyst with DBA skills
Generated from a Hybrid	Small to medium project	Single product definition	Light to medium	Analyst with good writing skills, RE, DBA, or Modeling skills

User' Manuals as SRSs

- Fred Brooks's Mythical Man Month describes using a users' manual as the requirements specification.
- Legend has it that the users' manual for the first Macintosh was written before coding began.
- Several have advocated writing a users' manual as a cost-effective way to kill two birds with one stone, to get both an SRS and a users' manual.
- Using a users' manual as an SRS works well for systems that interact with users, that are interaction driven.
- A good manual describes all scenarios for all use case.
- But…
- A users' manual does not describe
  - Non-functional requirements very well
  - Functional requirements of a system that does not interact with users, i.e., a system that is a function, filter, or translator.

Outline of a Users' Manual

- Introduction
  - Product overview and rationale
  - Terminology and basic features
  - Summary of display and report formats
  - Outline of the Manual
- Getting started
  - Sign-on
  - Help mode
  - Sample run
- Modes of operation:
  - Commands/Dialogues/Reports
- Advanced Features
- Command Syntax and system options

Users of Requirements Specification

Customer or end user	**Provide the requirements, check that they meet the needs**
Marketer or salesperson	Define a competitive feature set based on the voice of the buyer and manage product releases
Product developer	Use the requirements to understand what the system is to do and build the system
Quality assuror	Use the requirements to verify & validate the system through testing and asking customer.
Different regions and countries	Take the core product and enhance with local features.

Purpose of High-Quality Requirements Specification

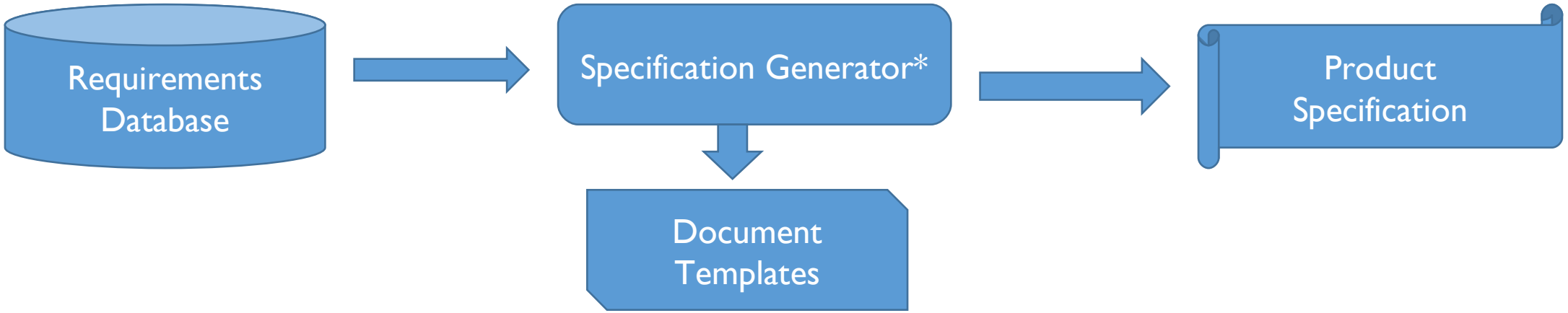
A high-quality requirements specification

- Is the documentation of all requirements describing what a product needs to provide
- Is the basis for a commercial contract to build the solutions
- Is the basis for deriving a test plan
- Defines the product's requirements in a measurable way
- Is the prerequisite for tracing of product requirements, and
- Influences the project plan to build the product

Usually, the specified solution is a system…

Generating a Requirements Specification

- The preferred mechanism for creating any type of requirement specification is to generate it from a requirements database
- The specification is never modified. The requirements in the database are modified and the document is regenerated



Criteria for a High-Quality Requirements Specification

- Feasible
- Correct = Validated
- Unambiguous
- Testable = Verifiable
- Modifiable
- Consistent
- Complete
- Traceable
- Project-or-product-specific other characteristics

When we are discussing written requirements specifications

➢Concise

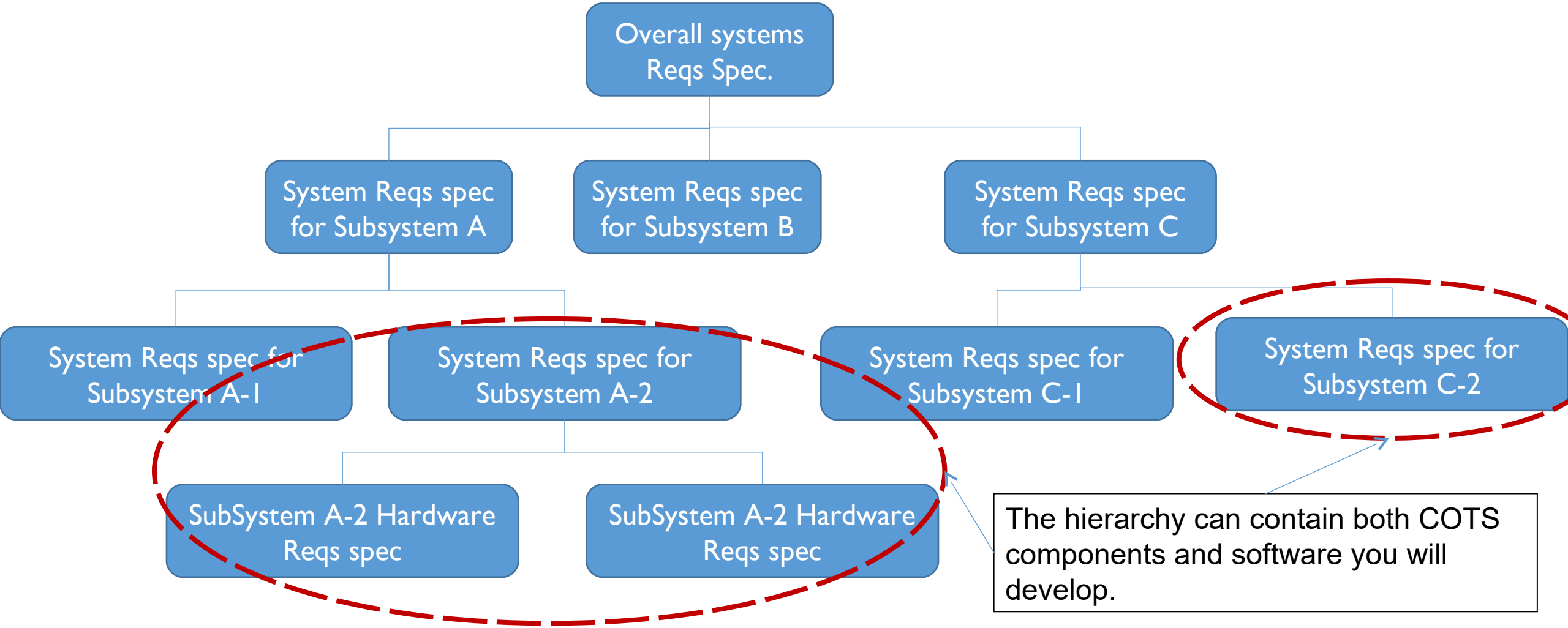
Concise

- Definition: A requirement statement is concise if
  - It describes a single property of the desired system
  - Includes no information beyond that necessary to describe the property
  - And is stated in clear, simple, and understandable terms.
  - Avoid words such as "should", "could", "might", and "may".
- Example:
  - "Emergency calls from the public shall be answered in the order in which they are received" is concise, but
  - "Emergency calls from the public should be answered in precisely the order in which they are received and stored in a first-in-first-out queue for being answered" is not concise

Conciseness – class exercise

- For each sentence, determine how the sentences is not concise, and rewrite the sentence to be more concise
- Exercises:
  - The program shall calculate the square root of the input accurate to one less decimal digit than the hardware's precision, using the Newton-Raphson iterative approximation.
  - To determine the first class passengers, the program shall step through the list of passengers, storing in a temporary file each passenger for which the passenger's class = "F" and then later shall present the elements of the temporary file as a list.

Structure of Requirements Specification



Conventions about Verbs in SRSs

- Indicative ("is" "was"): about domain independently of the system to be built
- "shall": about what the system is required to do, i.e., for stating a requirement of the system
- "will": about the domain after the system is deployed in it

Questions?

- How do you decide what to put where
- How do you decide what verbs to use?

- The next two dozen or so slides, until the section on ambiguities, are directed to answering these question.

SRS templates

- An SRS should be structured according to a standard prescribing predefined sections, i.e., a template.
- Templates for SRS make it easier to create uniform SRSs
- It is easier for QA to define SRS metrics.
- Templates are typically created for business and system requirements specifications as well as SRSs
- A template can be used to semi automatically generate an SRS from an RE database or a use-case model.

SRS Template Outline from IEEE-830

- Introduction
- Glossary
- Specification of customer's requirements
- System architecture
- Specification of system requirements
- System models
- Evolution of system
- Appendix
- Index

Another outline for SRS

I. Introduction
   A. System reference
   B. Overall description
   C. Software project constraints

II. Informal description
   A. Information content representation
   B. Information flow representation
      A. Data flow
      B. Control flow

III. Functional description
   A. Functional partitioning
   B. Functional description
      1. Processing narrative
      2. Restrictions/limitations
      3. Performance requirements
      4. Design constraints
      5. Supporting diagrams
   C. Control description
   D. Design constraints

Pro and Con of SRS Templates

Pro

- A template is helpful in that it is harder to forget to include important information when a complete outline is in front of you.

Con

- Not all sections of a template are relevant for all systems. If management or standards insist that all sections of a template be filled, people often write nonsense in the irrelevant sections, just to be able to fill the section and please management or the standards.
- Sometimes, the reader of an SRS e.g., an implementer, has difficulty distinguishing nonsense from true requirements.

Summary

- Write requirements as soon as possible
- Determine what attributes will be used to classify and elaborate requirements
- Produce an initial version to stimulate feedback
- Exposure to users is much better than analysis by "experts"
- Rules to follow when writing requirements:
  - Use simple direct language
  - Write testable requirements
  - Use defined and agreed terminology
  - Write one requirement at a time