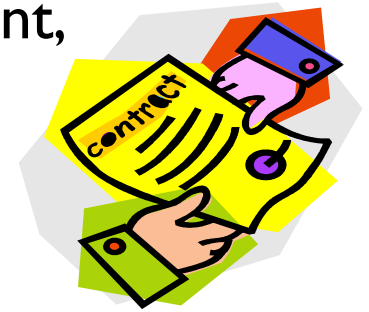
Software Requirements Specification (SRS)

- 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



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

CS10102302
UML交互建模

赵君峤 副教授

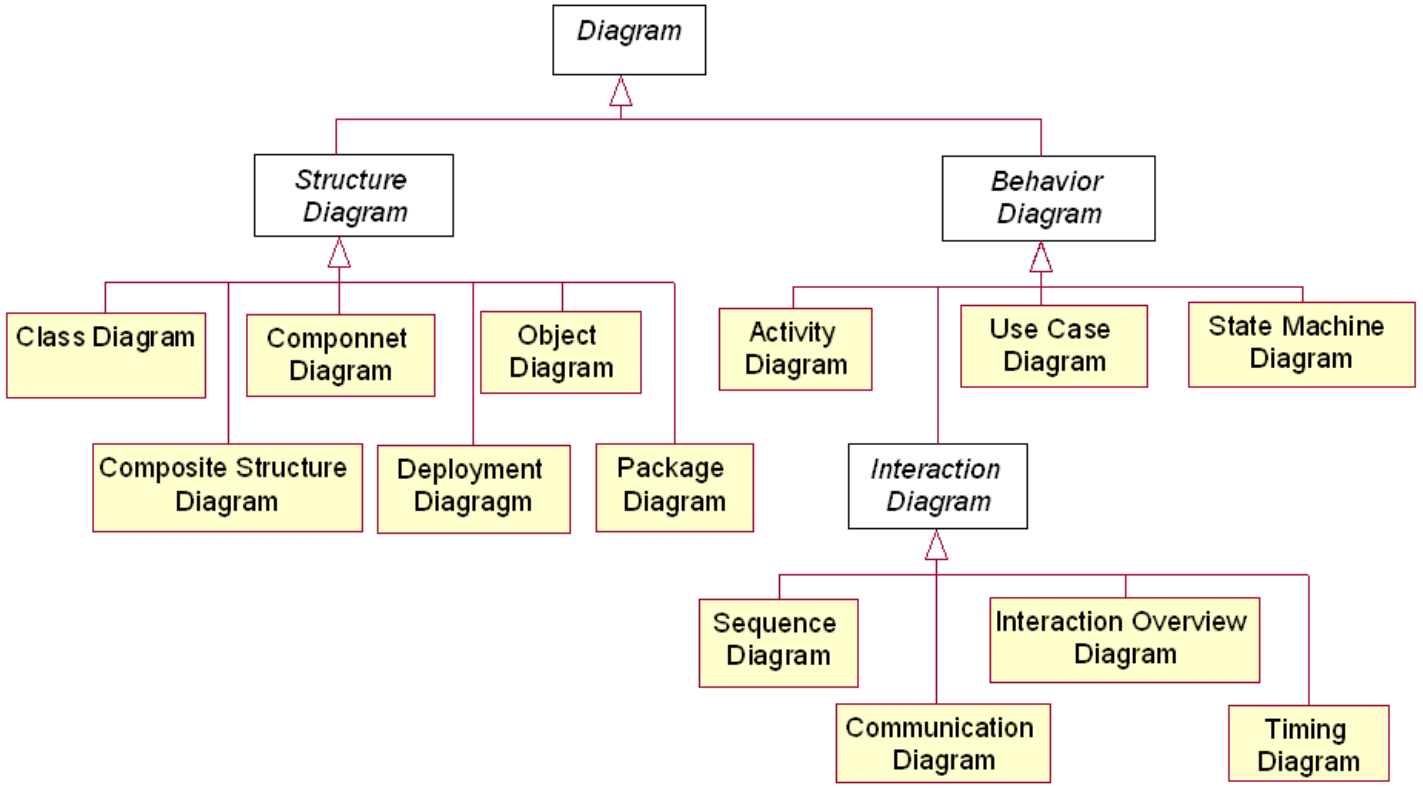
计算机科学与技术系 电子与信息工程学院

同济大学

Why Build Models?

- To understand the problem better
- To communicate with other persons
- To find errors or omissions
- To plan out the design
- To generate code

Classification of Diagrams in the UML 2.0



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）

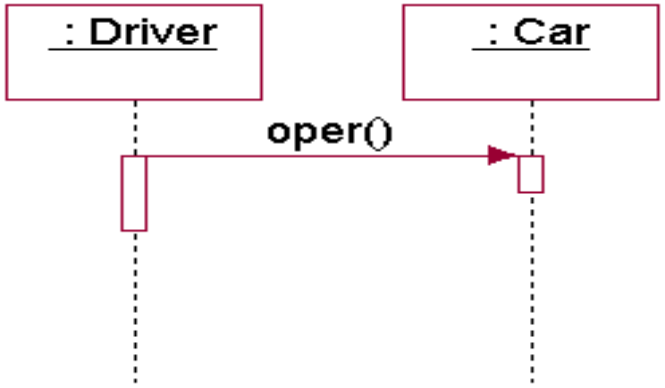
UML sequence diagrams

- **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。

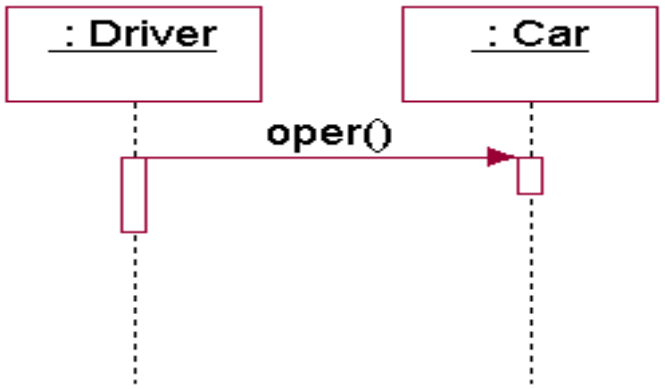
- In particular, it shows the **objects** participating in an interaction and the **sequence of messages** exchanged. 主要表示有哪些对象参加交互，以及消息传递的序列。



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.

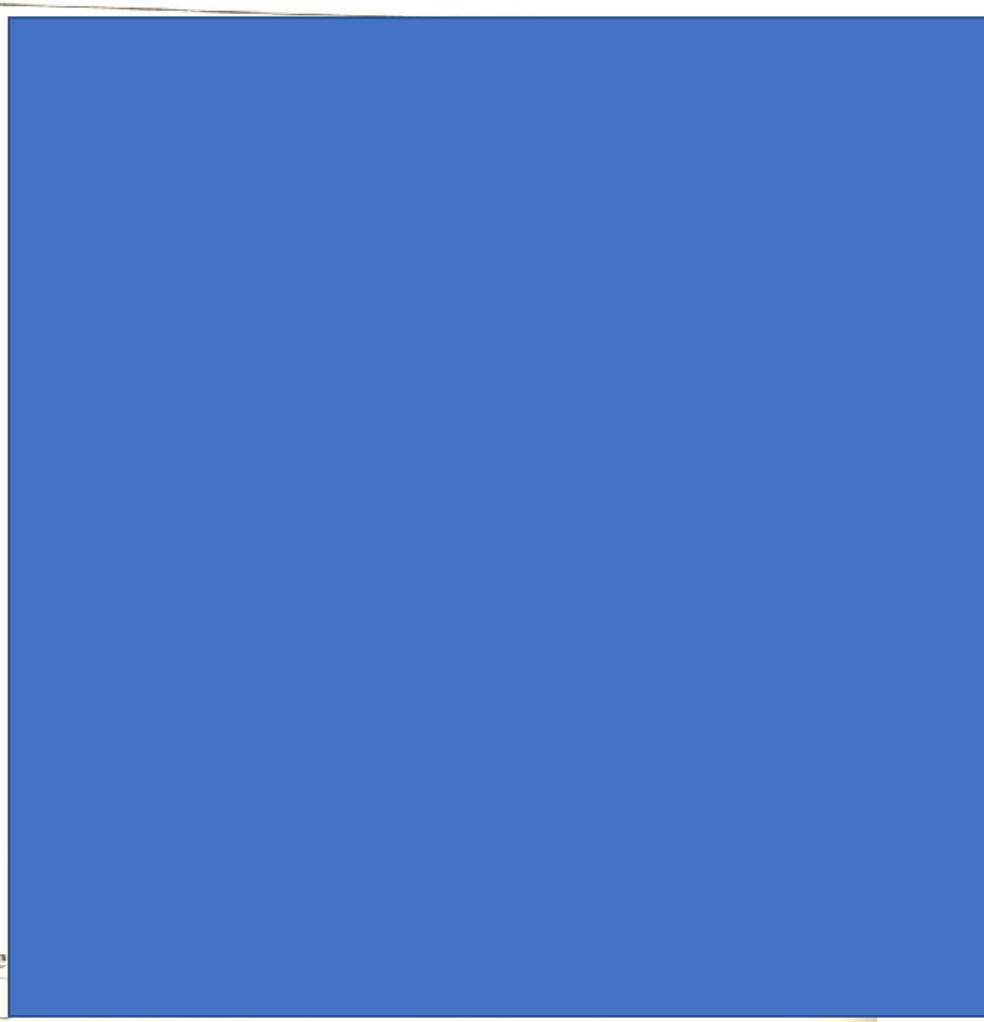
  顺序图表达对象间如何协作完成用例所描述的功能。

- 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

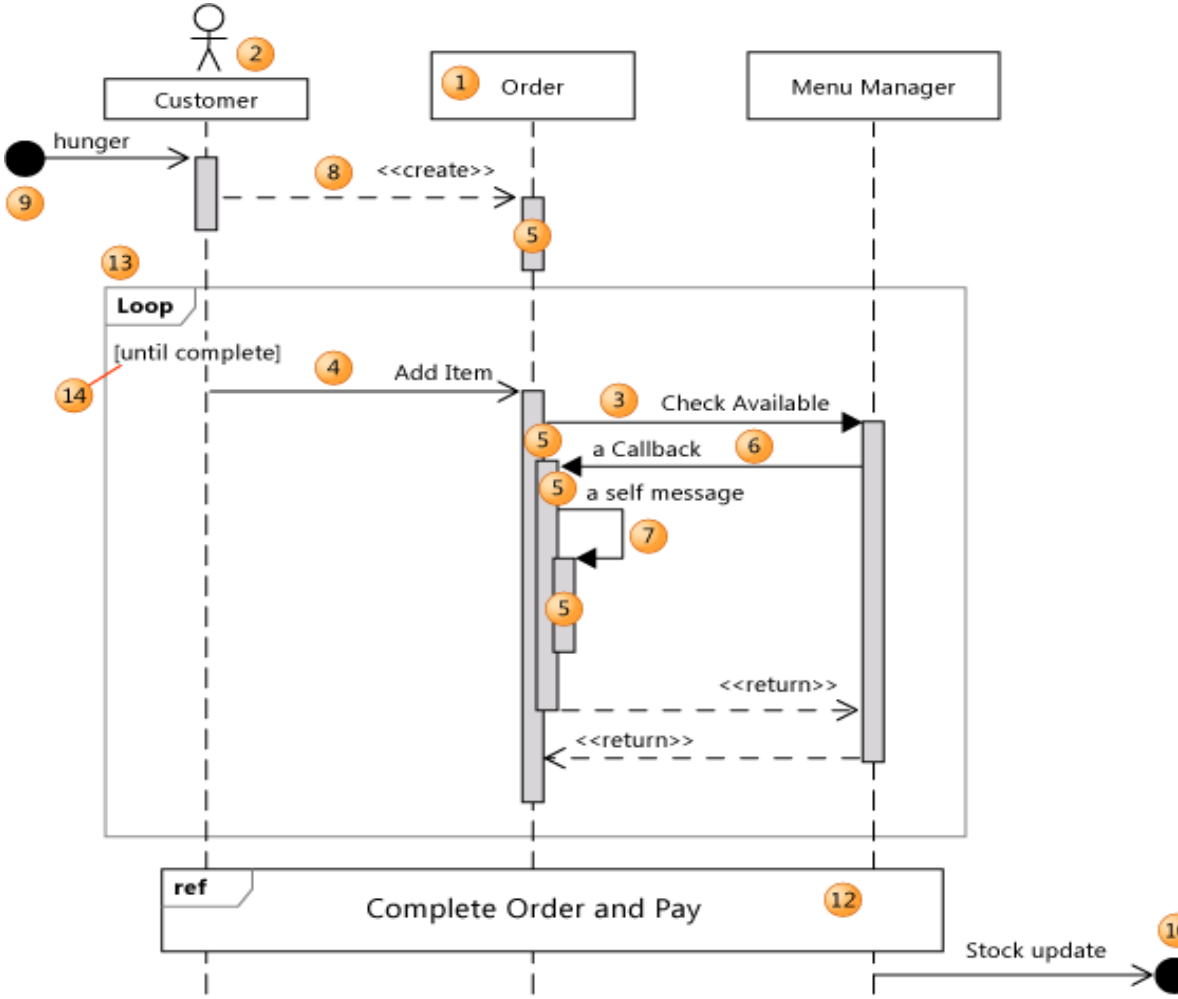
- The vertical line for each object represents its lifeline.

  每个对象下面的竖线表示该对象的生命线。

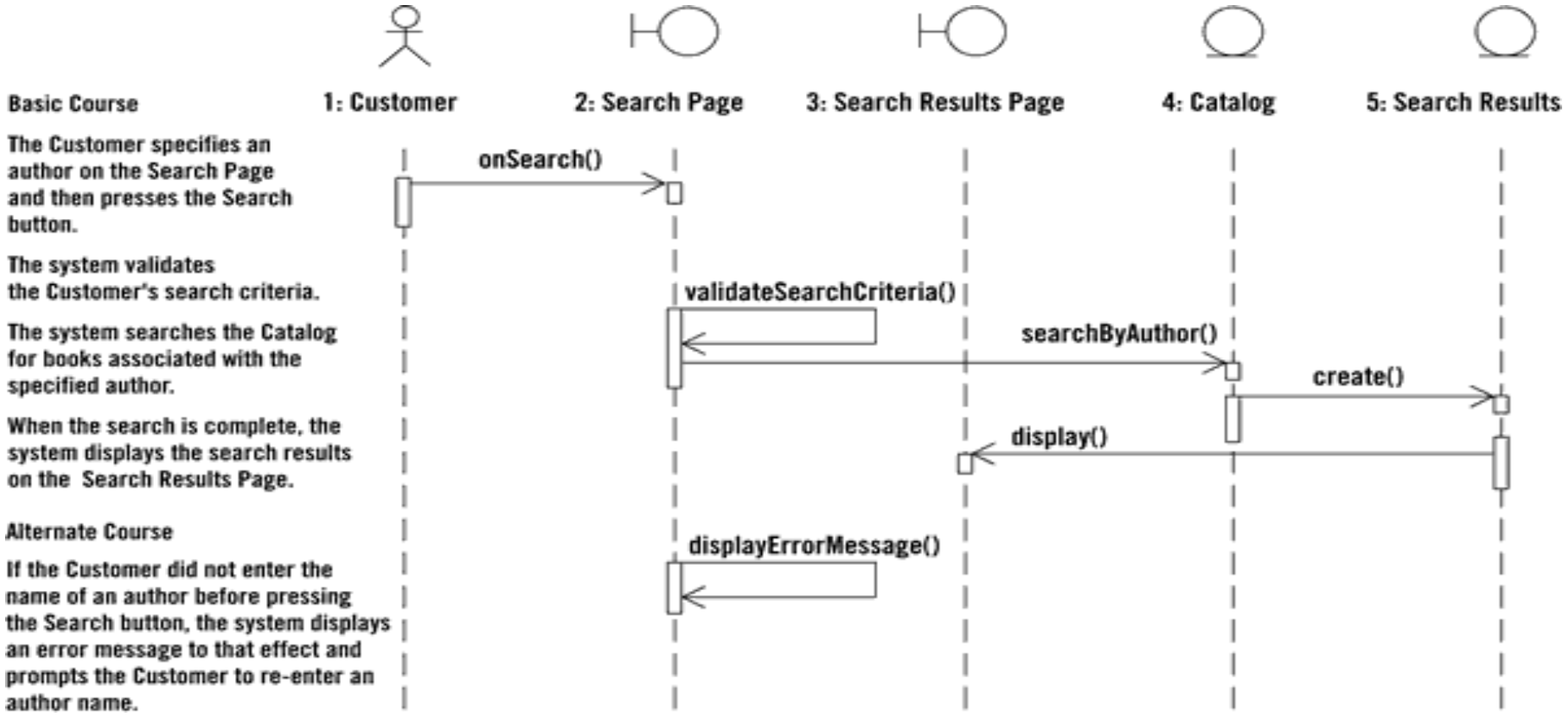
- The thick bar represents its activation. 生命线上的矩形框表示对象的激活期
  - i.e. when it is doing something 即正在执行某项操作

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)

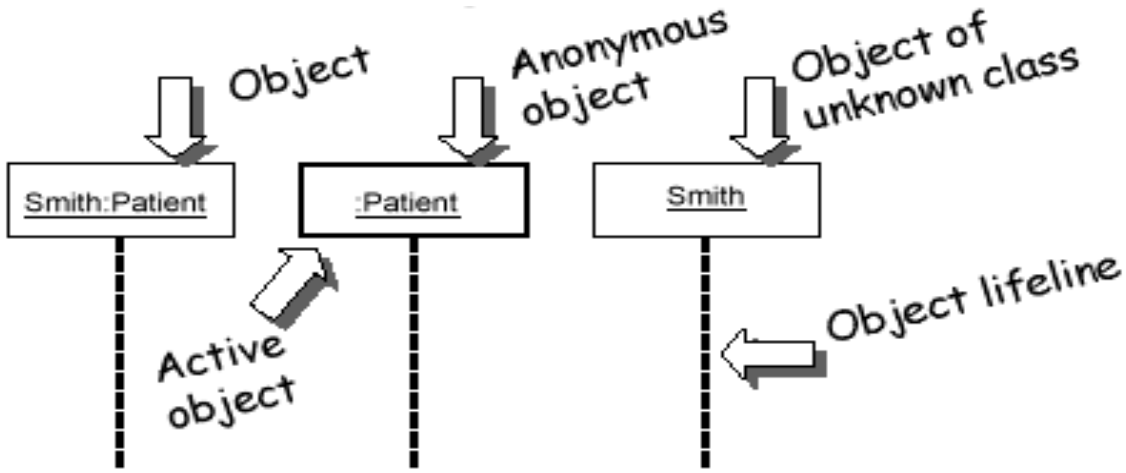


Sequence diagram from use case



Representing objects

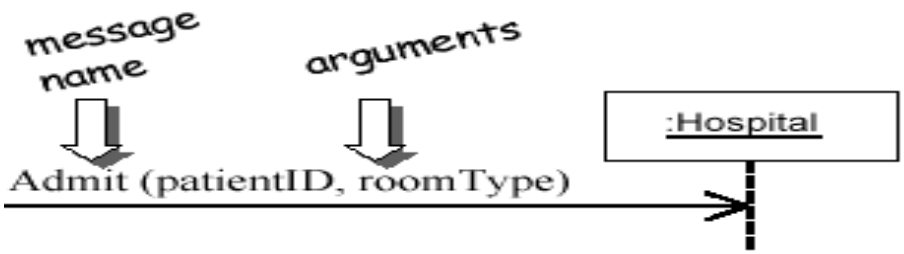
- squares with object type, optionally preceded by object name and colon
  - write object's name if it clarifies the diagram
  - object's "life line" represented by dashed vert. line



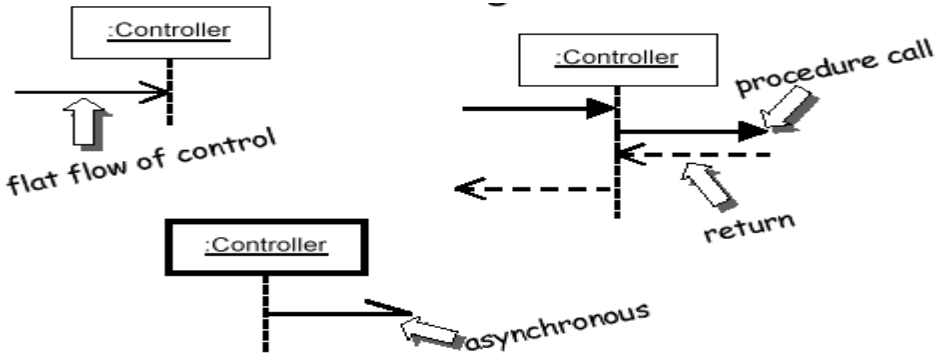
Name syntax: <objectname>:<classname>

Messages between objects

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

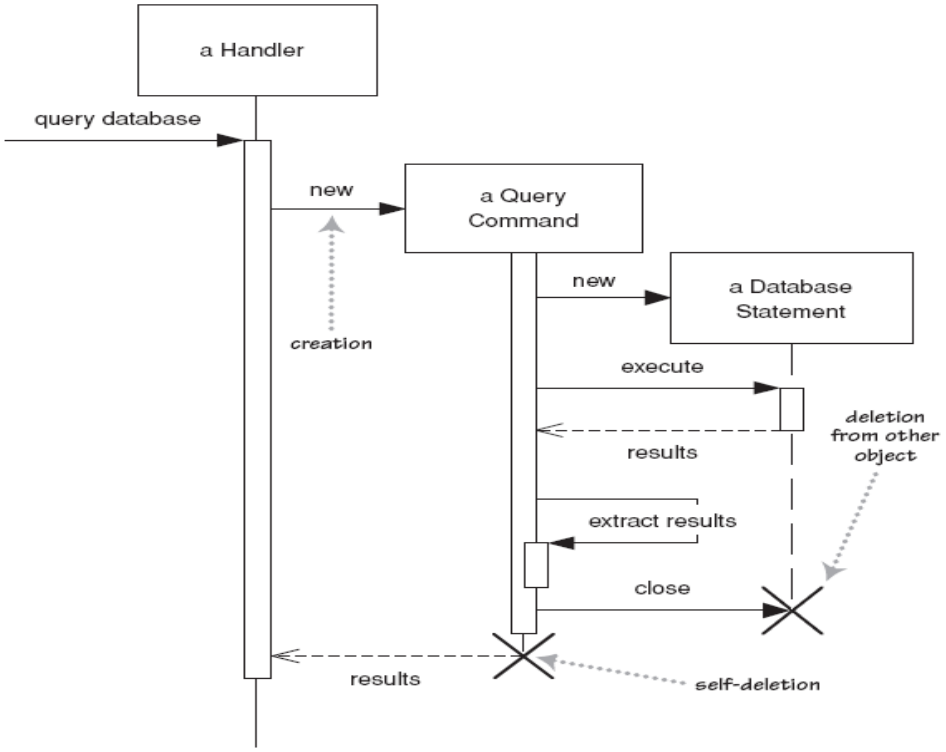


  - dashed arrow back indicates return
  - different arrowheads for normal / concurrent (asynchronous) methods



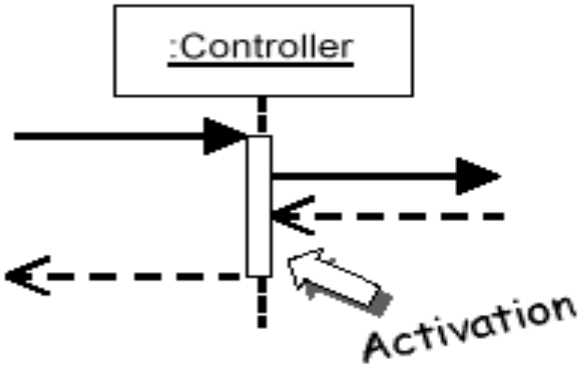
Lifetime of objects

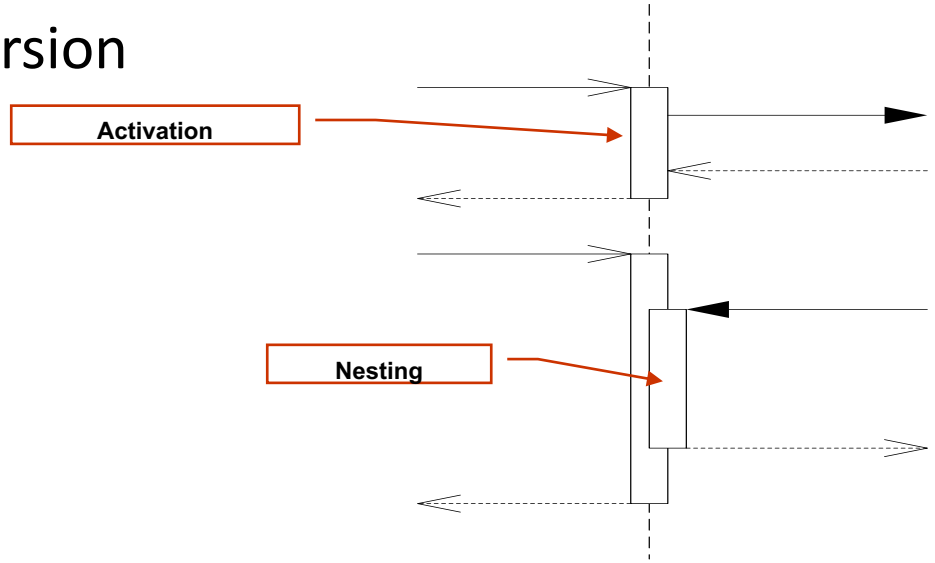
- creation: arrow with 'new' written above it
  - notice that an object created after the start of the scenario appears lower than the others
- deletion: an X at bottom of object's lifeline
  - Java doesn't explicitly delete objects; they fall out of scope and are garbage-collected



Indicating method calls

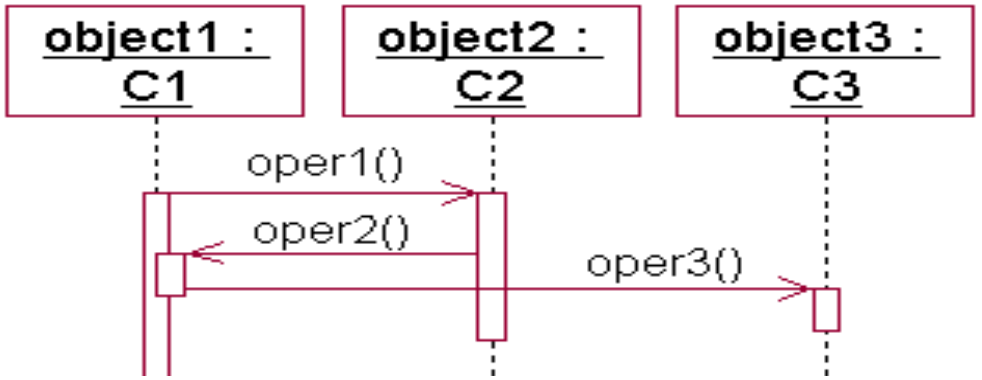
- **activation**: thick box over object's life line; drawn when object's method is on the stack
  - either that object is running its code, or it is on the stack waiting for another object's method to finish
  - nest to indicate recursion





focus of control的嵌套

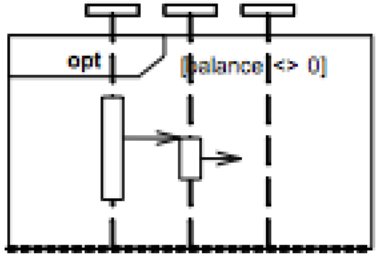
- 嵌套的FOC可以更精确地说明消息的开始和结束位置。
- 图例：

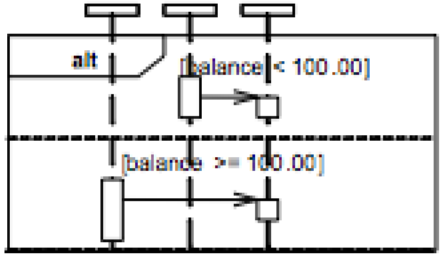


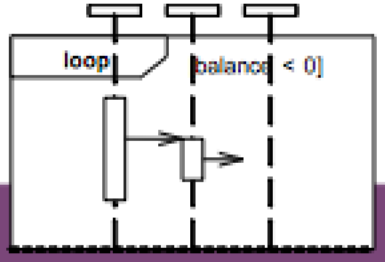
activation(激活期)表示对象执行一个动作的期间，也即对象激活的时间段。
An activation represents the period during which an object performs an operation either directly or through a subordinate operation.
FOC和**activation**是同一个概念。

Indicating selection and loops

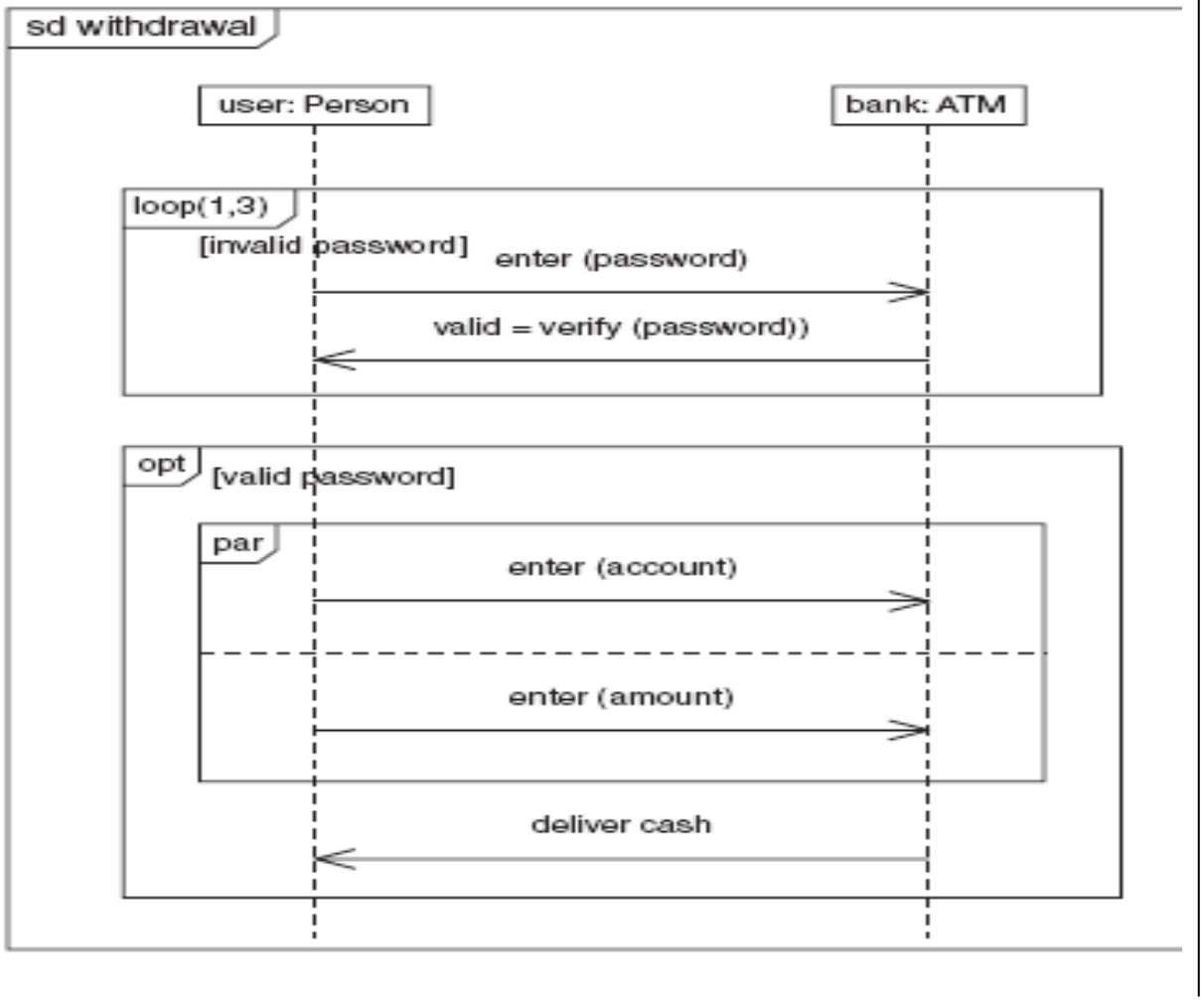
- frame: box around part of a sequence diagram to indicate selection or loop
  - if -> (opt) [condition]
  - if/else -> (alt) [condition], separated by horiz. dashed line
  - loop -> (loop) [condition or items to loop over]

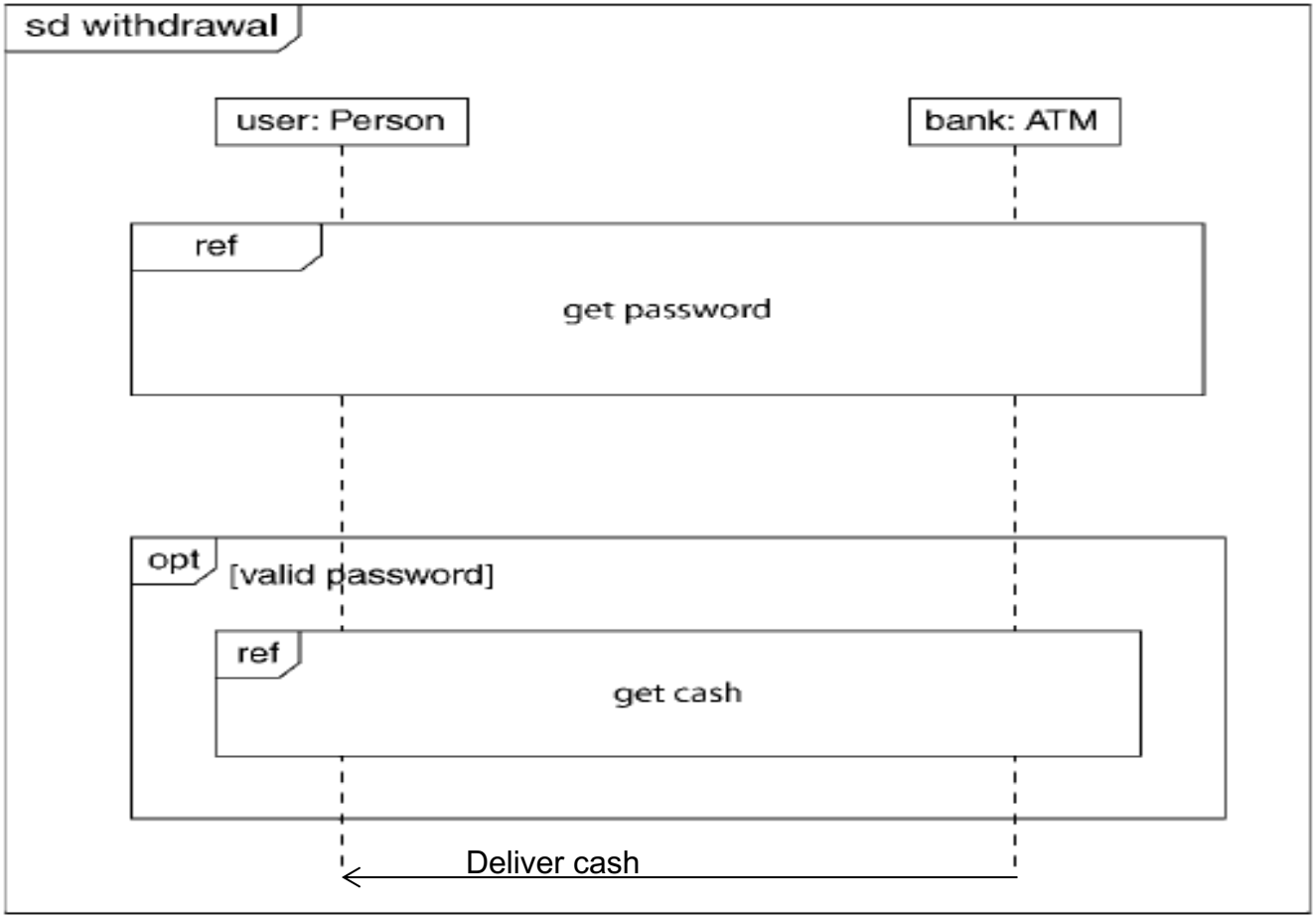


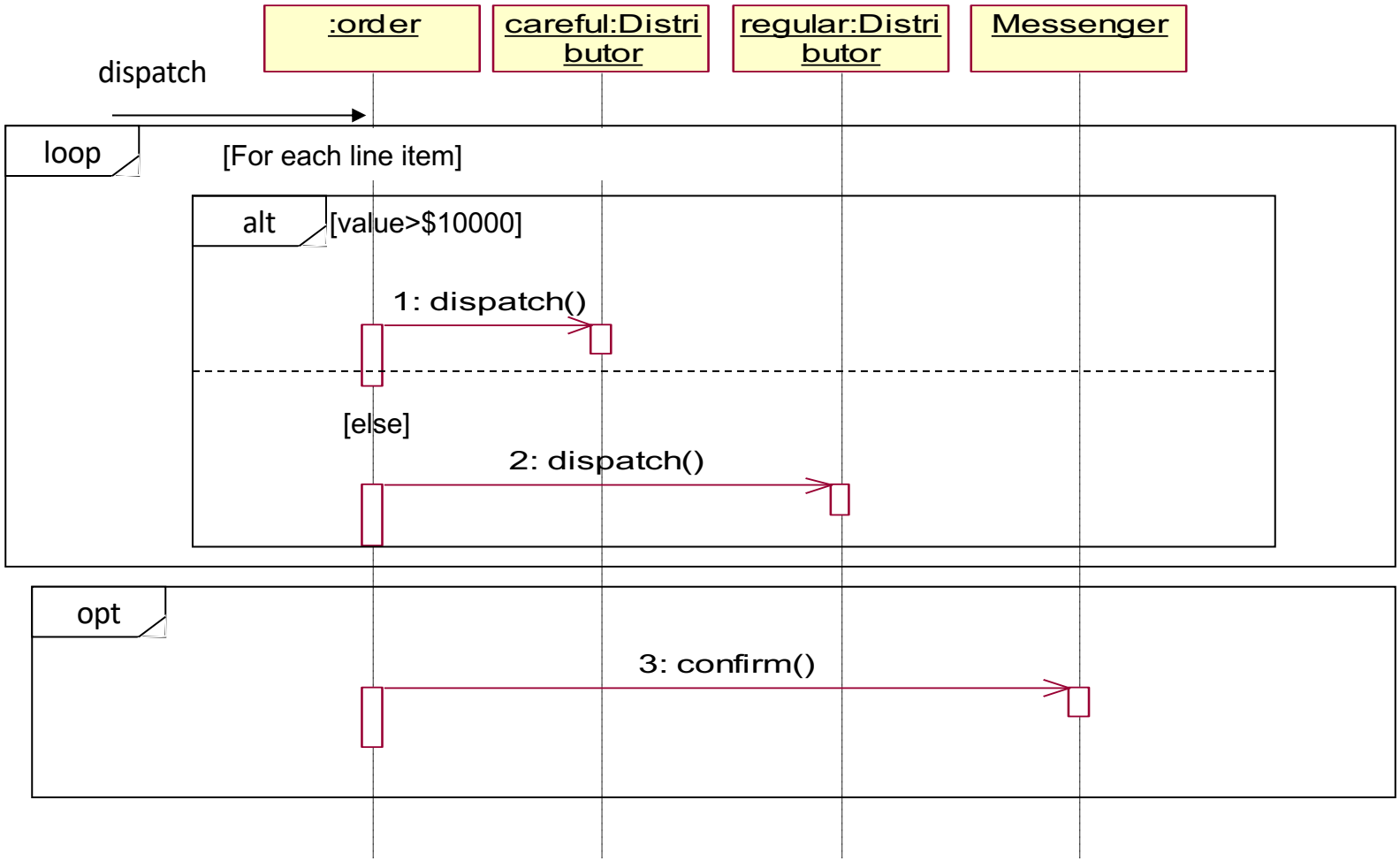




Another Example

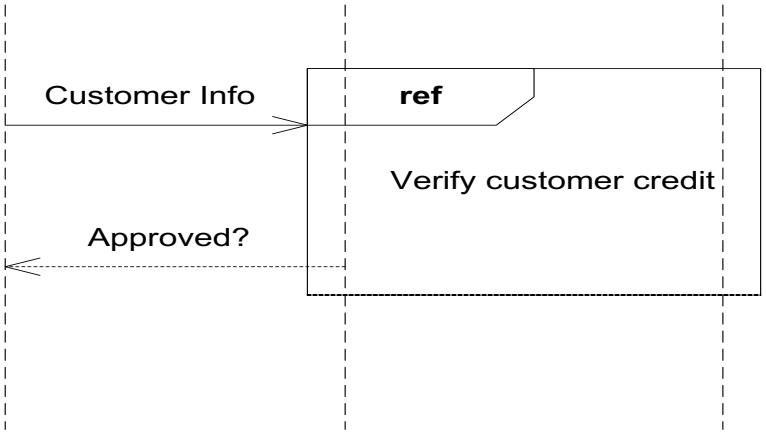


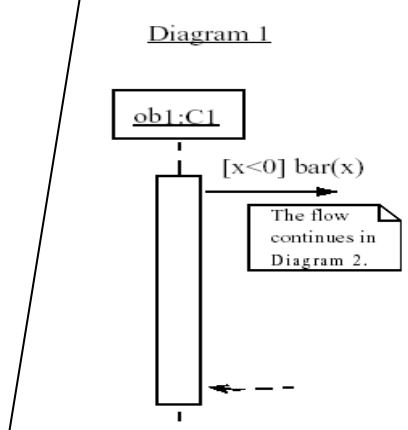

sd withdrawal
user: Person
bank: ATM
ref
get password
opt
[valid password]
ref
get cash
Deliver cash

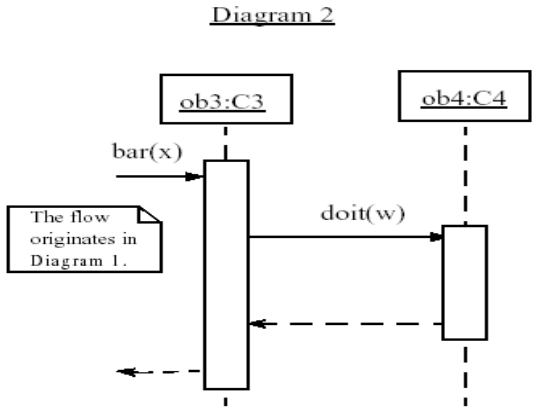

:order
careful:Distributor
regular:Distributor
Messenger
dispatch
loop
[For each line item]
alt
[value>$10000]
1: dispatch()
[else]
2: dispatch()
opt
3: confirm()

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?



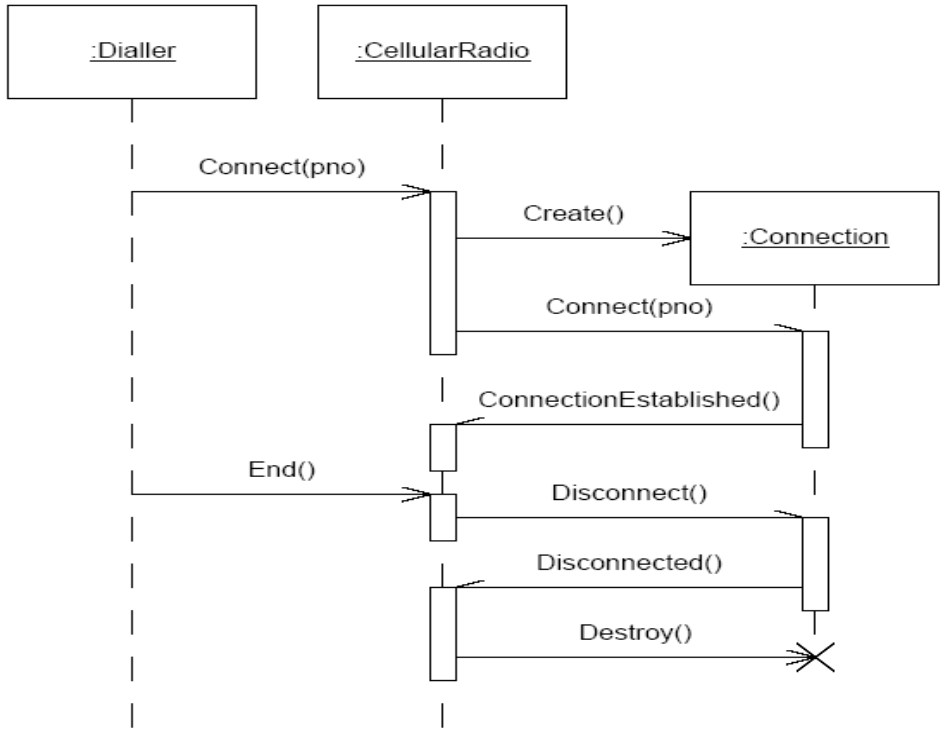




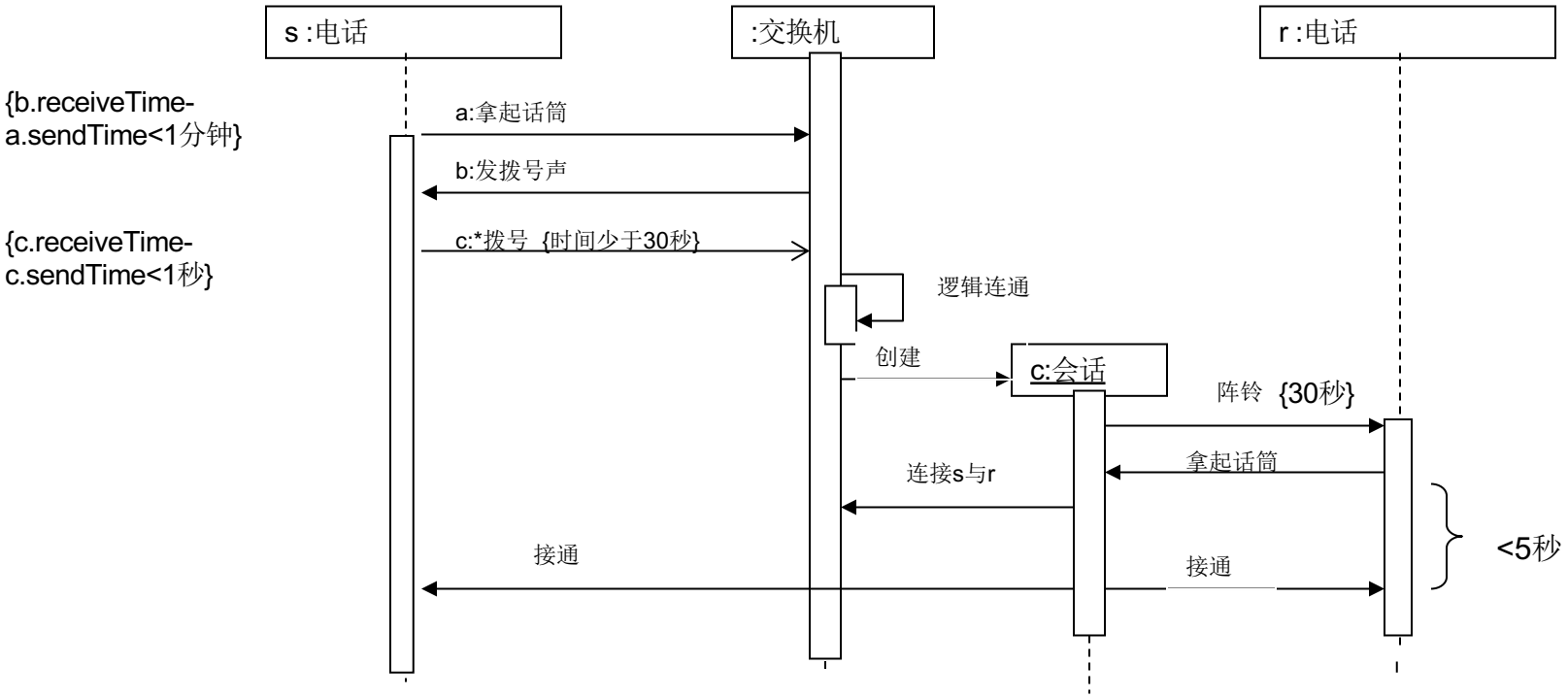
Flawed sequence diagram 1

- What's wrong with this sequence diagram?

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



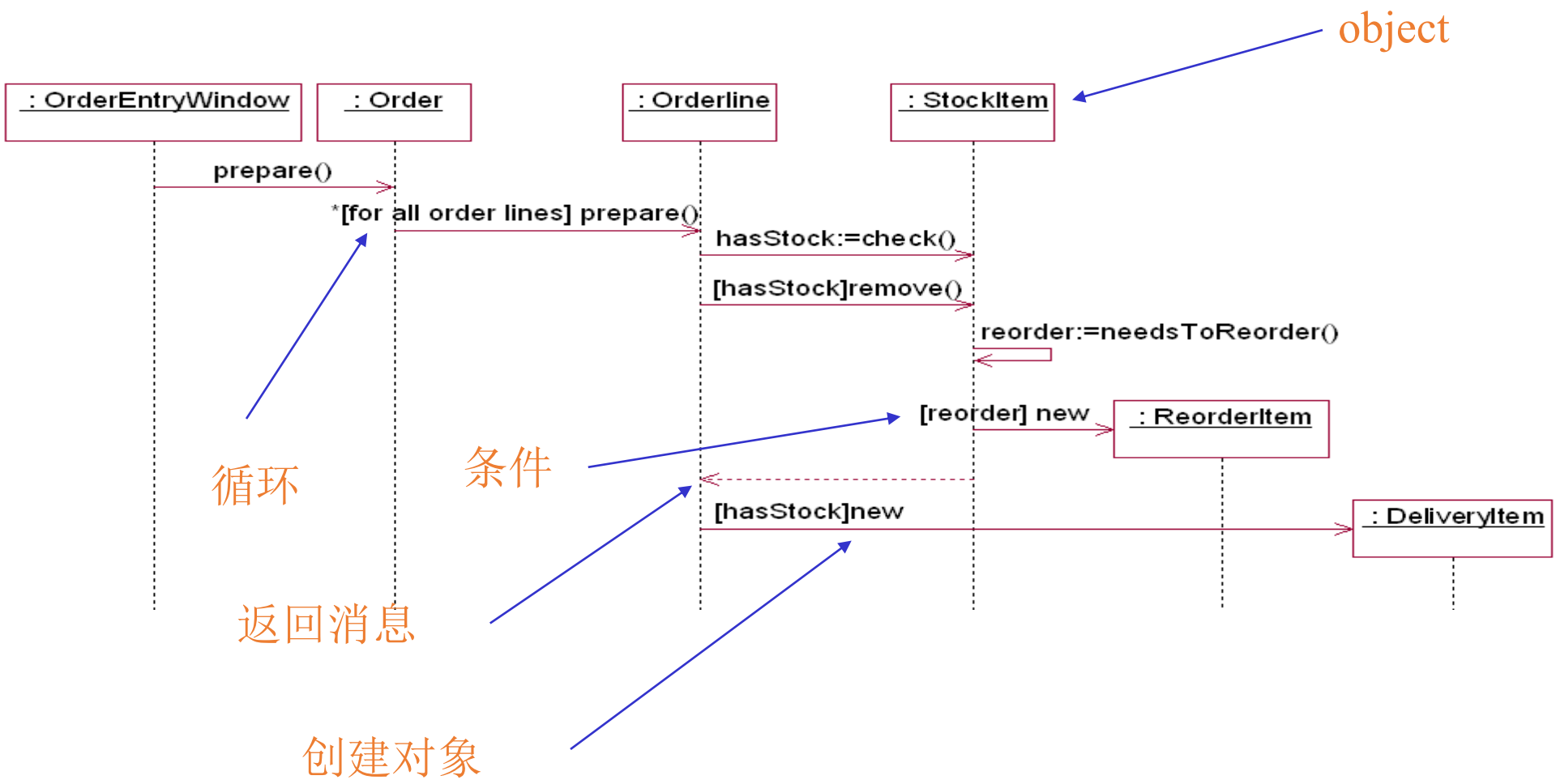
Possible right answer



问题：时间超过30秒的情况没说明

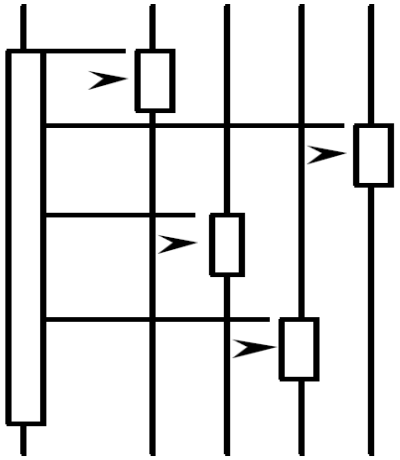
会话对象没有说明计费等情况

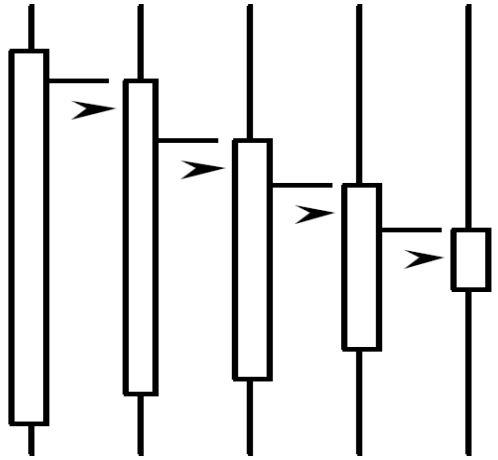
Some concepts used in a sequence diagram v1.5



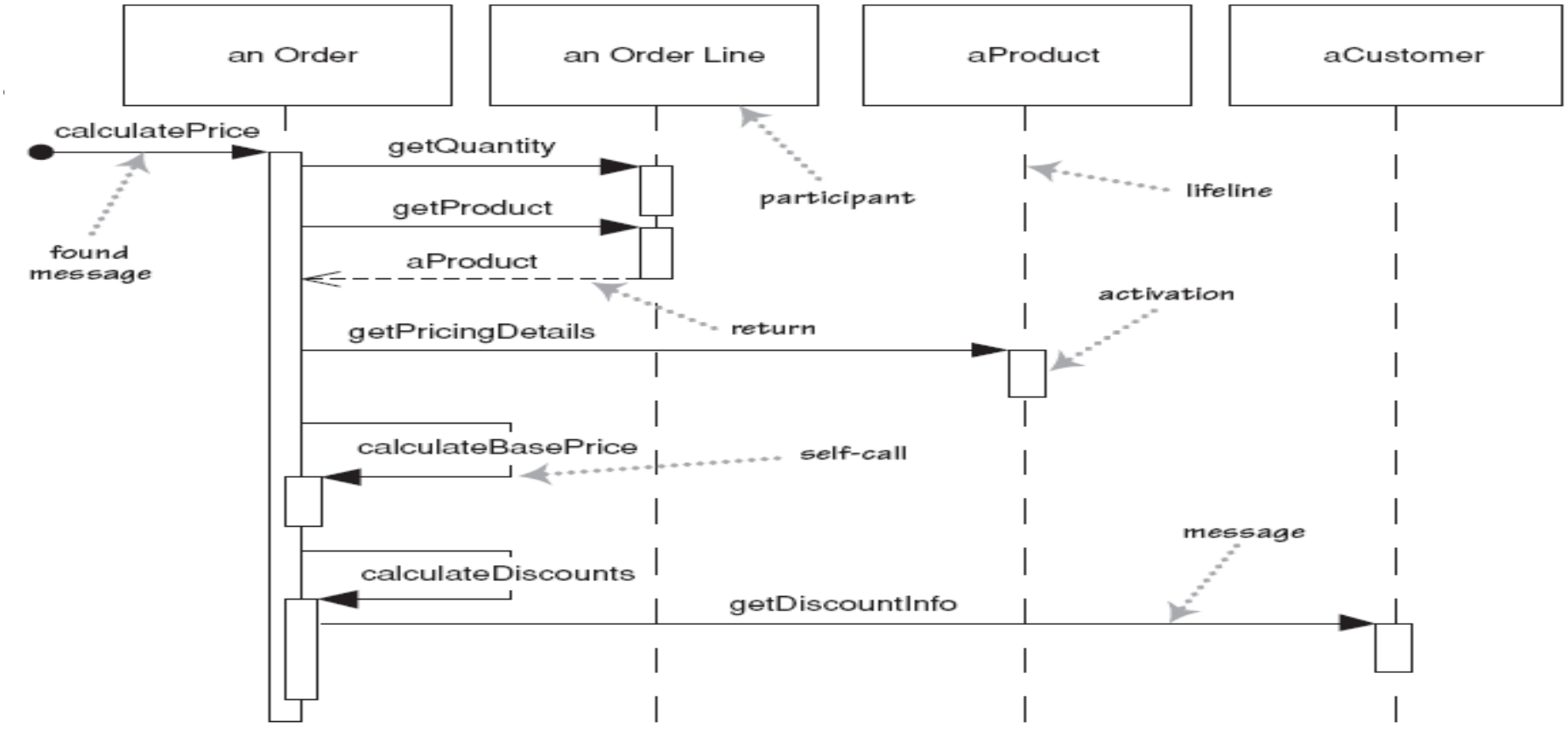
(De)centralized system control

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

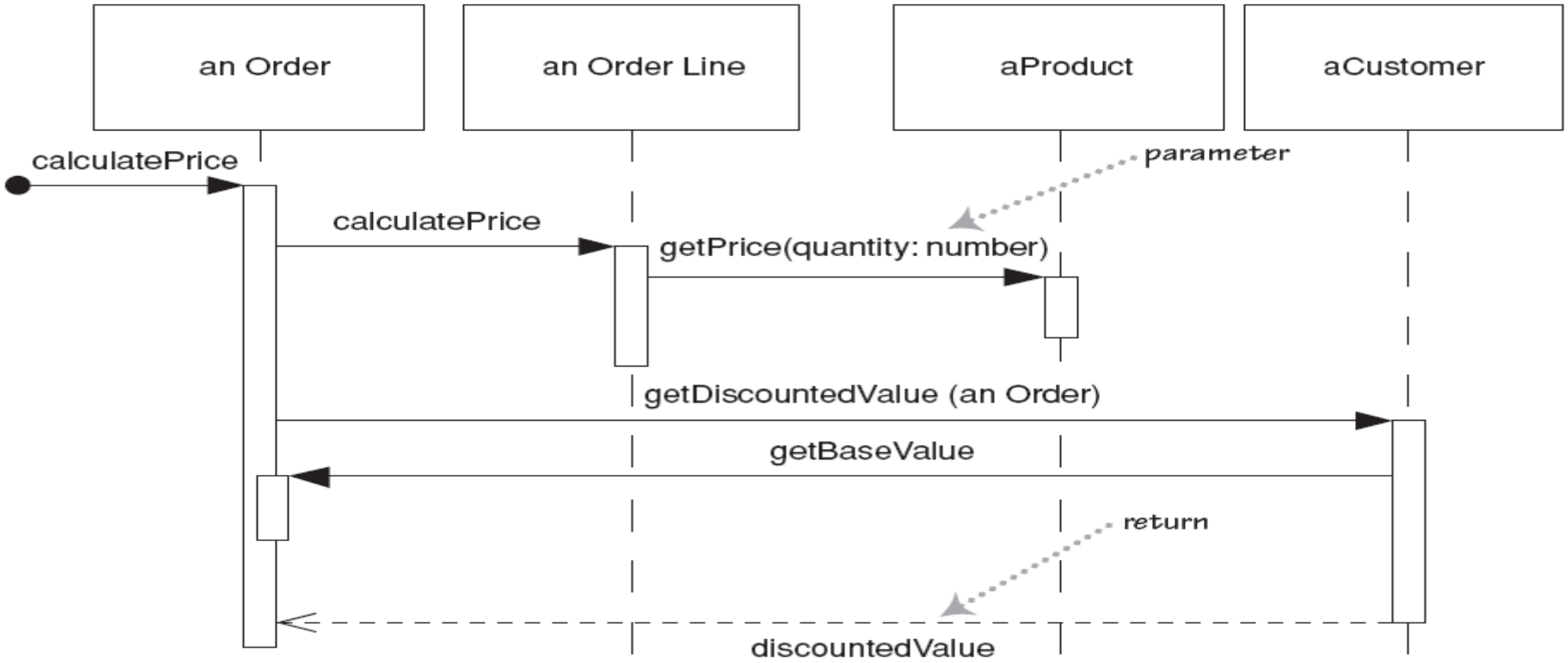




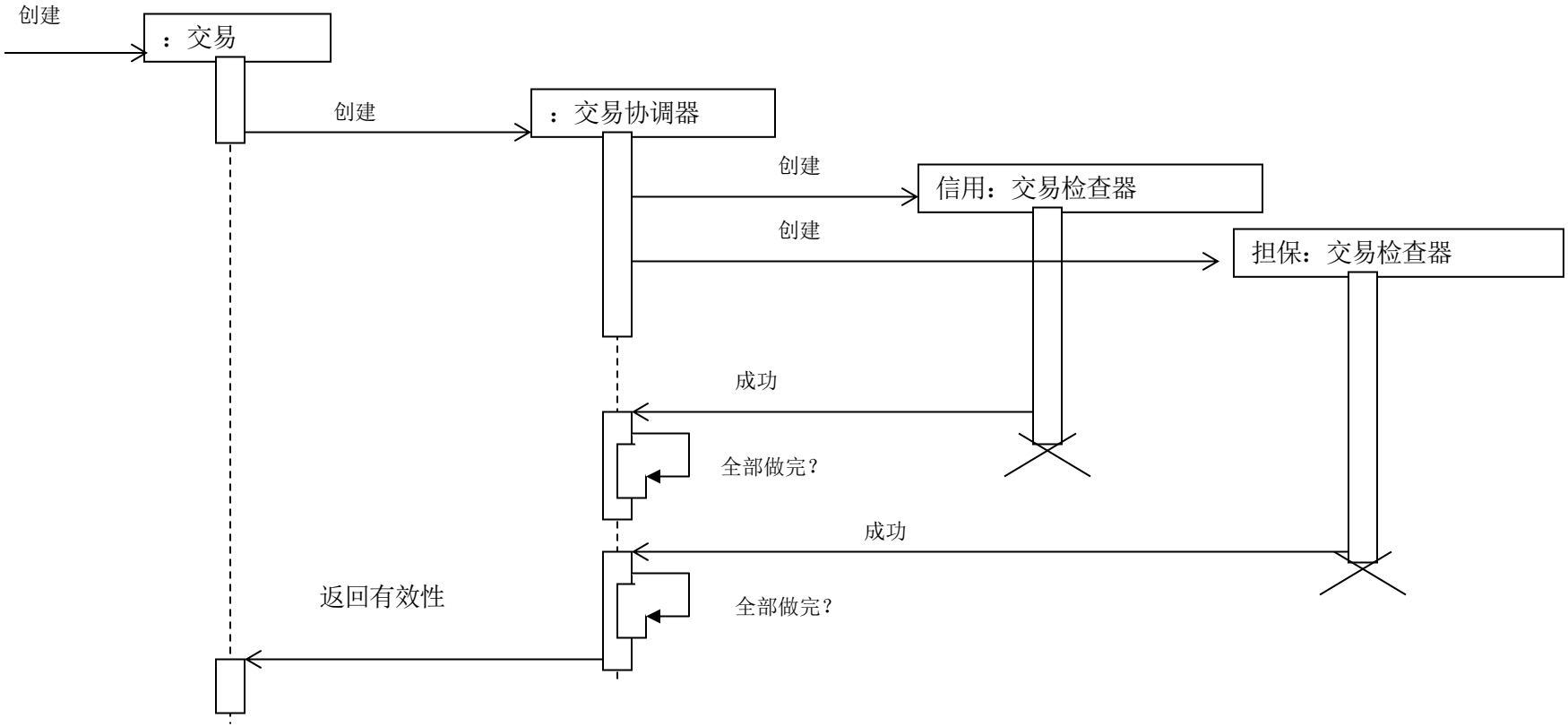
Example 1: A sequence diagram for centralized control
集中控制的计价系统顺序图



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



银行系统的交易验证



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)

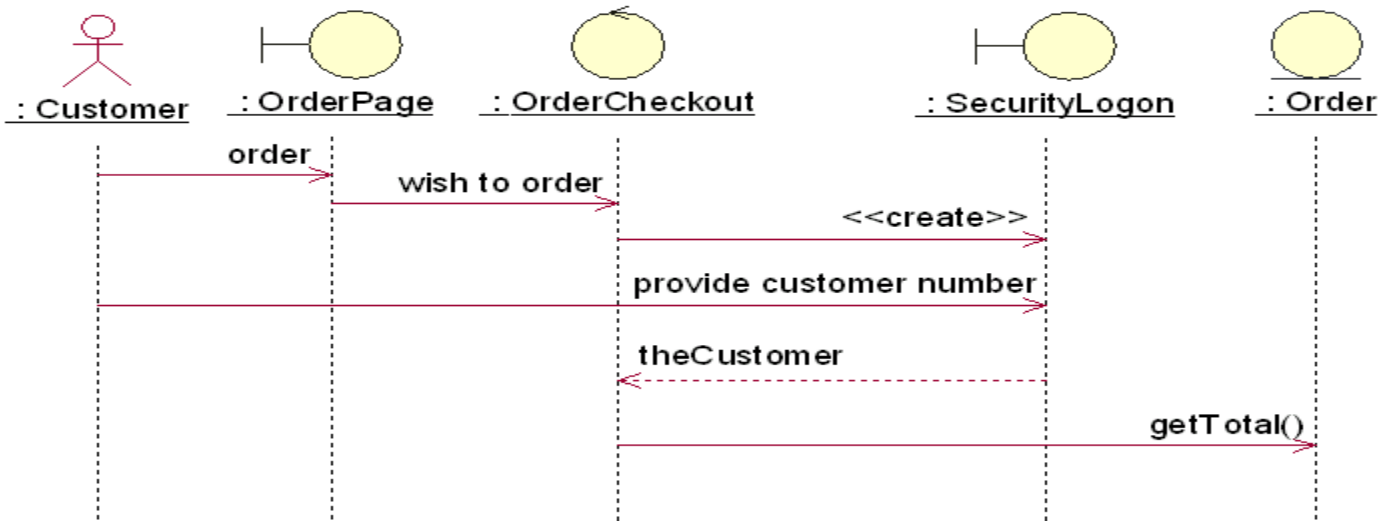
顺序图建模风格

- 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。

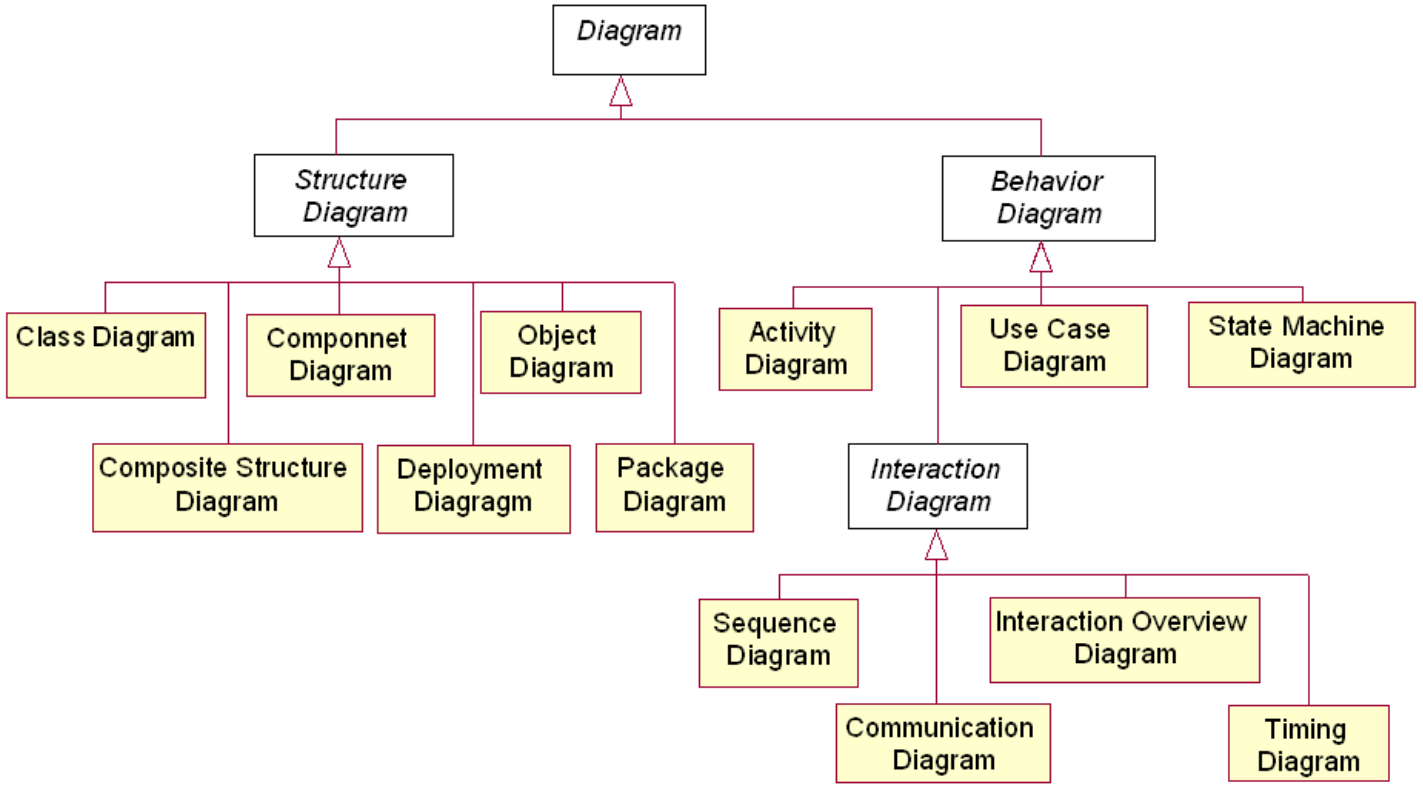
- 例如，消息addDeposit(amount, target)比addDeposit(Currency, Account)传递了更多的信息。
- 在消息中只使用类型信息不能传递足够的信息。
- 参数的类型信息用UML类图捕获更好。
- 建模风格只是建议，不是规定。如果只是需要表示“有某些东西要传递”这个事实，而不需要提供更多的细节，则可以指明参数的类型作为占位符。

- 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。

Classification of Diagrams in the UML 2.0



Collaboration/Communication (协作/通信图)

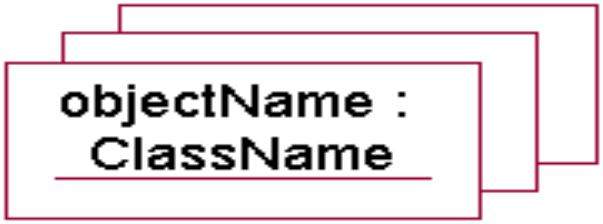
- **协作图**包含一组对象和链(link)，用于描述系统的行为是如何由系统的成员协作实现的。

- 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。

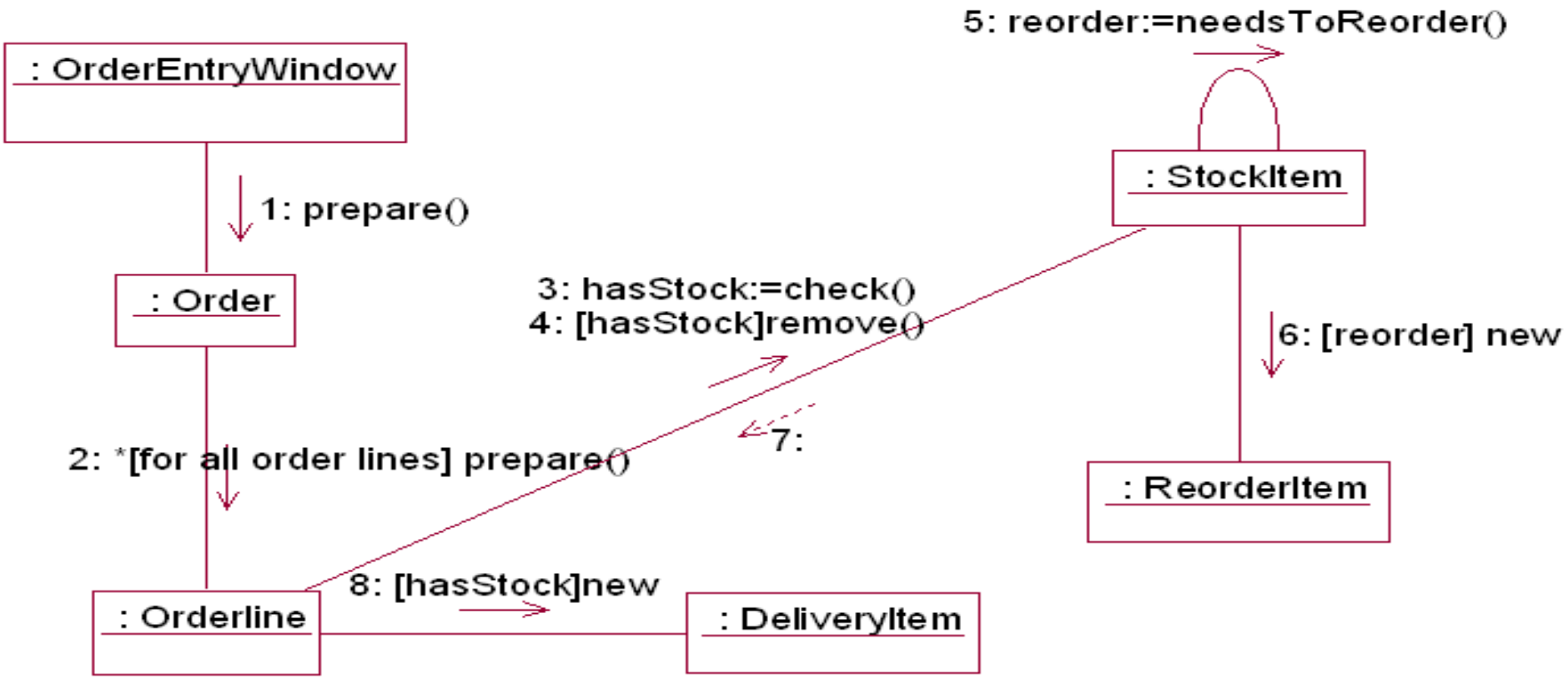
协作图中重复和条件分支的表示

- 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



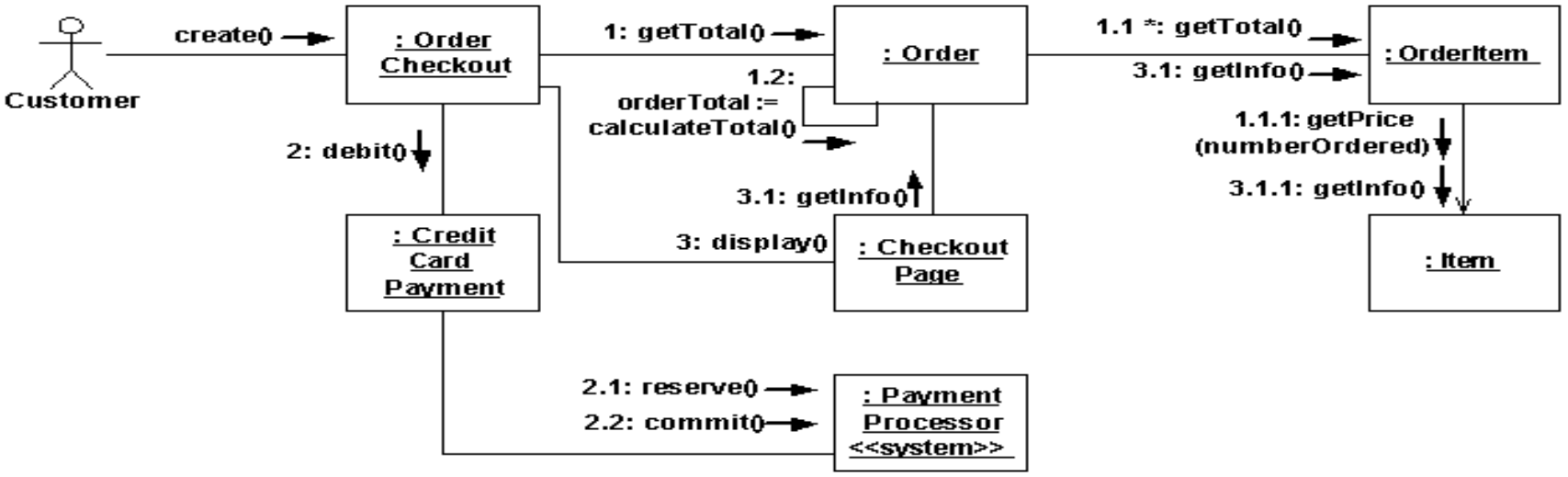
协作图建模风格

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：



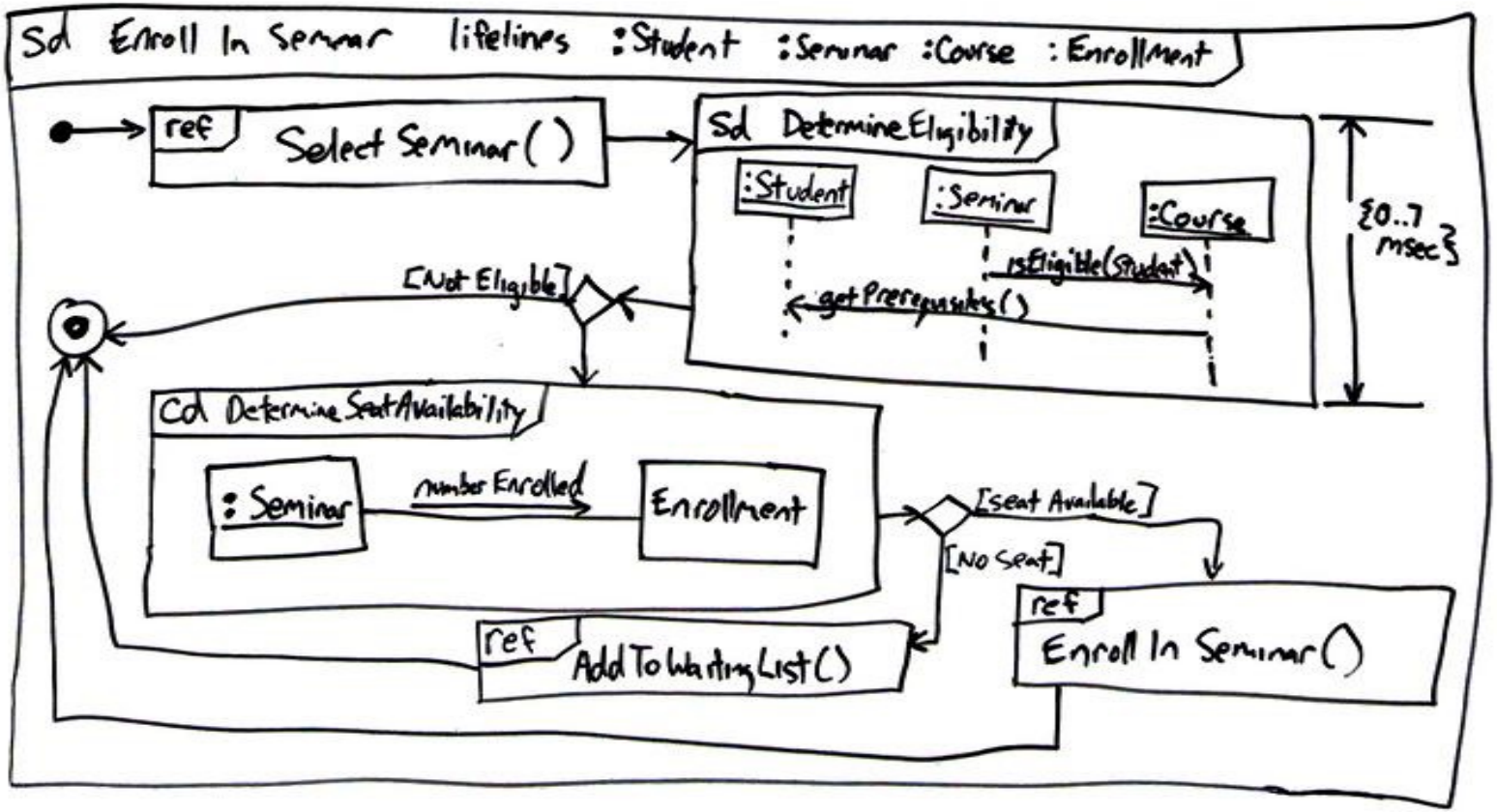
顺序图和协作图的比较

- 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。

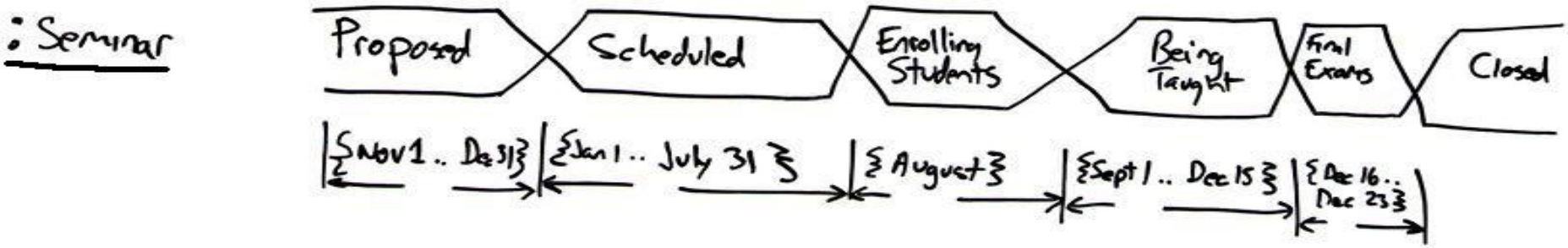
- Interaction Overview Diagram (交互概要图)

  Mix of sequence and activity diagram



- Timing Diagram (定时图)

  Interaction between objects; emphasis on timing



Reference

- 清华大学国家级精品课程 《软件工程》主讲人 刘强 副教授 刘璘 副教授
- https://www.icourses.cn/sCourse/course_3016.html
- https://www.xuetangx.com/course/THU08091000367/5883555?channel=learn_title

谢谢大家！

THANKS