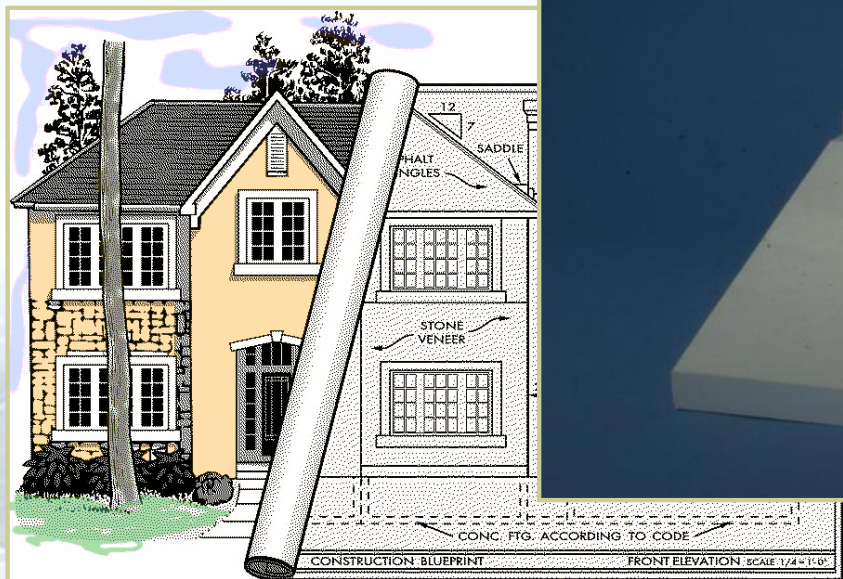




## 4.5 软件和软件开发过程的多视图特征

### 4.5.1 为什么单一视图解决不了问题？

- 工业化的成功经验是：从多个不同的层次和角度描述一个需要建造的产品。
  - 例如，建筑工业有：环境效果图、室内效果图、土建图、结构图、施工图等。





## 4.5 软件和软件开发过程的多视图特征

### 4.5.1 为什么单一视图解决不了问题？

- 软件的能力最终要靠程序来体现，而程序的文本是线性表示的，最终被转换成一维的指令序列。
- 一个多维结构到一维结构的拓扑不可能是唯一的，需要对多维结构给出多个投影（即视图）。
- 在没有完备的理论基础之前，视图越多，理解就越困难，保持一致性也越困难。
- 视图的种类要考虑信息系统具有的特征（数据、功能、行为）。
- 视图的种类要考虑软件开发过程的特征（分阶段、分层次、易于理解和交流、需求的可跟踪性等）。
- UML采用的是“4+1”视图。

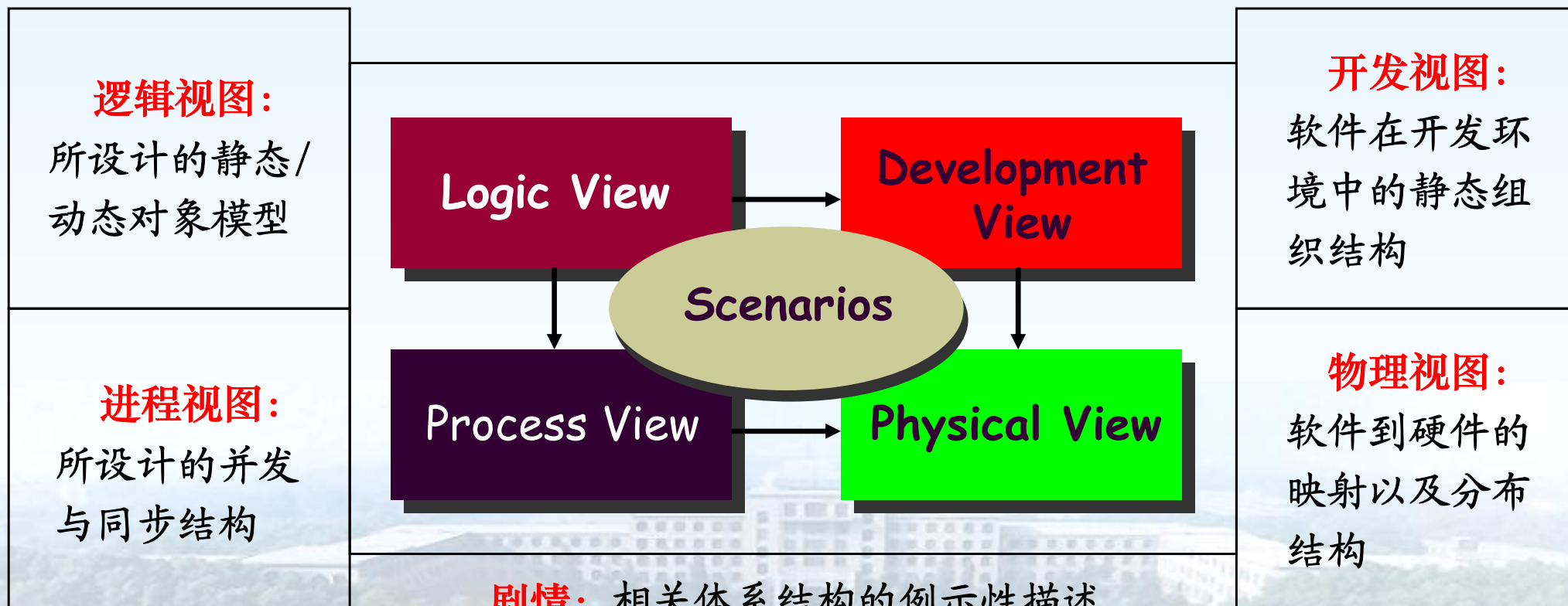


## 4.5 软件和软件开发过程的多视图特征

### 4.5.2 “4+1”视图

面向最终用户和程序员

面向项目管理者 and 程序员



面向系统集成人员

面向系统工程师





## 4.5 软件和软件开发过程的多视图特征

### 4.5.3 各种视图的 UML 表示手段

- **逻辑视图**：类图（*class diagram*）；协同图（*collaboration diagram*）；序列图（*sequence diagram*）；状态图（*state-chart diagram*）；活动图（*activity diagram*）；
- **进程视图**：组件图（*component diagram*）；
- **开发视图**：组件图；
- **物理视图**：部署图（*deployment diagram*）；
- **剧情**：用例图（*use case diagram*）；协同图；序列图。



## 4.6 UML的基本概念、结构 and 作用

### 4.6.1 UML 的发展过程

- 1994 年 10 月，Rational 公司的 Booch 和 Rumbaugh 决定将其 Booch 方法和 OMT 方法综合成一个新的建模语言，并于 1995 年 10 月公布了 Unified Method 0.8。
- 1995 年秋季，Jacobson 及其 OOSE 方法加入 Rational 公司，决定将 OOSE 方法与 Unified Method 进行综合，更名为 UML，并分别于 1996 年 6 月和 10 月公布了 UML 0.9 和 UML 0.91。
- 1996 年，DEC、HP、I-Logix、Itellicorp、IBM、ICON、MCI、Microsoft、Oracle、Rational、TI、Unisys 发起成立了 UML 成员协会，于 1997 年 1 月推出了 UML 1.0，并向 OMG 申请将其作为一种标准语言。



## 4.6 UML的基本概念、结构 and 作用

### 4.6.1 UML 的发展过程

- 1997 年 9 月产生了 UML 1.1, 11 月被 OMG 正式采纳。
- 1999 年 6 月, OMG 发布了 UML 1.3。
- 1999 年 7 月, UML RealTime 随 Rose RealTime 推出。
- 2001 年 9 月, OMG 发布了 UML 1.4。
- 2003 年 3 月, OMG 发布了 UML 1.5。
- OMG 称, UML 2.0 已经接近完成, 即将发布。



## 4.6 UML的基本概念、结构 and 作用

### 4.6.2 UML 的目标

- 提供易用的、表现力强的可视化建模语言；
- 提供可扩展、可定制的核心扩充机制；
- 不依赖于特定的程序设计语言 and 开发过程；
- 提供形式化基础以利于理解建模语言；
- 促进面向对象工具的市场拓展；
- 支持高层开发概念（如协同、构架、模式、部件等）；
- 集成最好的实践经验。





## 4.6 UML的基本概念、结构 and 作用

### 4.6.3 UML的基本概念与结构

- UML 并不是一种面向对象的开发方法，而是一种可视化的面向对象建模语言。它以 OOSE 为思维主干、以 OMT 为表示主体、辅以 Booch 表示。
- UML 把建模语言与开发过程明确进行了分离，专注于建模语言。
  - 过去的软件开发方法既规定了对应的开发过程，又给出了注记体系（*notation*），使得同样或类似的面向对象开发过程存在着形形色色的注记体系，难学难用，事与愿违。





## 4.6 UML的基本概念、结构 and 作用

### 4.6.3 UML的基本概念与结构

- UML 表示机制的层次结构:

1. 用例图

2. 类图

3. 行为图

3.1. 状态图

3.2. 活动图

3.3. 交互图

3.3.1. 序列图

3.3.2. 协同图

4. 实现图

4.1. 组件图

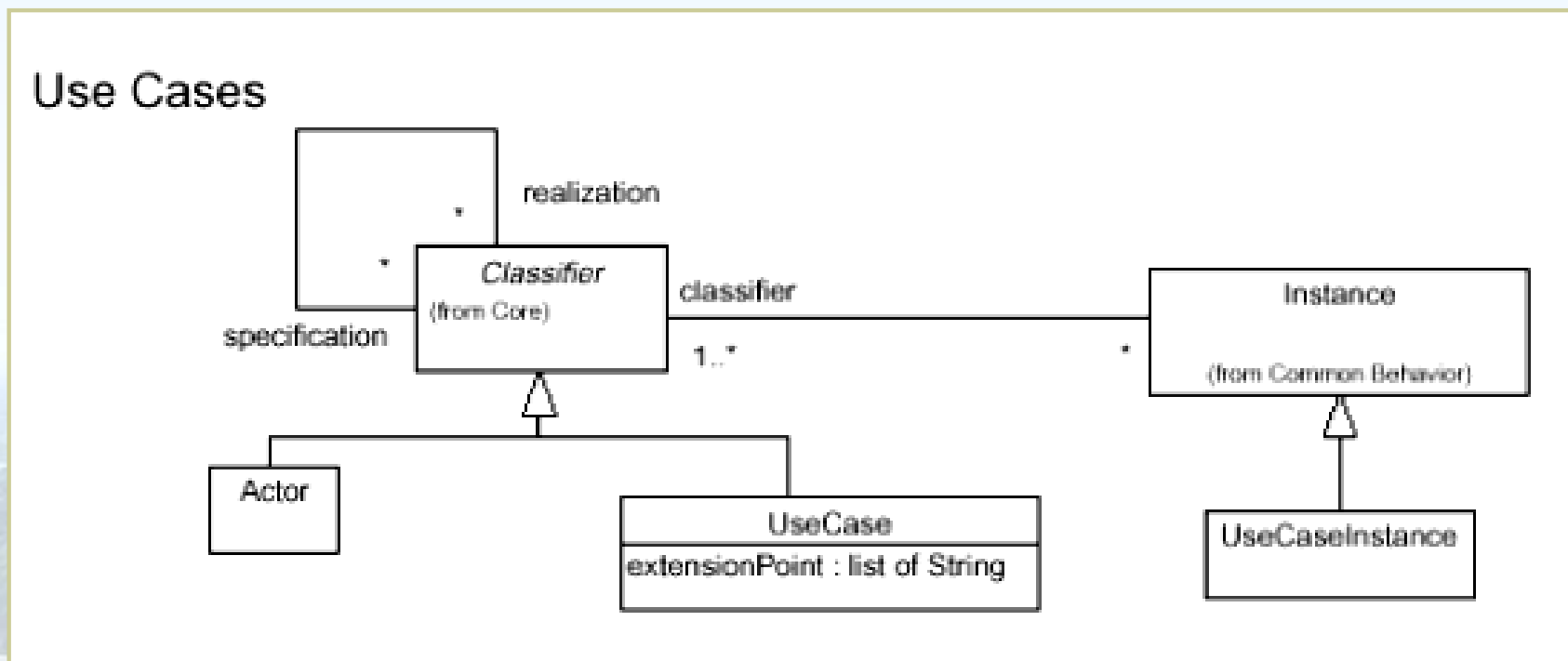
4.2. 部署图



## 4.6 UML的基本概念、结构 and 作用

### 4.6.3 UML的基本概念与结构

- UML 采用了先进的开放式语言设计理念，对语言的扩充提供了有效的支持：UML 元模型（*meta-model*），即：用 UML 本身来表示 UML。





## 4.6 UML的基本概念、结构 and 作用

### 4.6.3 UML的基本概念与结构

- UML 支持元模型扩充的手段：
  - **stereotypes**: 用于扩充 UML 元模型中的类，允许用户增加新的、有特定语义的表示符号；
  - **tagged values**: 用于扩充 UML 元模型中类的属性；
  - **constraints**: 用于扩充 UML 元模型的语义。
- 例如，**Rational Rose** 支持元模型扩充的接口是：
  - **REI** (*Rose Extensibility Interface*): 用 **Rose Scripts** 编写，或用 **C++** 编写、通过 **Rose Automation** 加入需要扩充的表示。





## 4.6 UML的基本概念、结构 and 作用

### 4.6.3 UML的基本概念与结构

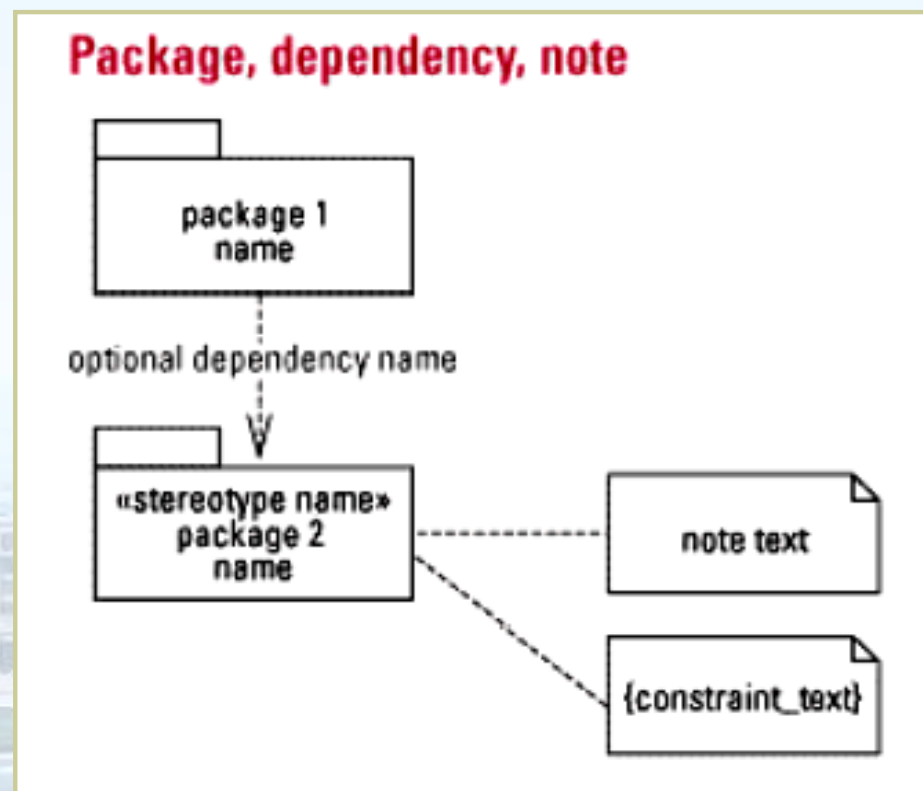
- UML 支持的“4+1”视图的覆盖面，比传统面向对象方法（如 Coad 方法）涉及的要广。后者主要涉及“4+1”视图中的逻辑视图：信息模型对应于类图，功能模型对应于活动图，行为模型对应于用例图。
- UML 的各种图之间保持一致，并且在相应的软件开发环境中都可以自动生成对应的文档，文档的模板可以定制。
- 用类图自动生成程序框架很实用，可以进行逆向工程。



## 4.7 UML基本表示

### 4.7.1 包 (Package)、依赖 (Dependency)、注释 (Note)

- package可以出现在多种图中，用来表示集合抽象。
- dependency表示图中实体之间的依赖：A指向B的依赖表示对B的修改可能导致A的修改。
- note是对图中实体的附加信息说明。





## 4.7 UML基本表示

### 4.7.2 用例图 (Use-Case Diagram)

- actor表示要与本系统发生交互的一个角色单元（人或其他系统）。
- use-case表示由本系统提供的一个业务功能单元。

#### USE-CASE DIAGRAM

Shows the system's use cases and which actors interact with them

#### Actor, use case, and association







## 4.7 UML基本表示

### 4.7.2 用例图

- Example: ATM withdrawal – scenario
  1. enter ATM card
  2. enter PIN number
  3. system verifies that PIN is correct
  4. system asks “show balance” or “withdrawal”
  5. customers selects “withdrawal”
  6. system asks amount
  7. customers enters amount
  8. system verifies amount  $\leq$  available balance
  9. system dispenses money
  10. system asks if receipt is required
  11. customers requests receipt
  12. system prints receipt
  13. systems returns ATM card



## 4.7 UML基本表示

### 4.7.2 用例图

- Variant: PIN incorrect
  - At step 2, the system determines the PIN is incorrect
  - 2b. ask user to re-enter PIN
  - 3b. return to primary scenario at step 3
- Variant: user requests account balance
  - At step 5, user selects “show balance”
  - 5c. system displays account balance
  - 6c. system asks if other services are required
  - 7c. customers confirms
  - 8c. return to primary scenario at step 4



## 4.7 UML基本表示

### 4.7.2 用例图

- Actors
  - the customer withdrawing money
  - the database system that contains account information
- Use cases
  - Enter PIN
  - Verify PIN
  - Select Service (withdrawal or balance)
  - Enter amount
  - Check account balance
  - Print receipt
  - Dispense money





## 4.7 UML基本表示

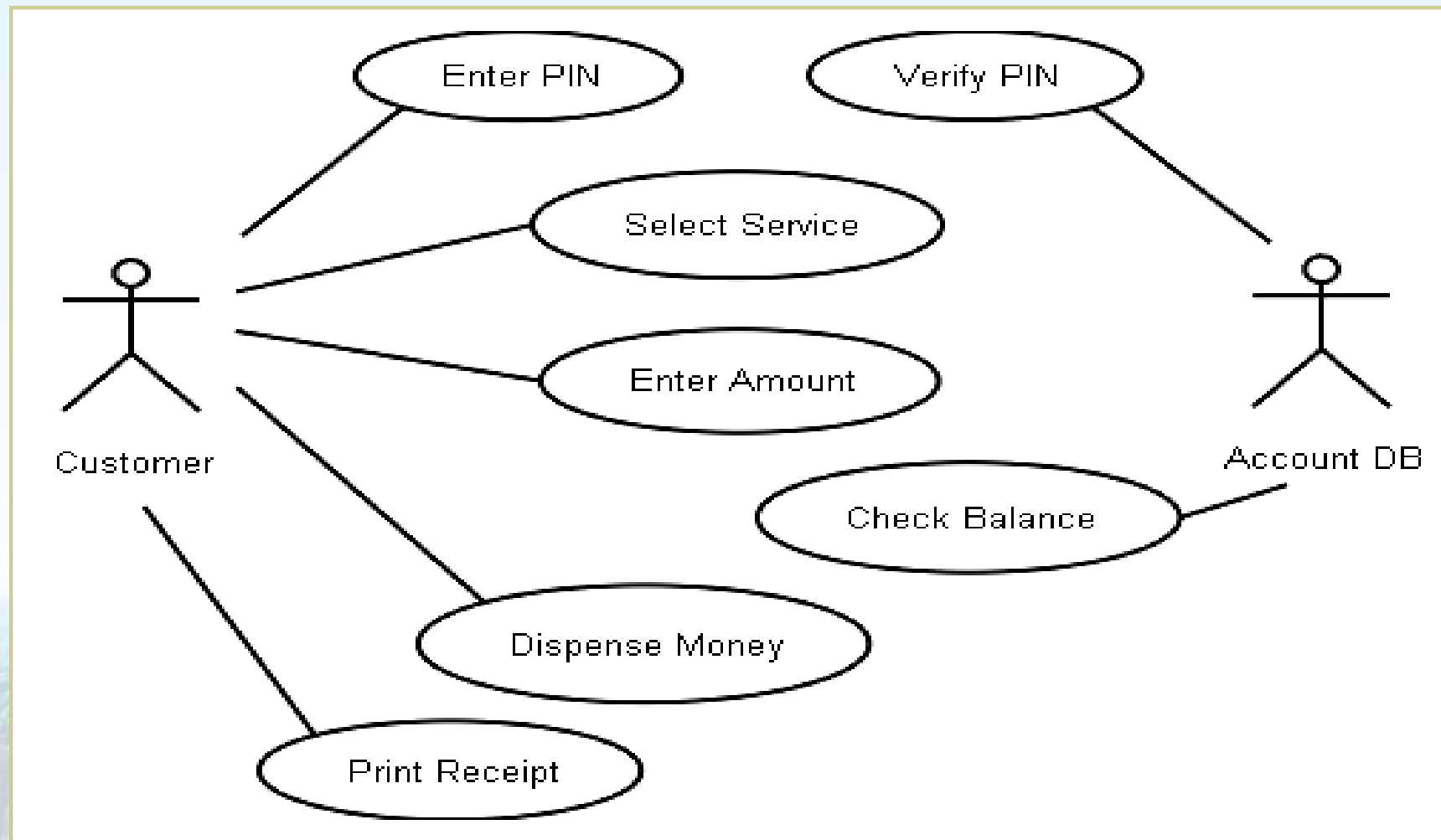
### 4.7.2 用例图

- Example: ATM withdrawal – scenario
    - 1. enter ATM card
    - 2. enter PIN number
    - 3. system verifies that PIN is correct
    - 4. system asks “show balance” or “withdrawal”
    - 5. customers selects “withdrawal”
    - 6. system asks amount
    - 7. customers enters amount
    - 8. system verifies amount  $\leq$  available balance
    - 9. system dispenses money
    - 10. system asks if receipt is required
    - 11. customers requests receipt
    - 12. system prints receipt
    - 13. systems returns ATM card
- Enter PIN  
— Verify PIN  
} Select Service  
} Enter amount  
— Check account balance  
— Dispense money  
} Print receipt



## 4.7 UML基本表示

### 4.7.2 用例图

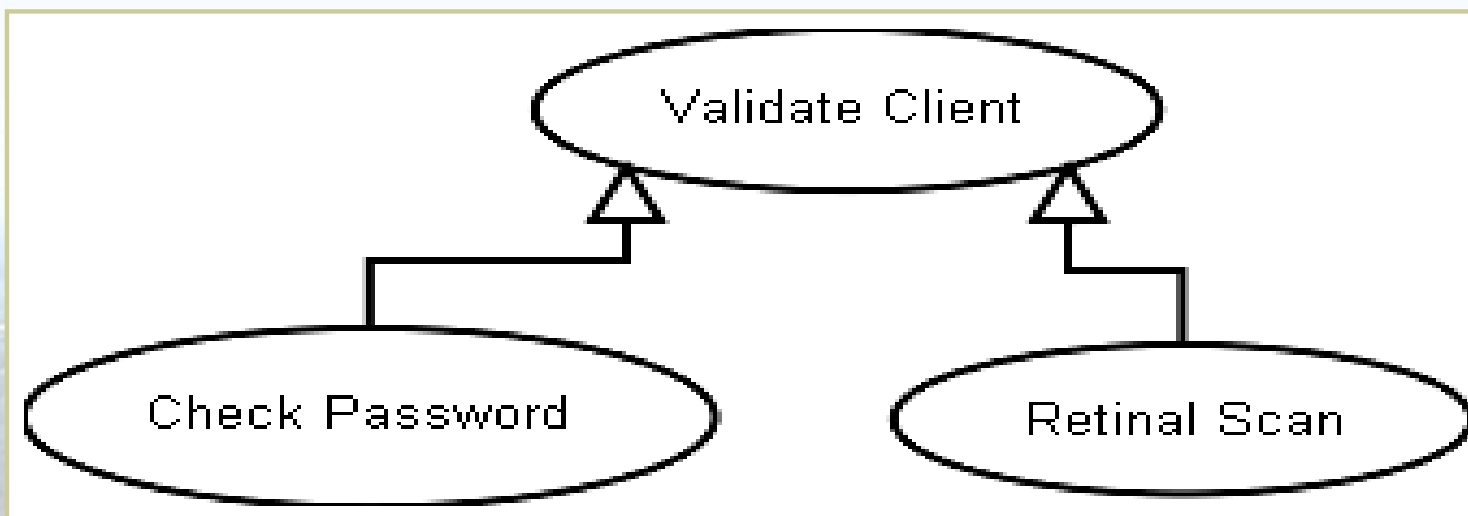




## 4.7 UML基本表示

### 4.7.2 用例图

- Generalization:
  - child use case inherits behavior and meaning from parent use case
  - child may add or override parent behavior
  - child may be substituted where parent occurs



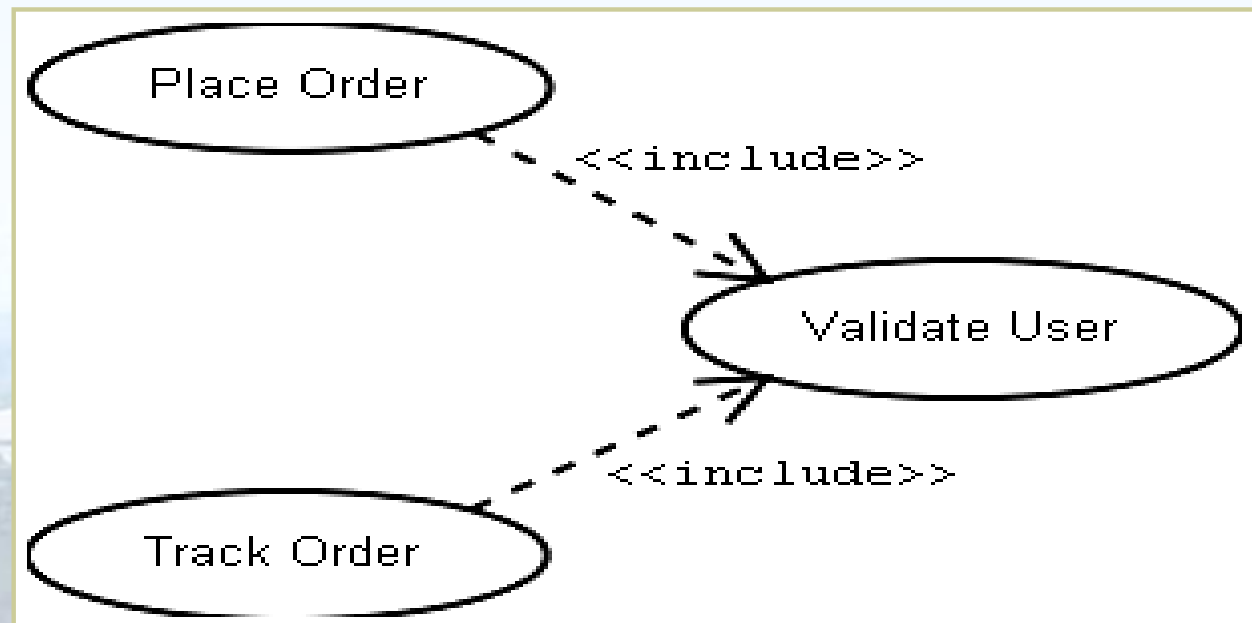




## 4.7 UML基本表示

### 4.7.2 用例图

- Include:
  - avoid duplication of the same flow of events by putting common behavior in a use case of its own use to avoid copy & paste in use case descriptions

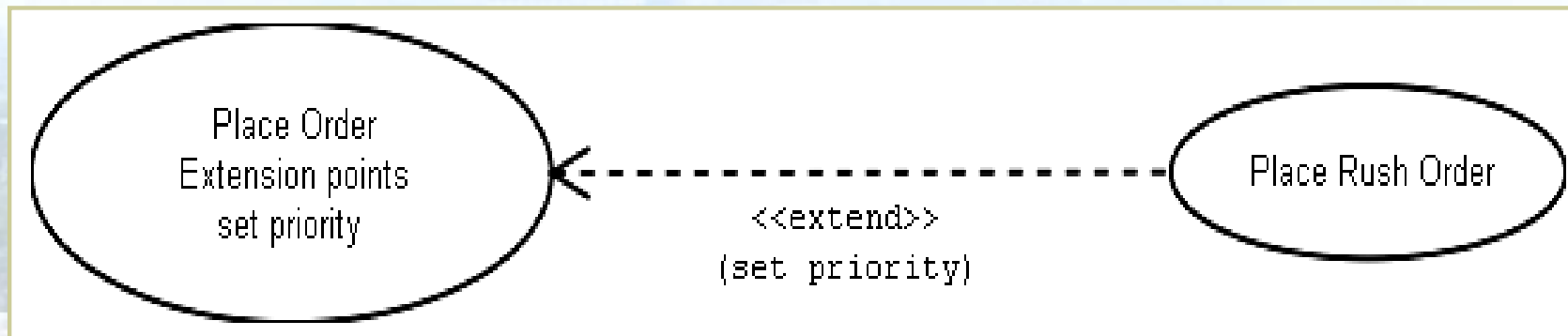




## 4.7 UML基本表示

### 4.7.2 用例图

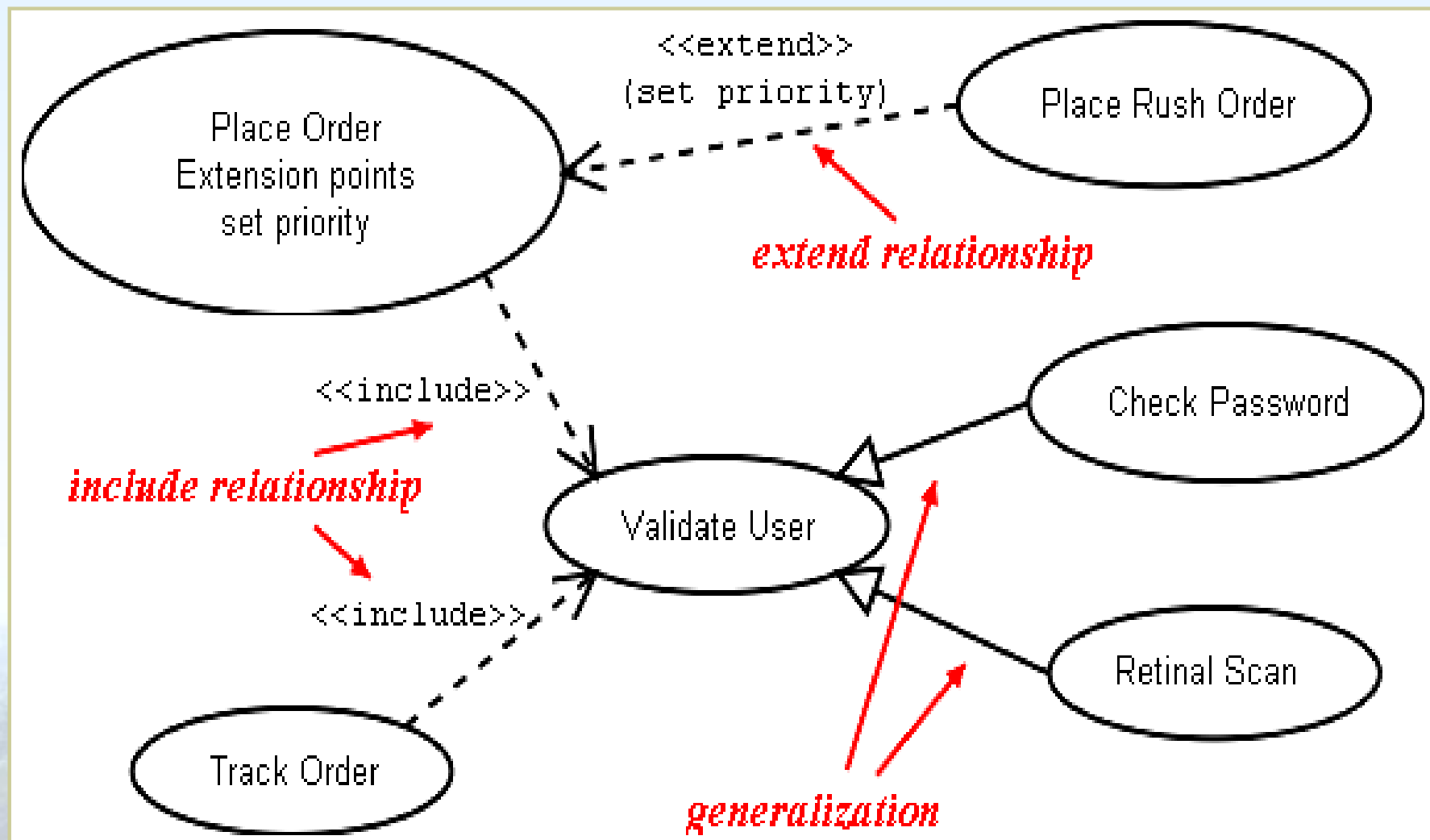
- Extend :
  - similar to generalization, but more restricted
  - the extending use case may add behavior to the base use case, but:
    - the base use case must declare certain extension points
    - the extending use case may add additional behavior only at those extension points
  - primary use: modeling optional system behavior





## 4.7 UML基本表示

### 4.7.2 用例图





## 4.8 UML 静态建模表示—类图

### 4.8.1 类 (Class)

- 类是类图中的基本表示元素之一;
- 类的完整表示由类名、属性集和操作集组成;
- 属性集和操作集都可以在图中不出现;
- 类的简略表示只有类名。

**Class**

Class Name

Class Name

**attribute**

**attribute : data\_type**

**attribute : data\_type = init\_value**

...

**operation**

**operation (arg\_list) : result\_type**

...





## 4.8 UML 静态建模表示—类图

### 4.8.1 类

- 属性和操作都有能见度;
- **public**、**protected**、**private** 的语义与 C++ 类似;
- **implementation** 指只允许包含了这个类的包（通常对应于一个文件）中的实现访问（规定了其作用域在该文件内）。

#### Visibility and properties

Class
<ul style="list-style-type: none"><li>- private attribute</li><li># protected attribute</li><li>/- private derived attribute</li><li>+\$class public attribute</li></ul>
<ul style="list-style-type: none"><li>+ public operation</li><li># protected operation</li><li>- private operation</li><li>+\$class public operation</li></ul>

#### Optional visibility icons

##### Attributes

- public
- protected
- private
- implementation

##### Operations

- public
- protected
- private
- implementation



## 4.8 UML 静态建模表示—类图

### 4.8.1 类

- Perspectives (视点) on Classes:
  - Conceptual (domain analysis)
    - shows concepts of the domain
    - implementation-independent
  - Specification (design)
    - general structure of the system
    - used in high-level design
  - Implementation (programming)
    - structure of the implementation
    - code (class' definition and operations' frame) generation
    - most often used
- Always try to draw from a single perspective!

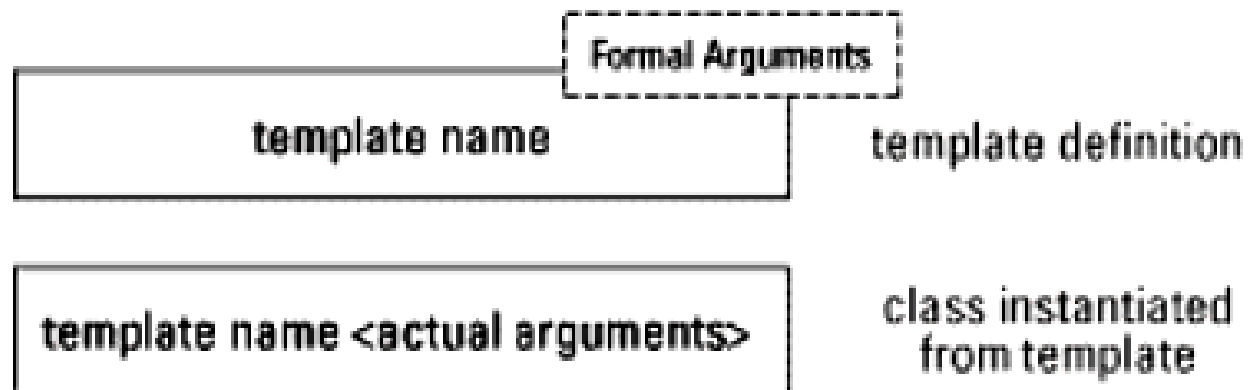


## 4.8 UML 静态建模表示—类图

### 4.8.2 模板 (Template)

- 语义与 C++ 中的 `template` 结构类似;
- 表示上分为模板定义和实例化类两部分。

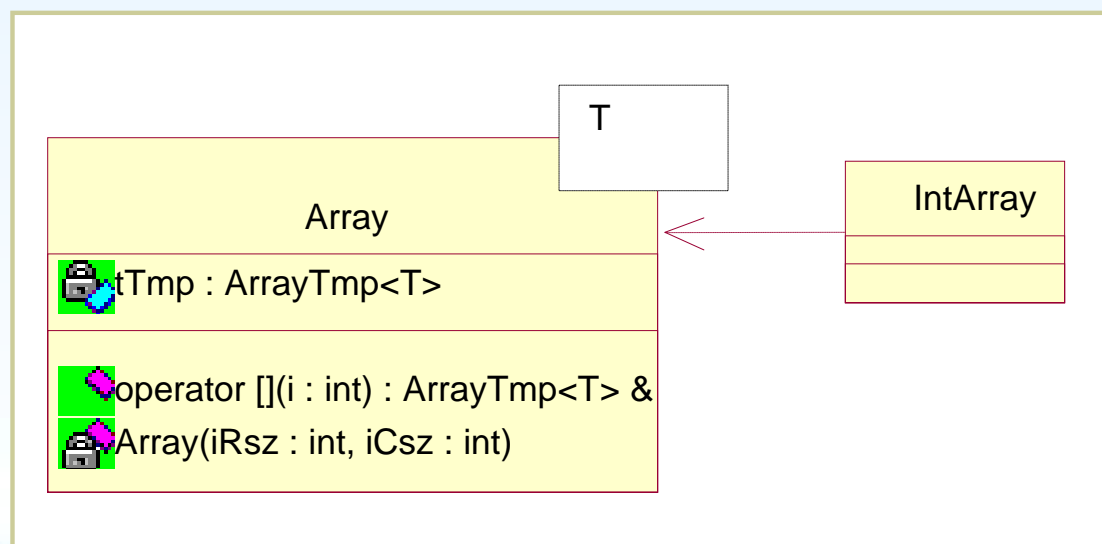
#### Parameterized class





## 4.8 UML 静态建模表示—类图

### 4.8.2 模板



类别声明

Class Specification for IntArray

Relations	Components	Nested	Files
General	Detail	Operations	Attributes

Name: IntArray Parent: Logical View

Type: InstantiatedClass

Stereotype:

Class Specification for IntArray

Relations	Components	Nested	Files
General	Detail	Operations	Attributes

Multiplic: n

Space:

Persistence: ☐ Persistent ☒ Transient

Concurrency: ☒ Sequential ☐ Guarded ☐ Active ☐ Synchronous

☐ Abstract

Actual Arguments:

Name	Type	Default Value
int	class	

OK Cancel Apply Browse Help

类型实参

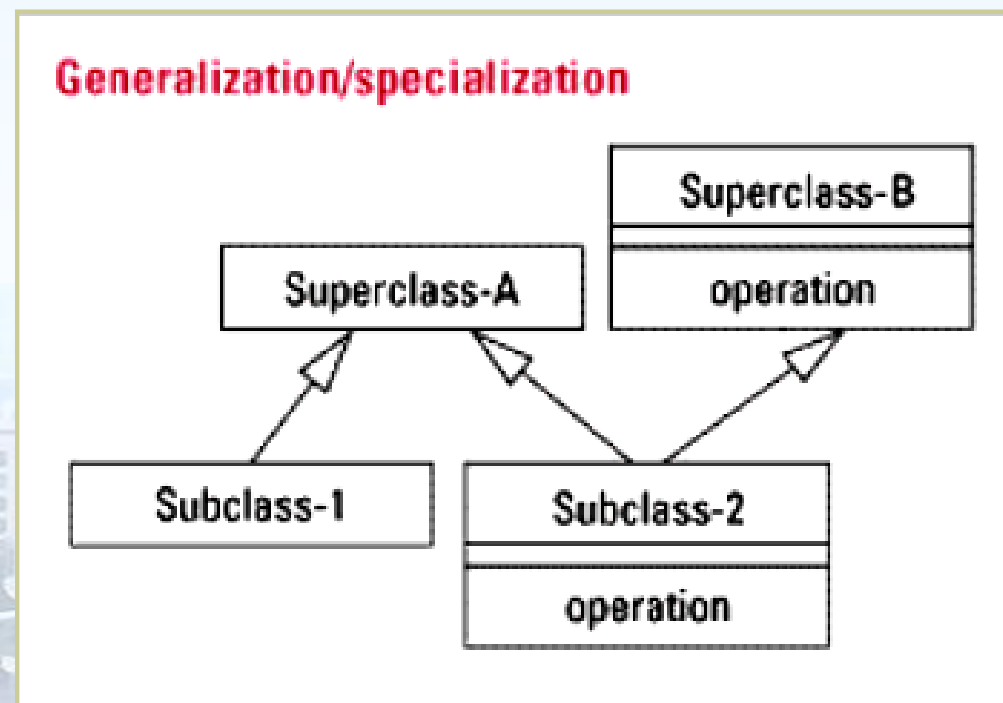




## 4.8 UML 静态建模表示—类图

### 4.8.3 概括 (Generalization)

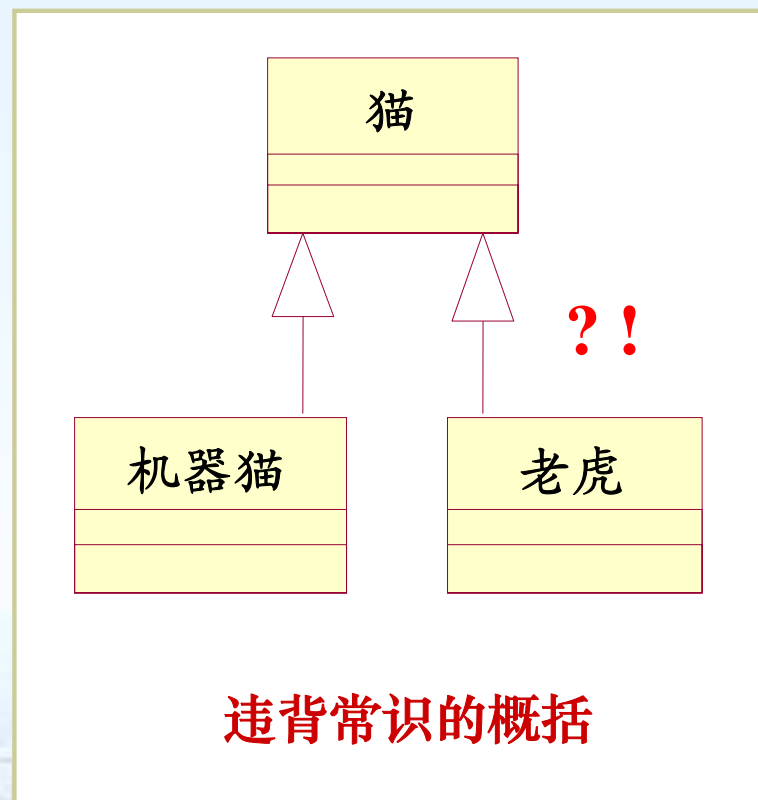
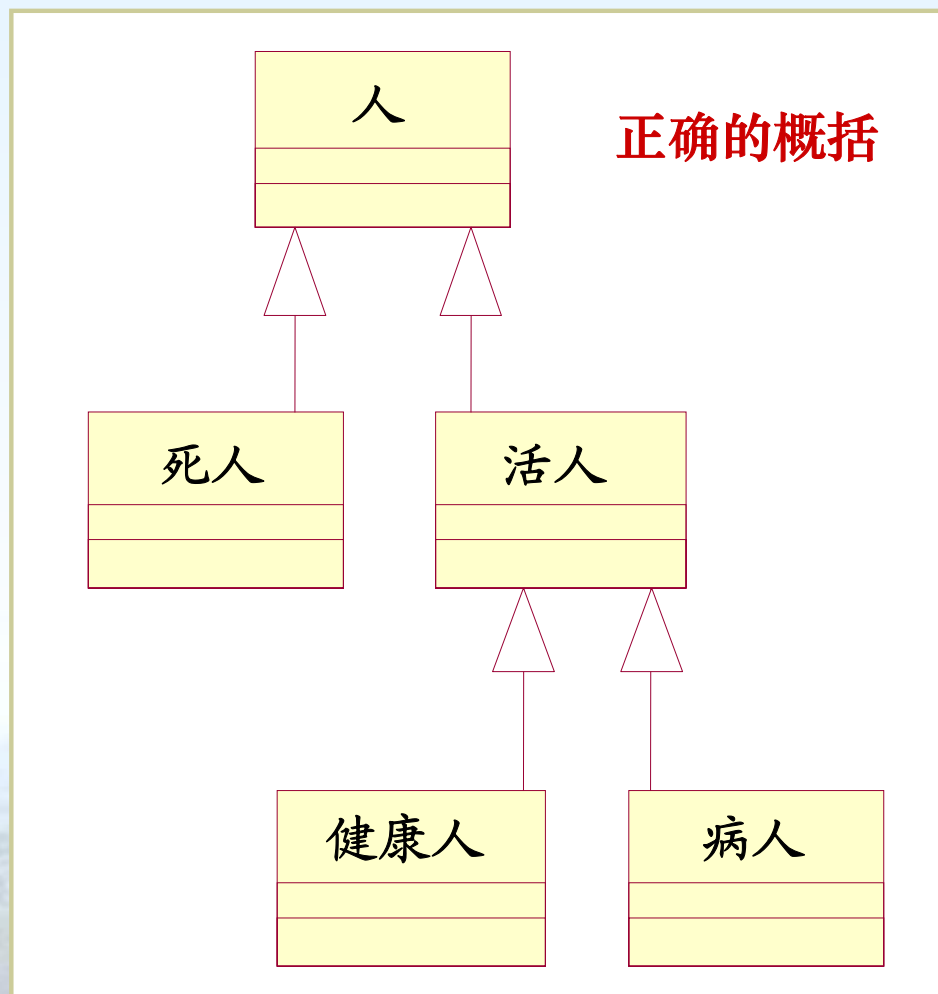
- 语义：当A概括 B，则B除具有A的全部特性外，还可定义新的特性，以及置换从A继承的特性。
- 概括是传递、非对称的。
- 概括关系的拓扑应是格。
- 可用于类、用例、包等类实体。





## 4.8 UML 静态建模表示—类图

### 4.8.3 概括

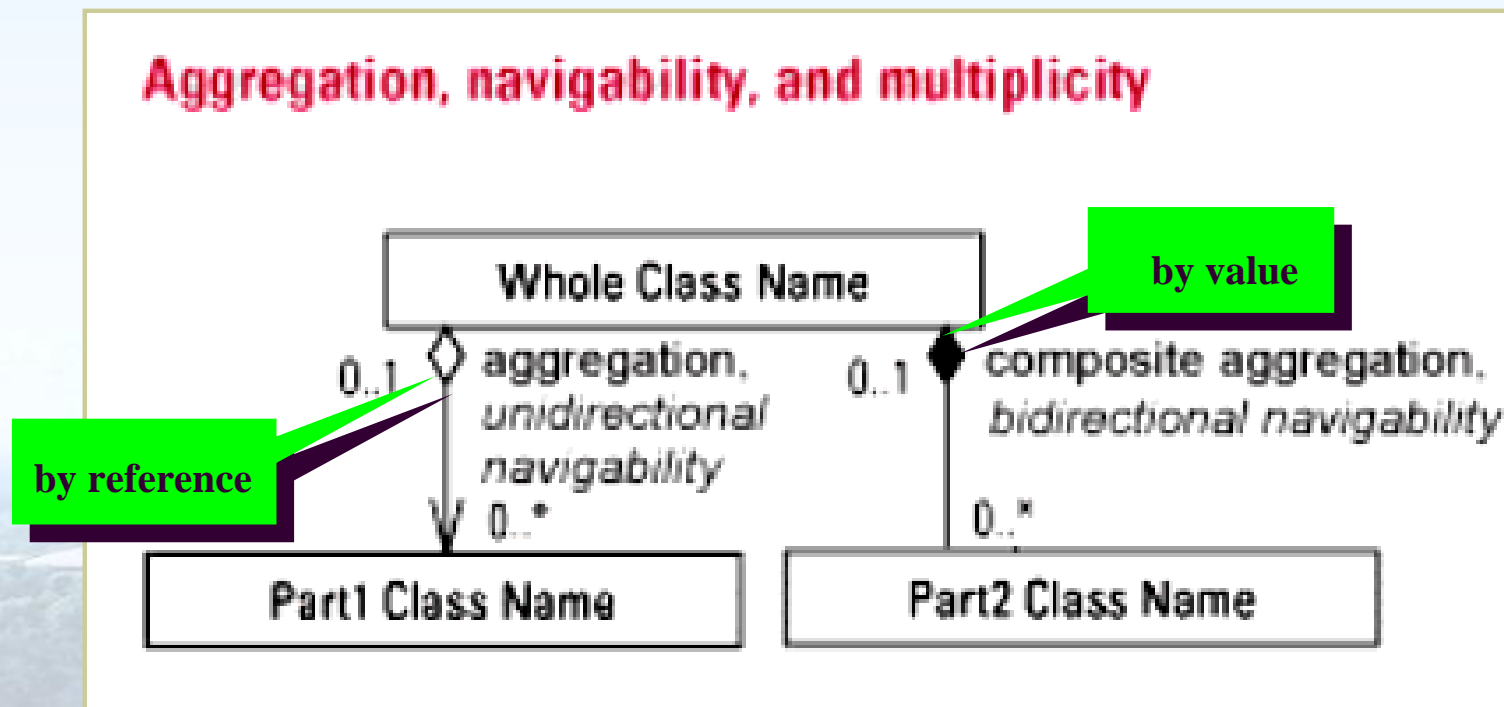




## 4.8 UML 静态建模表示—类图

### 4.8.4 聚集 (Aggregation)

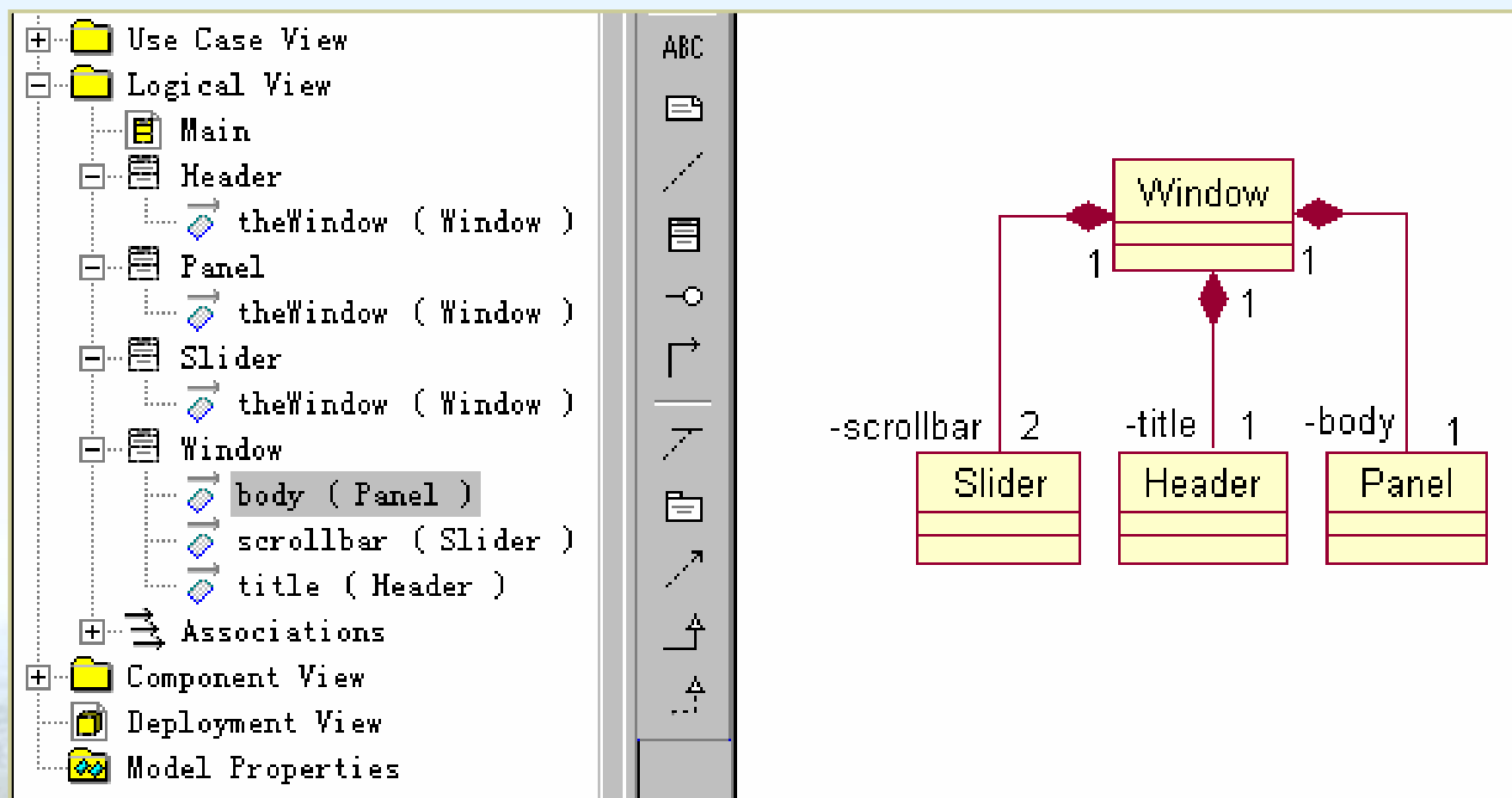
- 语义：当类 A 聚集类 B，则 B 的实例是 A 的实例的子对象。
- 用于表示整体与部分的关系。





## 4.8 UML 静态建模表示—类图

### 4.8.4 聚集

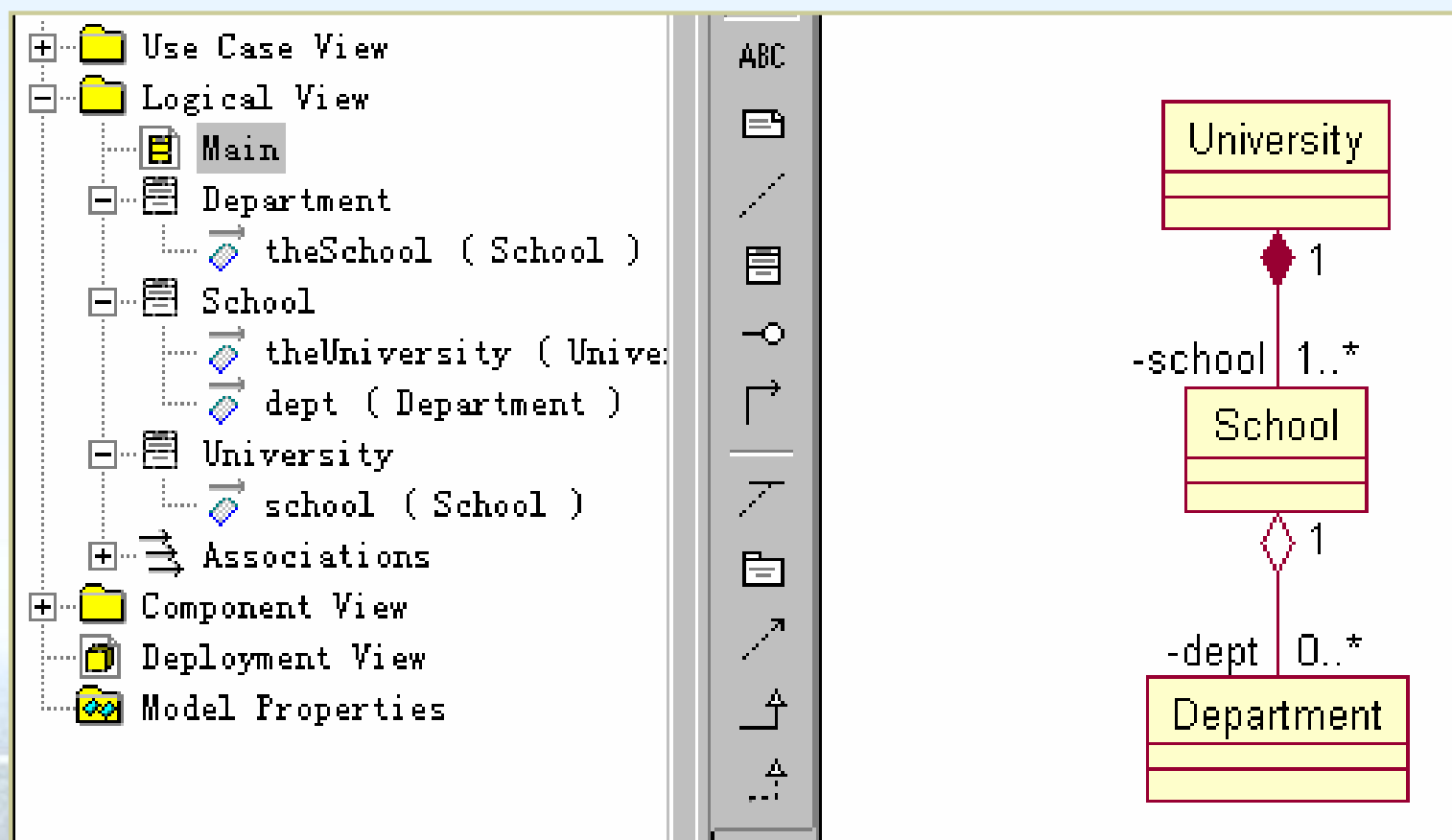






## 4.8 UML 静态建模表示—类图

### 4.8.4 聚集



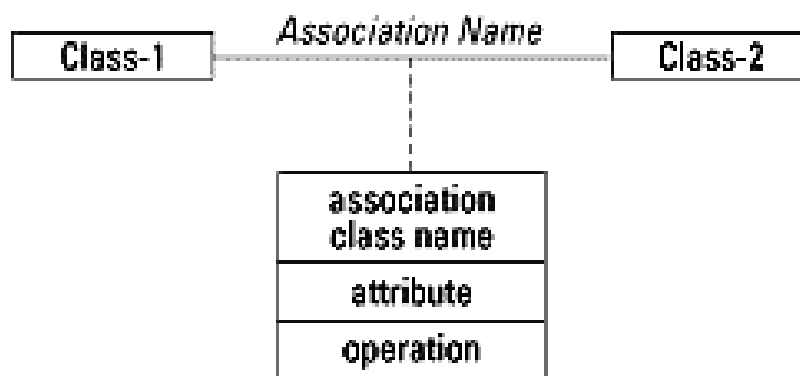


## 4.8 UML 静态建模表示—类图

### 4.8.5 关联 (Association)

- 语义：表示两个可以独立存在的类的实例之间具有关联关系。
- 关联关系可以用属性或另一个类来表现。
- 在需要时，可以将可推导出的关联用导出关联显式表示。

#### Association classes



#### Role names and derived associations

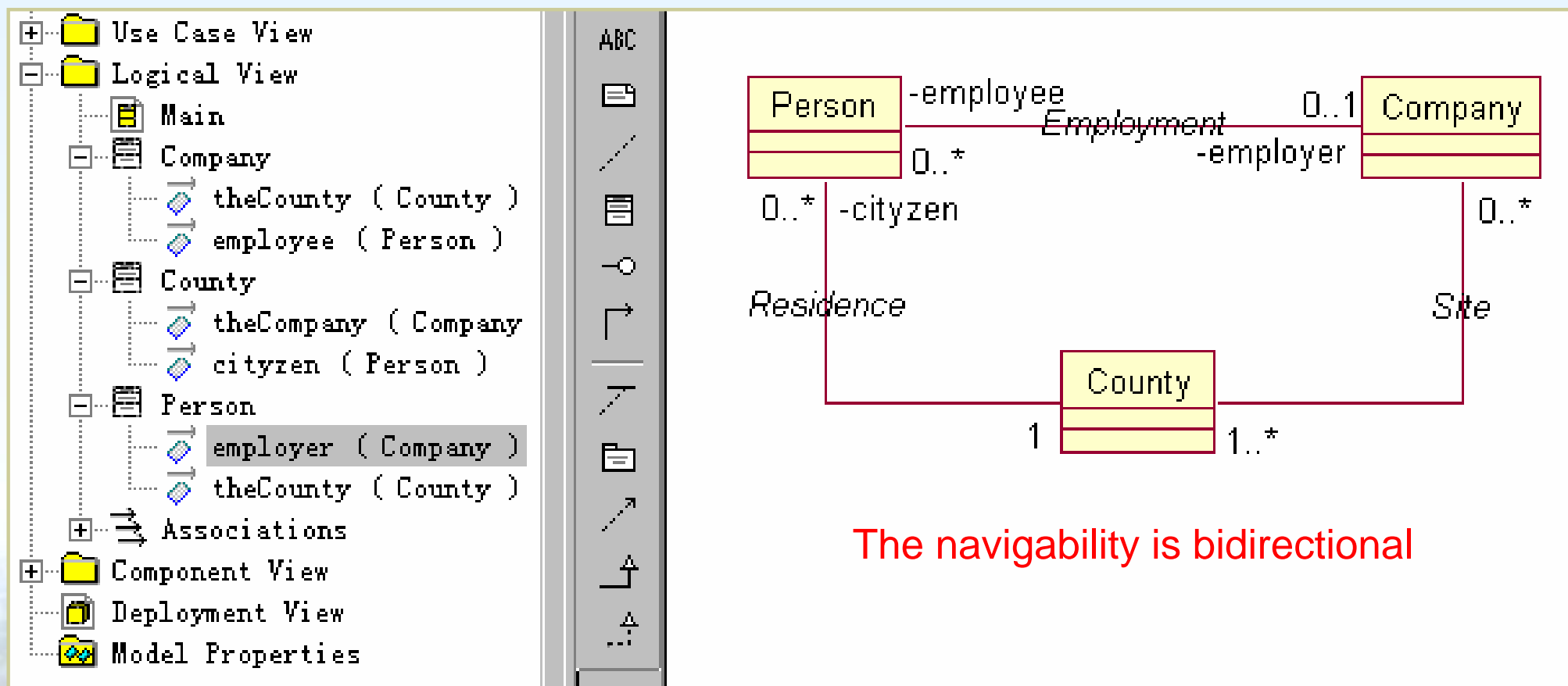
##### Association





## 4.8 UML 静态建模表示—类图

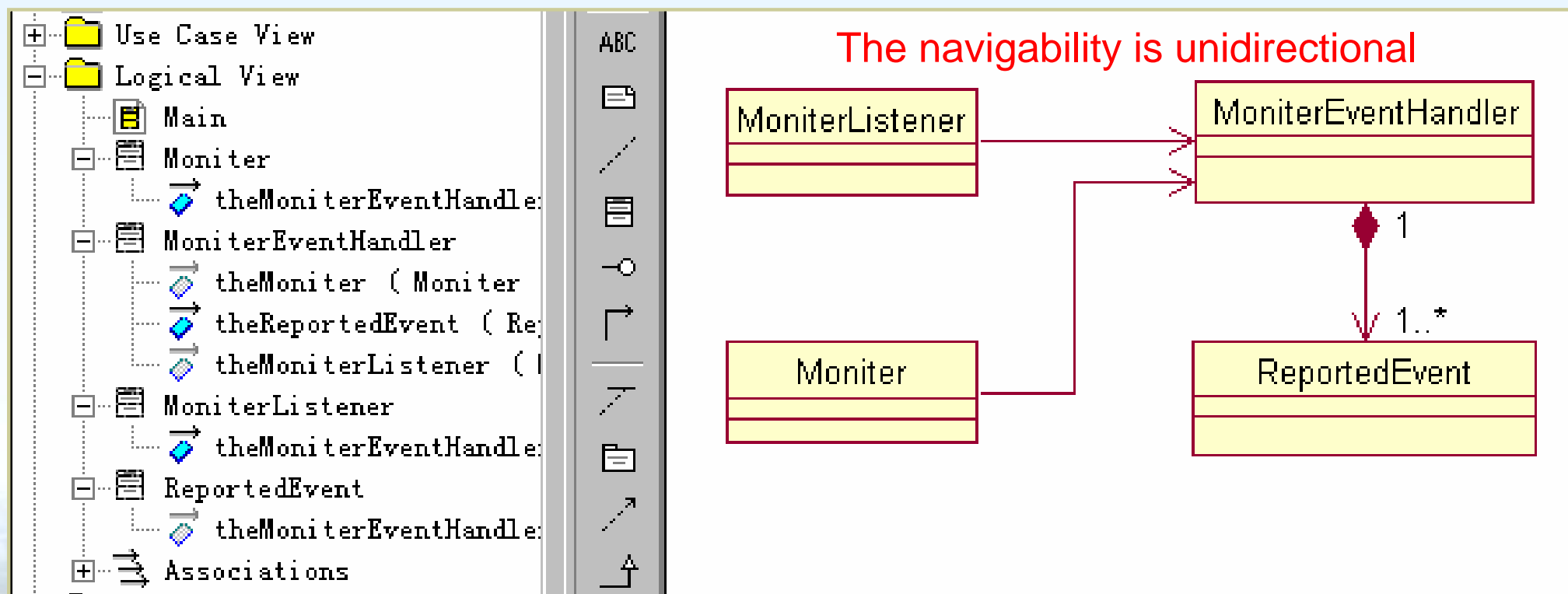
### 4.8.5 关联





## 4.8 UML 静态建模表示—类图

### 4.8.5 关联

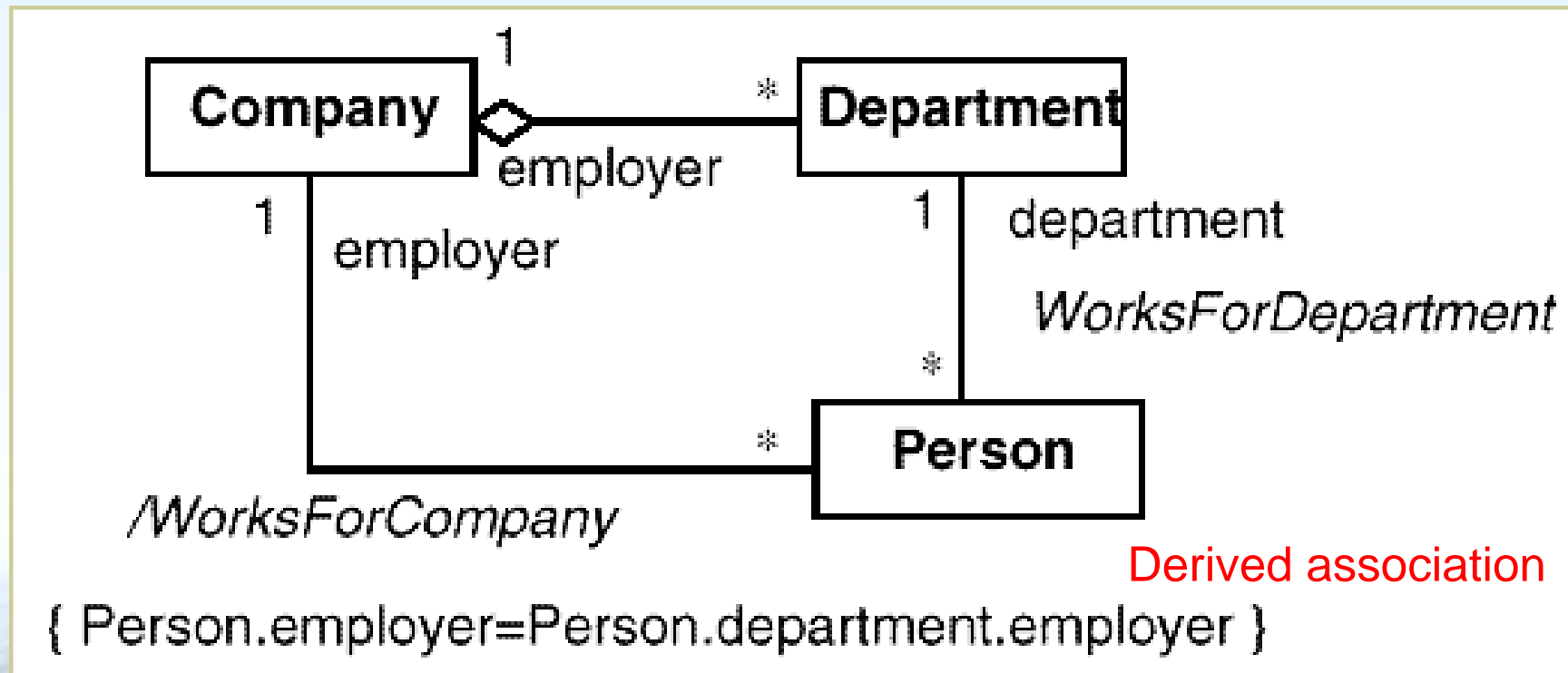






## 4.8 UML 静态建模表示—类图

### 4.8.5 关联





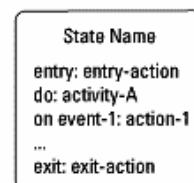
## 4.9 UML动态建模表示

### 4.9.1 状态图 (State-Transition Diagram / Statechart)

- 状态图是表示状态机的图，用来表示模型元素（如对象、交互等）的行为特征。
- 状态图注重描述可能的状态序列，以及在特定状态下模型元素对外部离散事件的响应动作。

**STATE-TRANSITION DIAGRAM** Shows the state space of a given context, the events that cause a transition from one state to another, and the actions that result

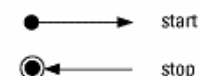
#### State icon



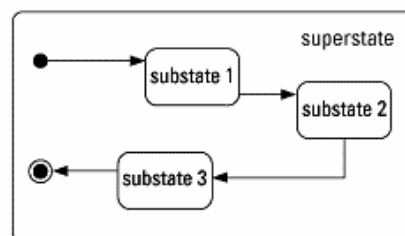
#### History (H)

#### State transitions

event(arguments)[condition]  
/action  
^send target.send event (send arguments)



#### Nesting





## 4.9 UML动态建模表示

### 4.9.1 状态图

- A statechart diagram shows the flow of control from state to state:
  - for **a single object**.
  - shows how the state of the object changes as a result of events that occur.
- Elements:
  - **states**
  - **transitions, guarded transitions**
  - **events**
  - **activities**
- Composite states:
  - a single state consists of a state machine.



## 4.9 UML动态建模表示

### 4.9.1 状态图

- A **state** is a condition or situation in the life on an object during which it satisfies some condition, performs some activity, or waits for some event:
  - typically described by a set of attribute values.
- Examples:
  - the state of a credit card account depends on current balance, payment history, ...
  - the state of an order can be pending (待处理), filled (已执行), onBackOrder (因欠款而未执行), cancelled (取消), delivered (已交货), ...
  - the state of a fax can be Idle, Sending, Receiving.



## 4.9 UML动态建模表示

### 4.9.1 状态图

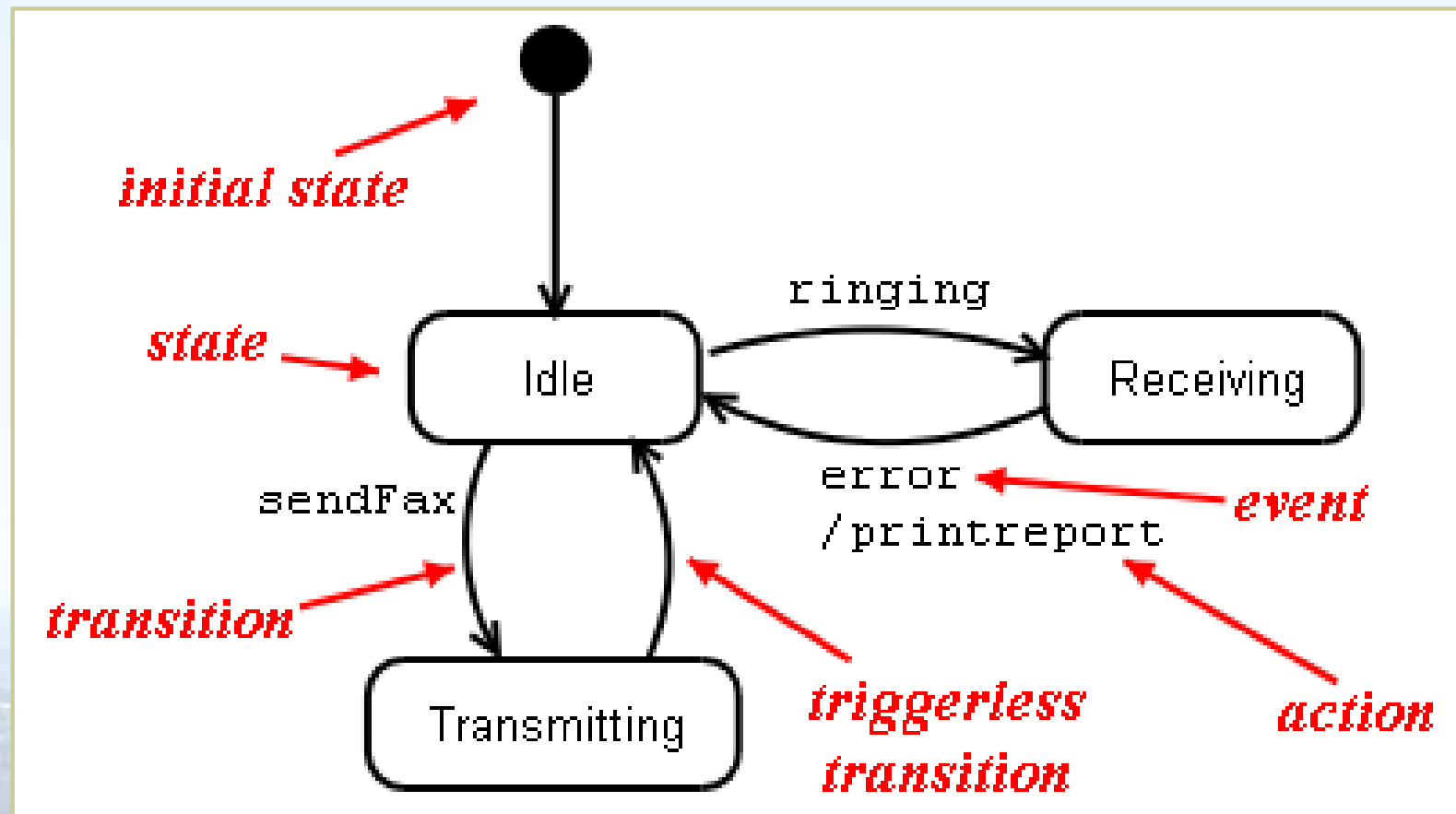
- A **state transition** occurs as a result of an event:
  - State transitions are considered to be atomic (cannot be interrupted).
  - State transitions may be labeled:  
Event [ Guard ] / Action
- An **event** is the specification of a significant occurrence (a message or signal that is received).
- An **action** is associated with a transition:
  - a process that occurs quickly and is not interruptible.
- A **guard** is a logical condition:
  - returns true or false.





## 4.9 UML动态建模表示

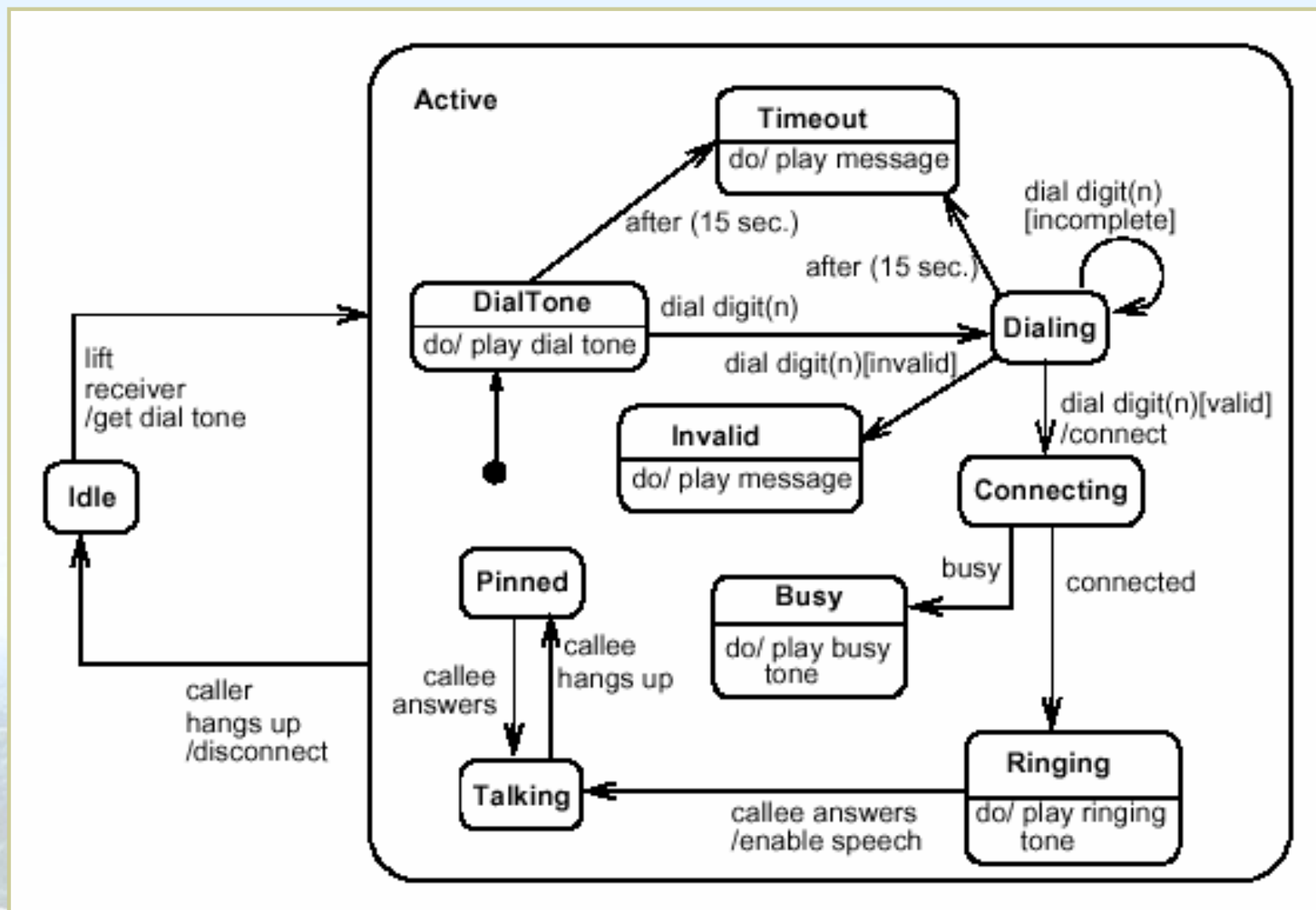
### 4.9.1 状态图





## 4.9 UML动态建模表示

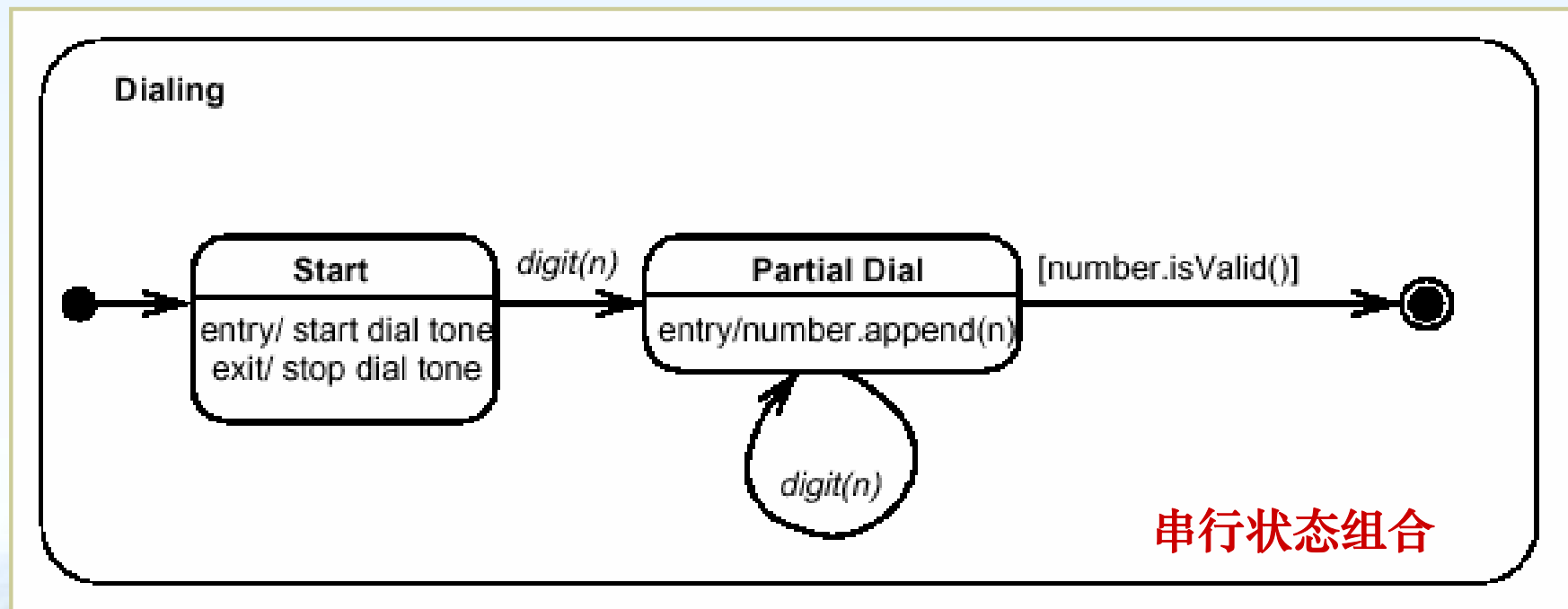
### 4.9.1 状态图





## 4.9 UML动态建模表示

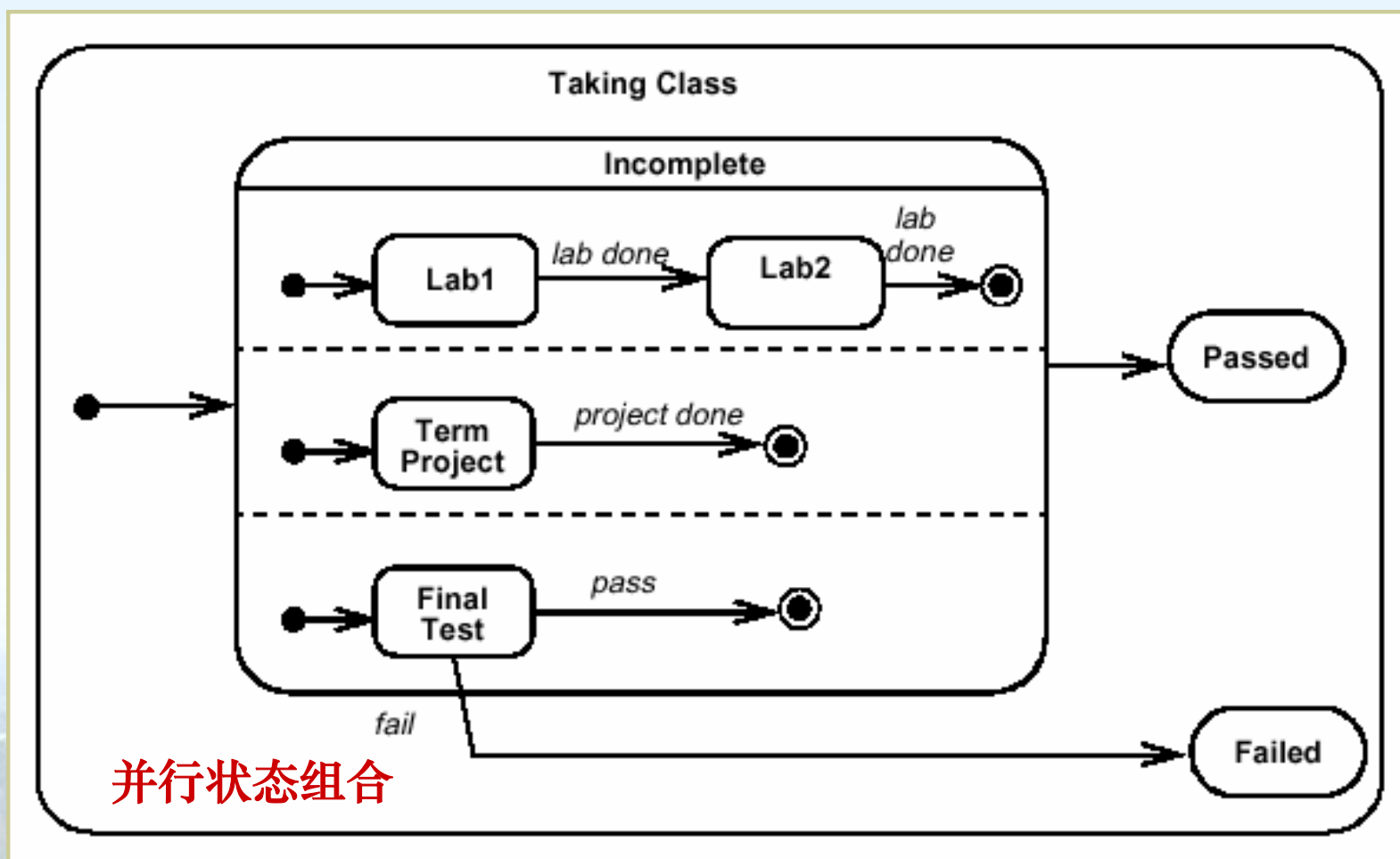
### 4.9.1 状态图





## 4.9 UML动态建模表示

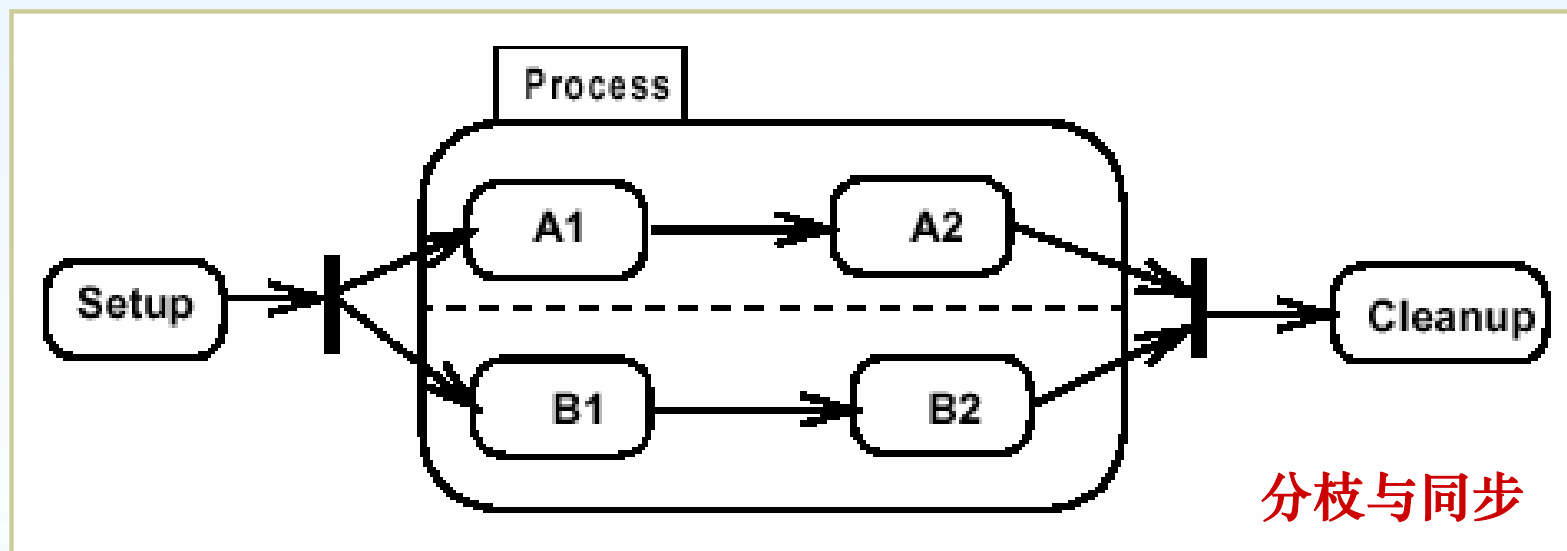
### 4.9.1 状态图





## 4.9 UML动态建模表示

### 4.9.1 状态图







## 4.9 UML动态建模表示

### 4.9.1 状态图

例：现有一种自动售货机，其行为特性如下：

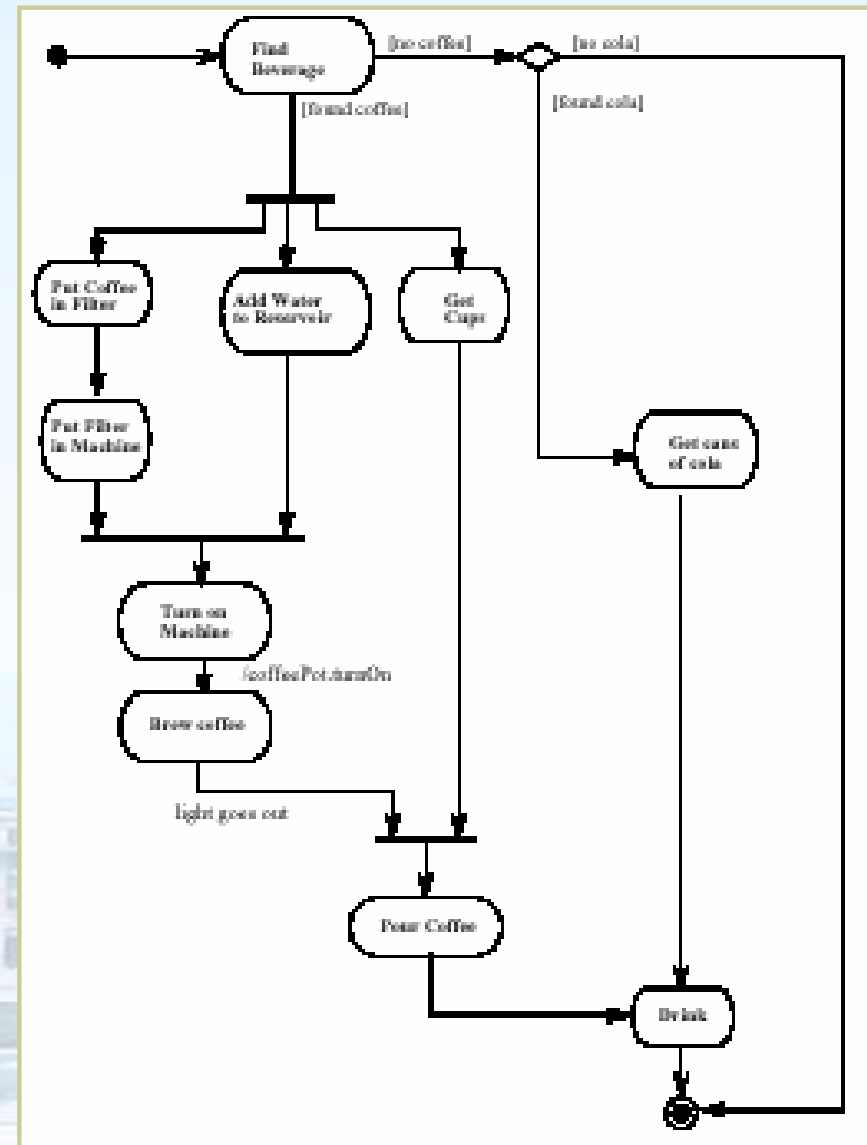
- (1) 只售一种价值 \$0.25 的商品；
- (2) 只接受顾客以任意顺序投入的 \$0.05、\$0.10 和 \$0.25 硬币；
- (3) 顾客投入的硬币总值达到或超过 \$0.25 后，从出货口自动吐出一件商品，并从找币口找零（例如，顾客投入三枚 \$0.10 硬币后，或者依次投入 \$0.05、\$0.05、\$0.10、\$0.10 四枚硬币后，找币口都将吐出一枚 \$0.05 的硬币）；
- (4) 如果顾客投入的硬币总值没有达到 \$0.25 就再也不投币了，在其投入最后一枚硬币30秒后，从找币口吐出与该顾客已投入的硬币总值等值的硬币，但不吐出商品；
- (5) 完成一次（3）或（4）所述行为后，等待下一次投币。



## 4.9 UML动态建模表示

### 4.9.2 活动图 (Activity Diagram)

- 活动图也是一种状态机，但表示的是特定过程的状态机。
- 活动图用来表示内部处理的控制流，类似于程序流程图。





## 4.9 UML动态建模表示

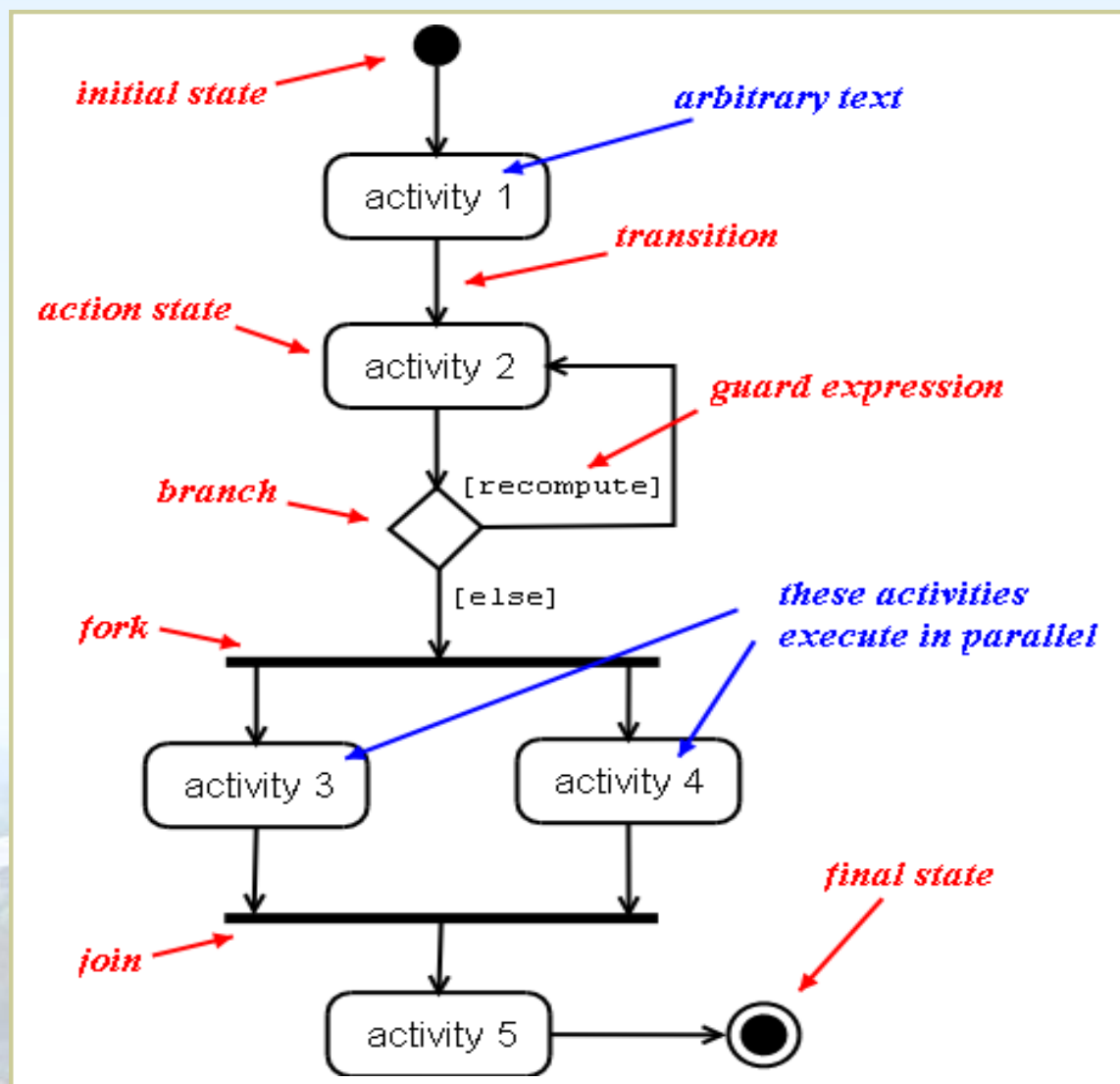
### 4.9.2 活动图

- UML's variation on the well-known flowchart.
- Describes the sequencing of activities, with support for both sequential and parallel behavior.
- **Flow of control** from activity to activity (workflow):
  - start in an initial state.
  - transition via action states to a final state.
- Use free-format text to describe activities.
- Unlike other diagrams, the main emphasis is not on classes and / or objects:
  - but it is possible to specify object flow.



## 4.9 UML动态建模表示

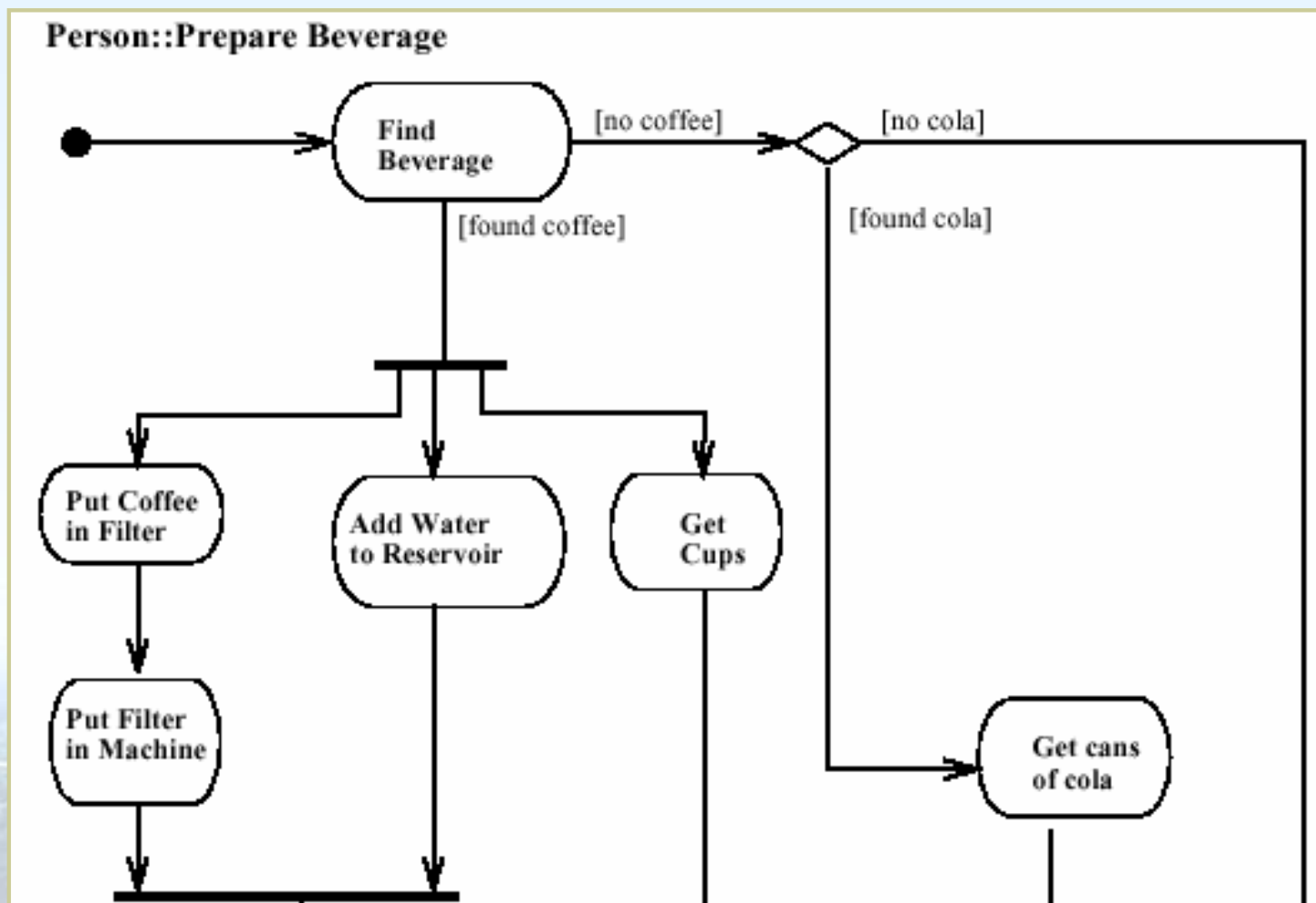
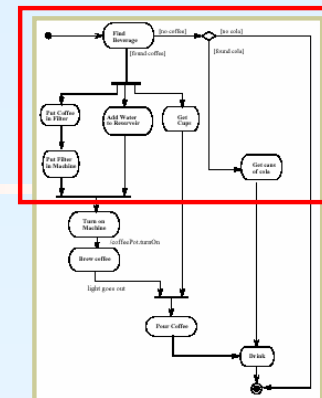
### 4.9.2 活动图





## 4.9 UML动态建模表示

### 4.9.2 活动图







## 4.9 UML动态建模表示

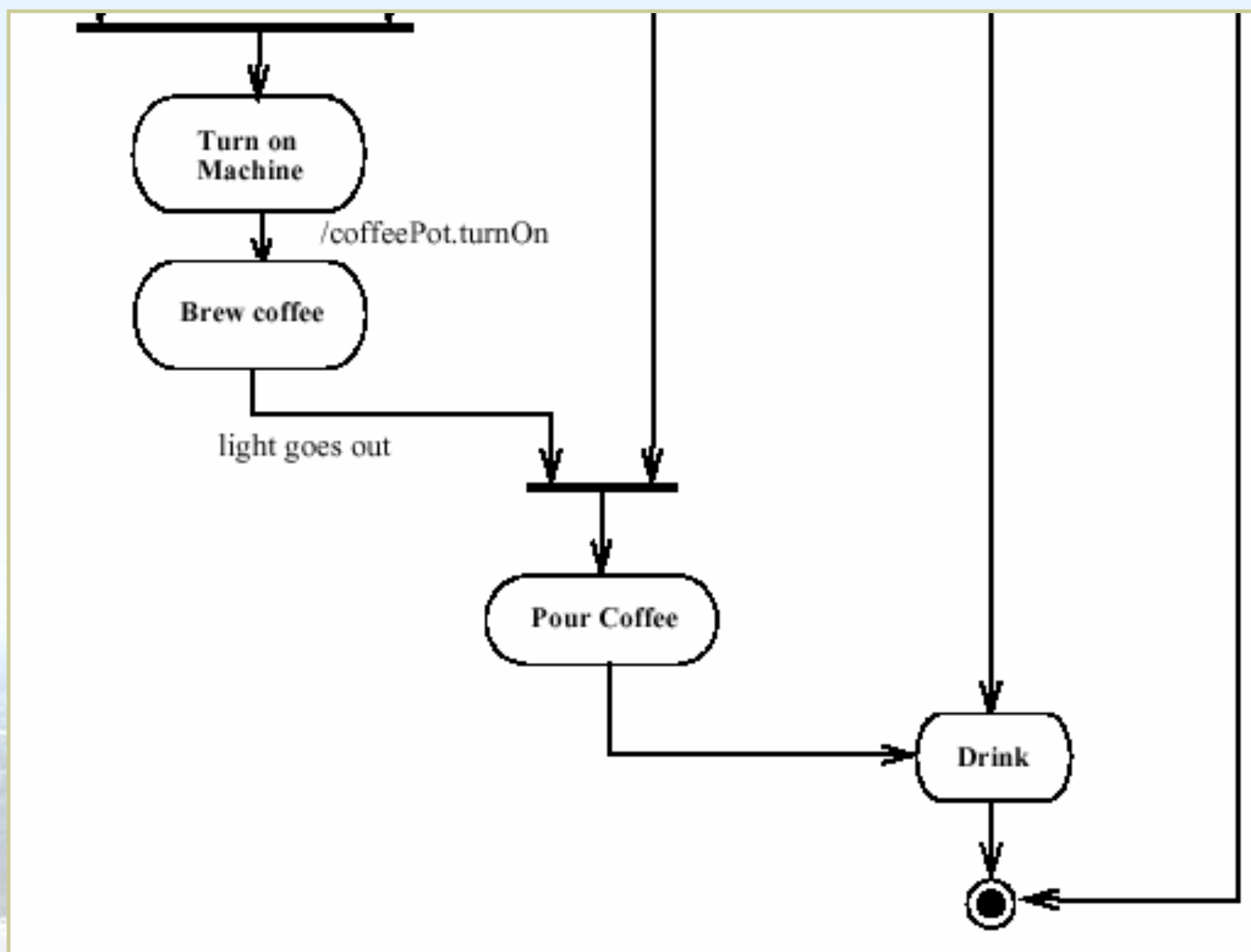
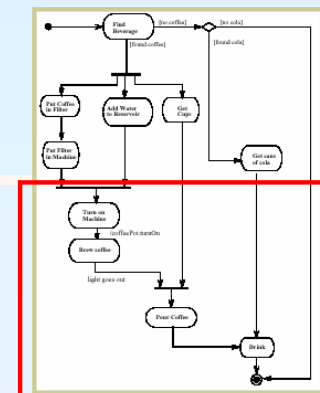
### 4.9.2 活动图





## 4.9 UML动态建模表示

### 4.9.2 活动图





## 4.9 UML动态建模表示

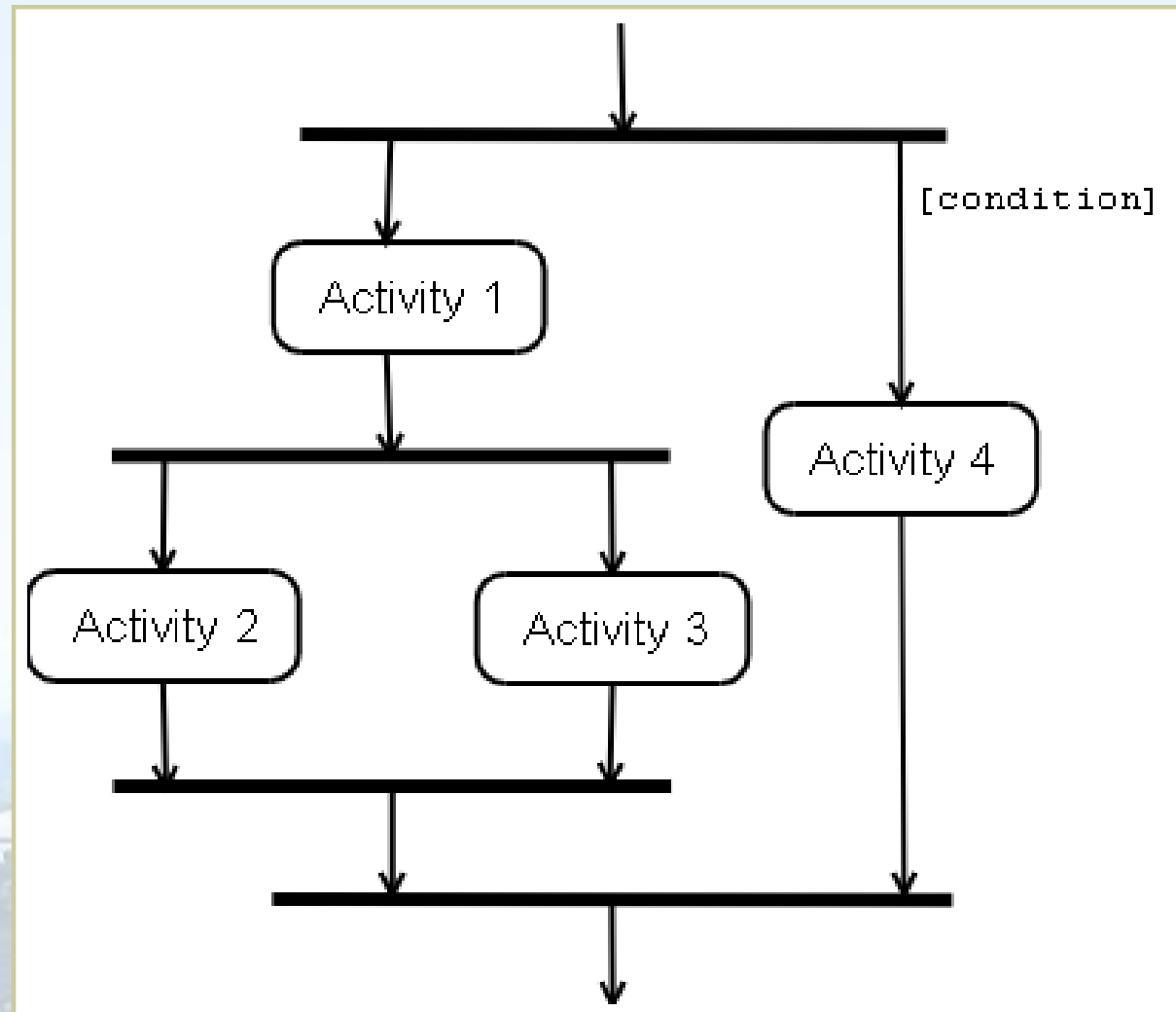
### 4.9.2 活动图

- Fork and joins:
  - Simple case: all **concurrent branches** (threads) must have finished before we transition out of the join.
  - Threads leaving a fork may be traversed in any order (includes interleavings).
  - Forks and joins may be nested.
  - Conditional threads out of forks only need to be completed if their condition holds.



## 4.9 UML动态建模表示

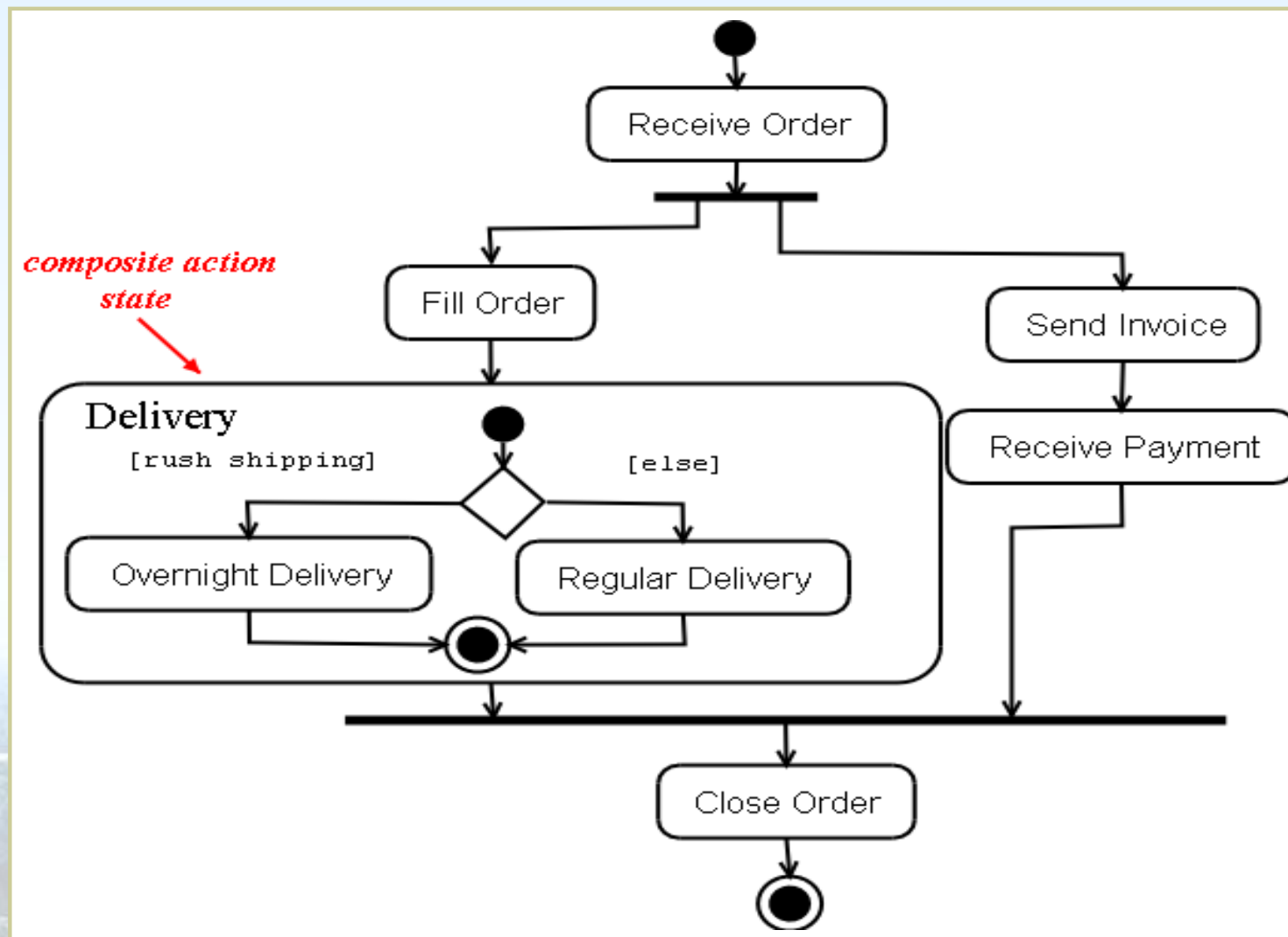
### 4.9.2 活动图





## 4.9 UML动态建模表示

### 4.9.2 活动图



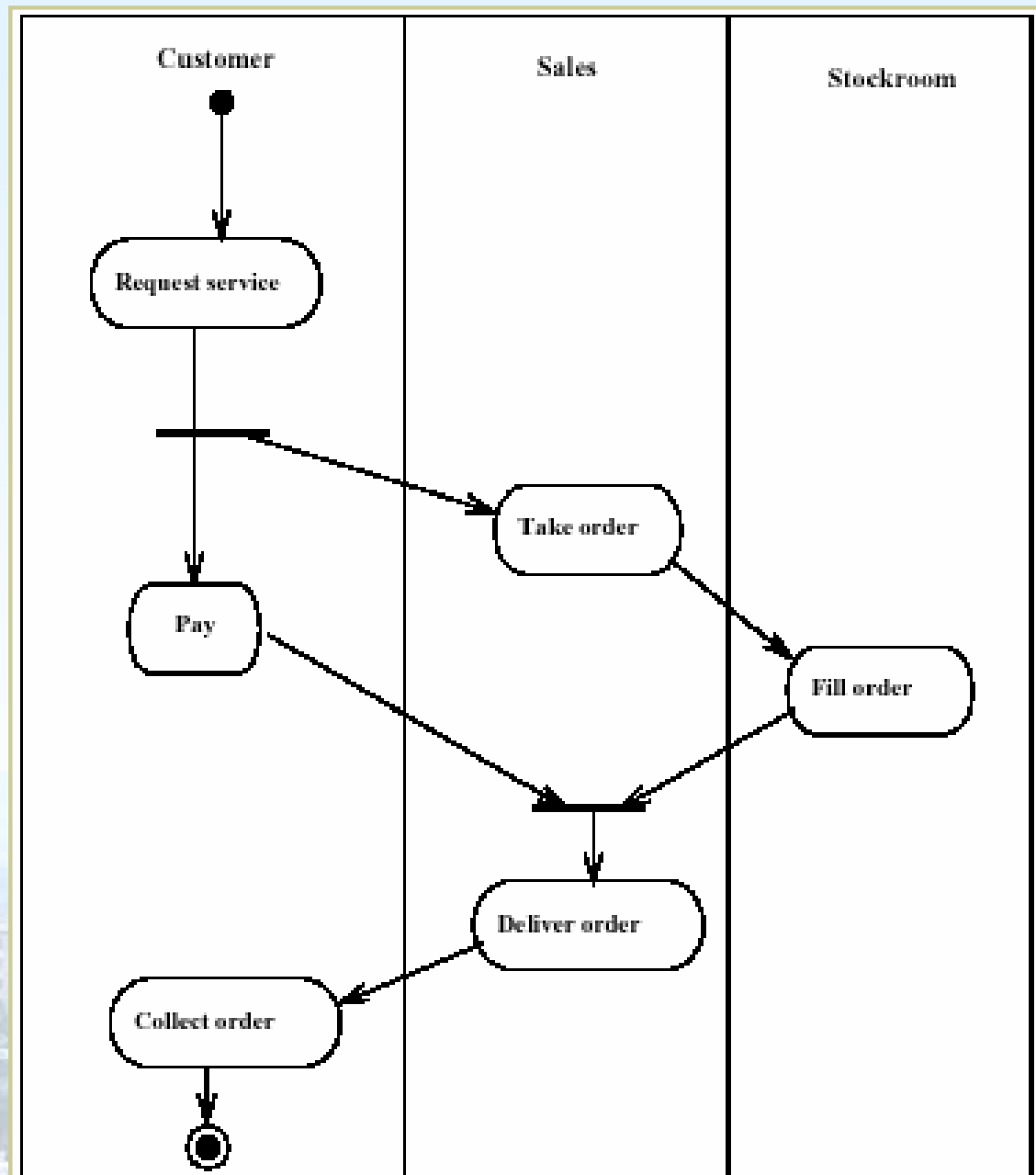




## 4.9 UML动态建模表示

### 4.9.2 活动图

- 依据类的划分，对活动或子活动的描述进行组织，通常可用于描述业务模型。

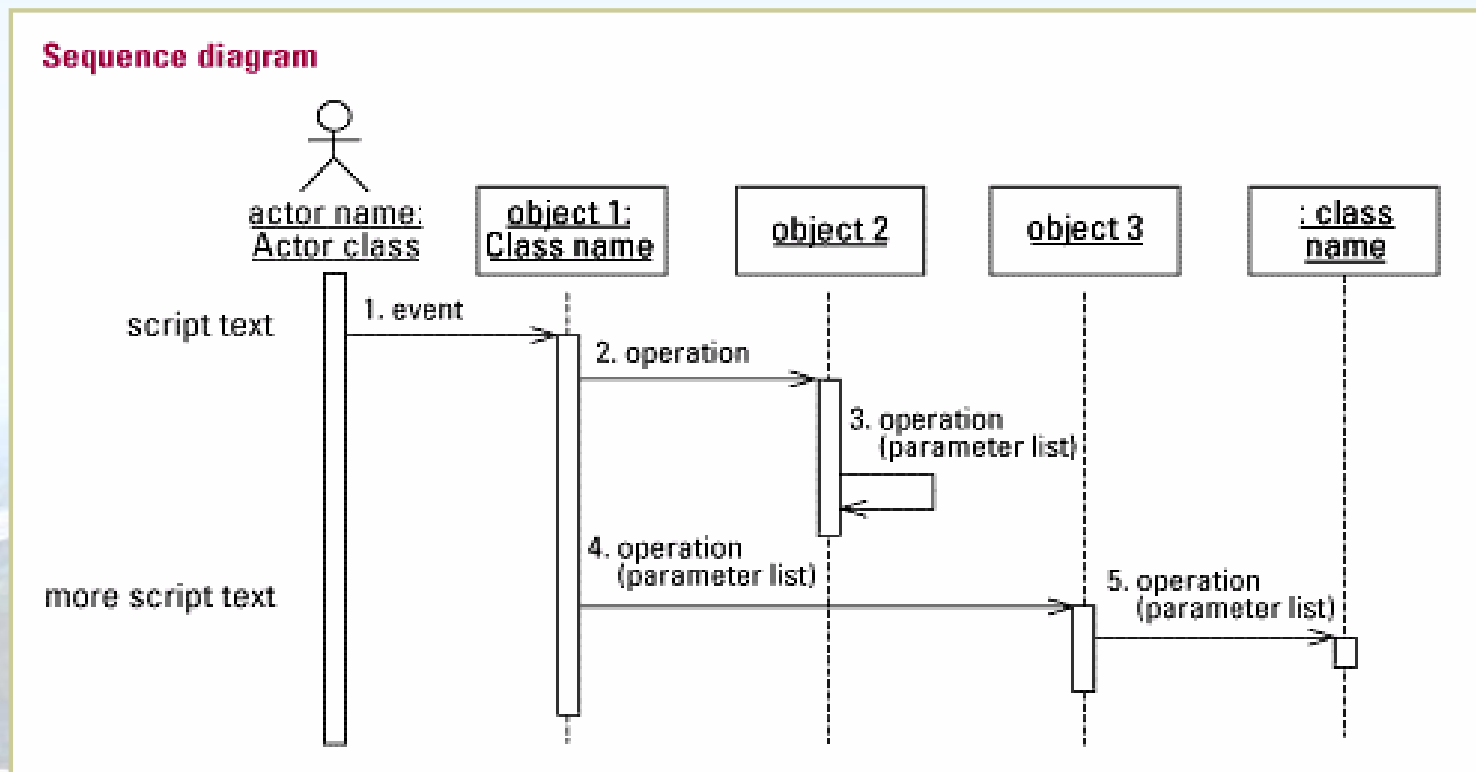




## 4.9 UML动态建模表示

### 4.9.3 序列图 (Sequence Diagram)

- 序列图用于描述以时间为序的交互。
  - 这里的交互是指双方通过消息传递来执行动作和获取结果的过程。





## 4.9 UML动态建模表示

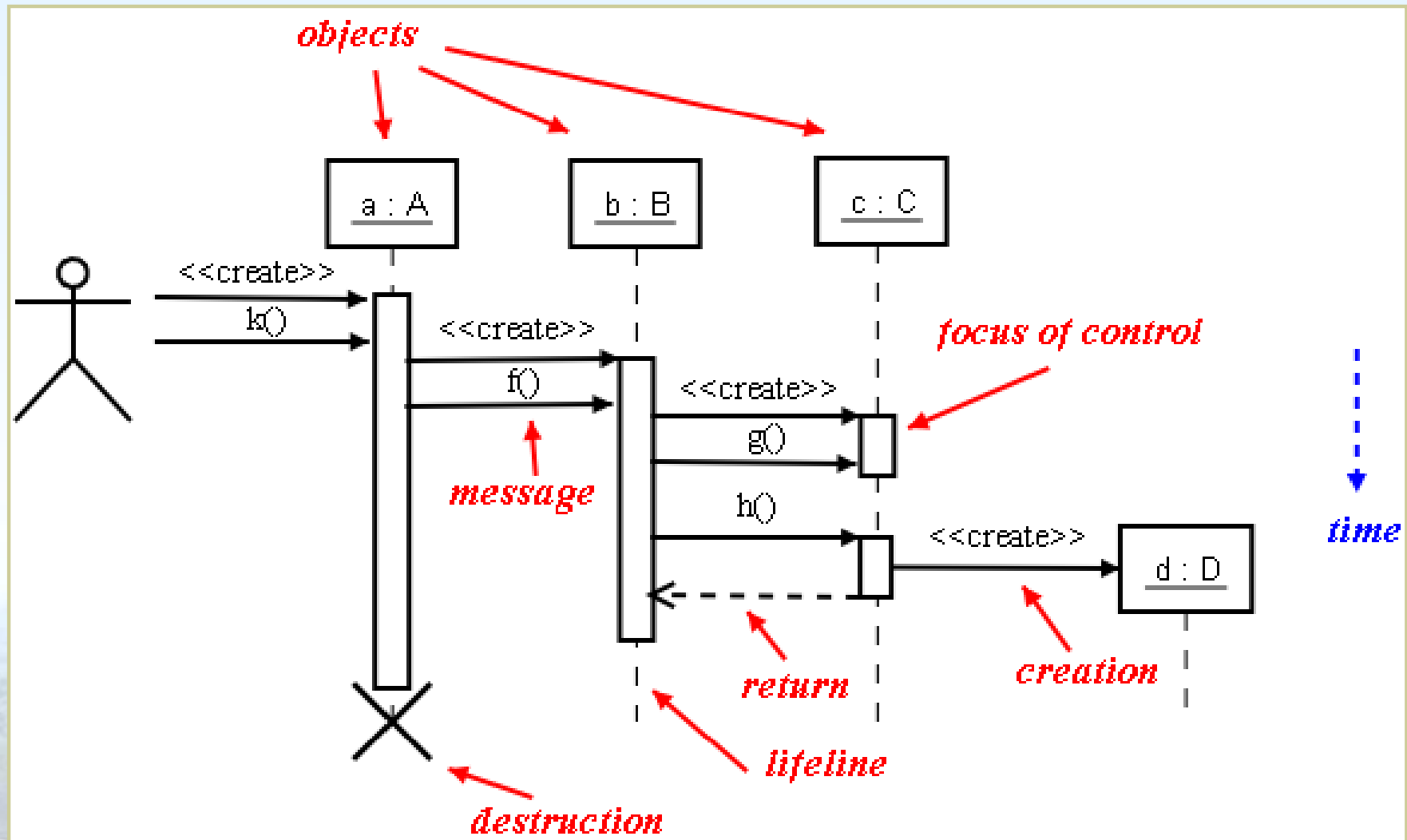
### 4.9.3 序列图

- Shows sequence of operations over time.
- Relates behavior of multiple objects.
- Elements:
  - **objects** (and their classes).
  - **lifelines** represent an object's existence in time.
  - **message exchanges** between objects.
- Arrangement:
  - objects organized horizontally.
  - time flows downward to model an object's "lifeline".
  - arrows between lifelines represent method calls
    - bidirectional to model call/return.



## 4.9 UML动态建模表示

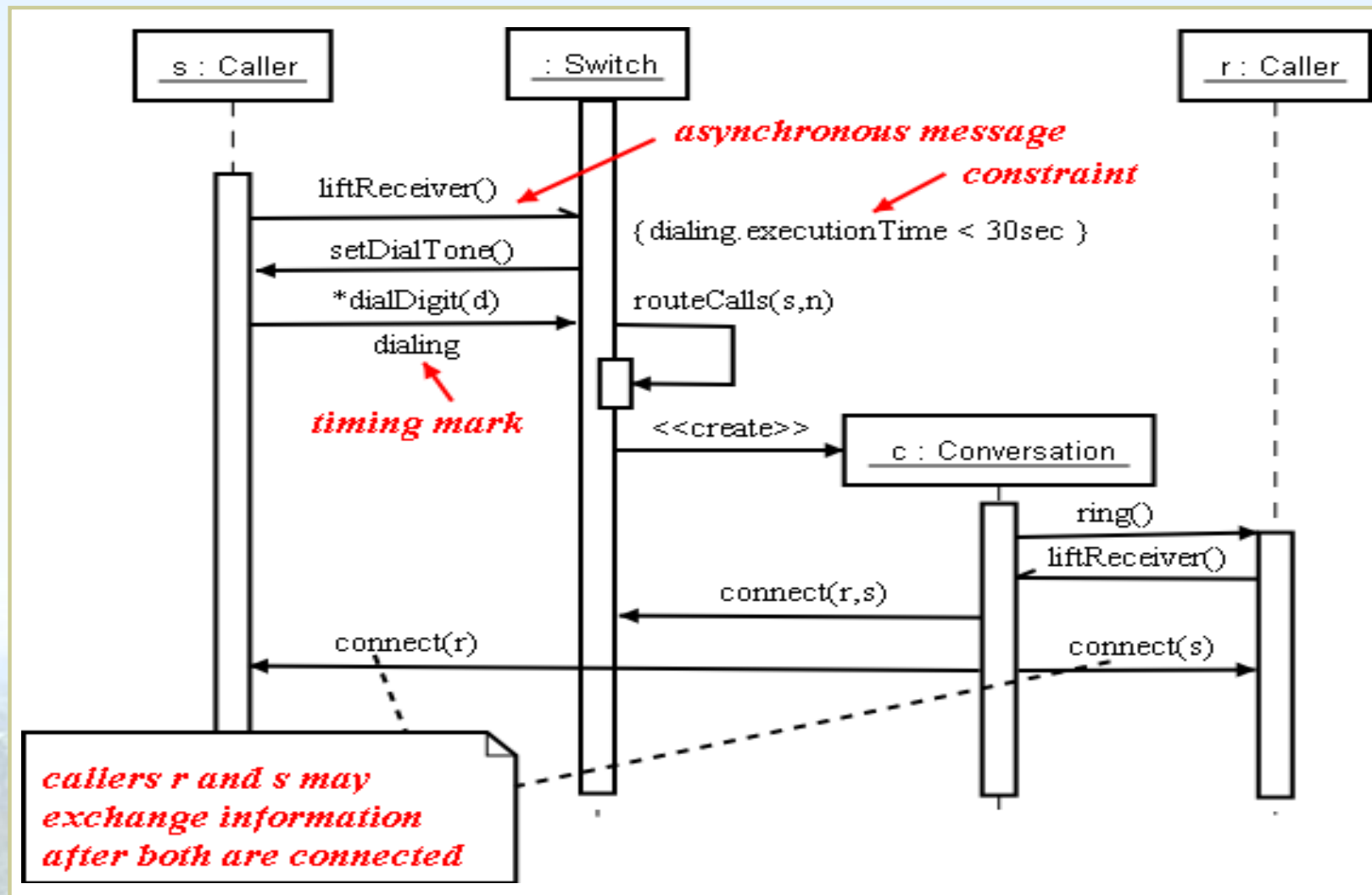
### 4.9.3 序列图





## 4.9 UML动态建模表示

### 4.9.3 序列图





## 4.9 UML动态建模表示

### 4.9.3 序列图

- Synchronous messages:
  - correspond to procedure calls.
  - caller is “blocked” until control returns.
  - notation: full arrow.
- There are also asynchronous messages:
  - correspond to “signals”.
  - notation: half-headed arrow.
  - can create new thread, new object, communicate with existing thread.





## 4.9 UML动态建模表示

### 4.9.3 序列图

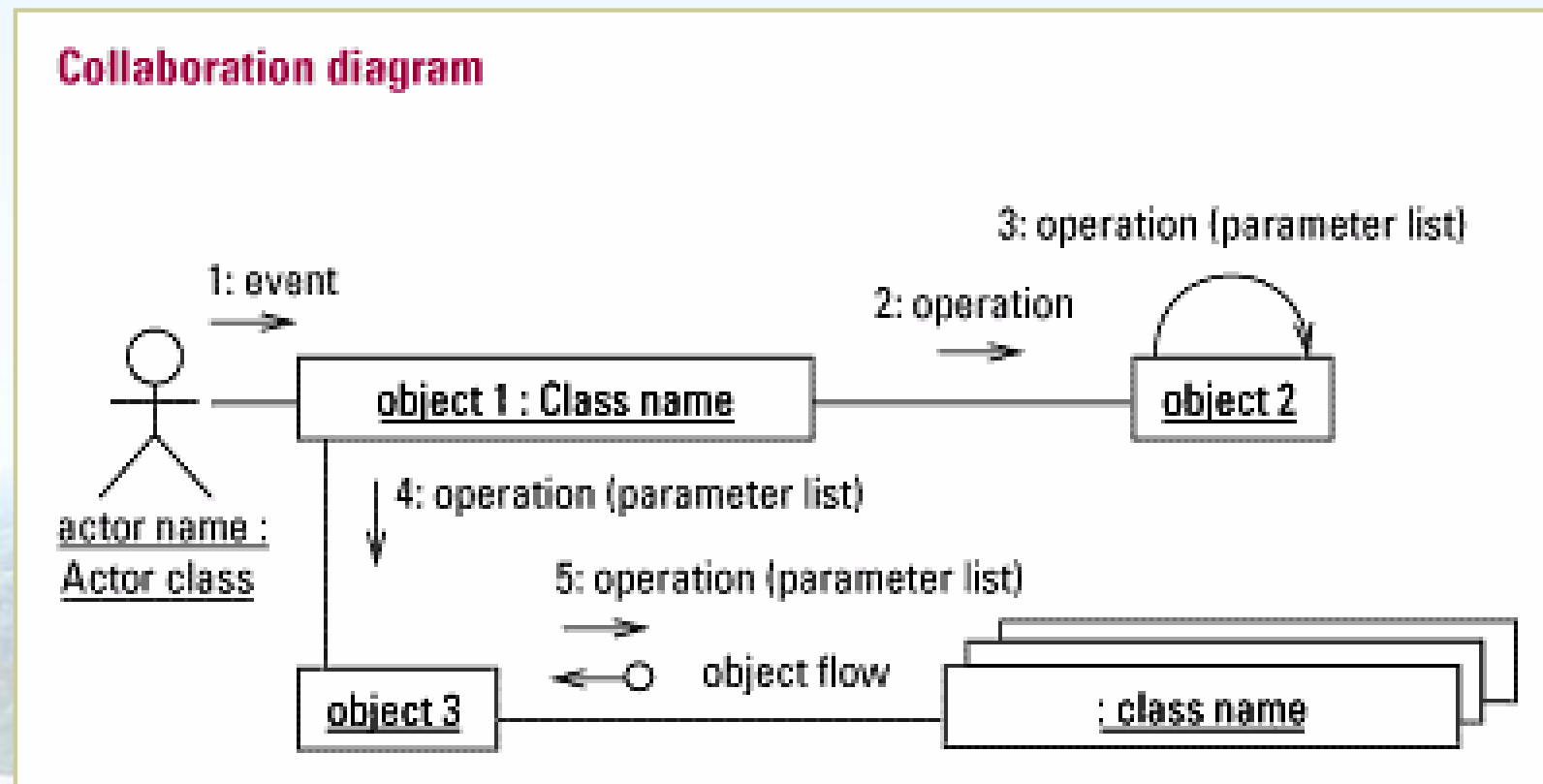
- Typically, a sequence diagram models **only one flow of control**:
  - use conditions for simple conditional logic.
  - otherwise, use multiple sequence diagrams .
- Returns are typically omitted.
- Use focus of control to visualize when computation is taking place.
- You can show actors in sequence diagrams of this helps understanding the diagram.
- Keep it simple!



## 4.9 UML动态建模表示

### 4.9.4 协同图 (Collaboration Diagram)

- 协同图用于描述系统中不同角色之间的交互。
  - 这里的交互描述强调角色之间的联系，而不是时间顺序。





## 4.9 UML动态建模表示

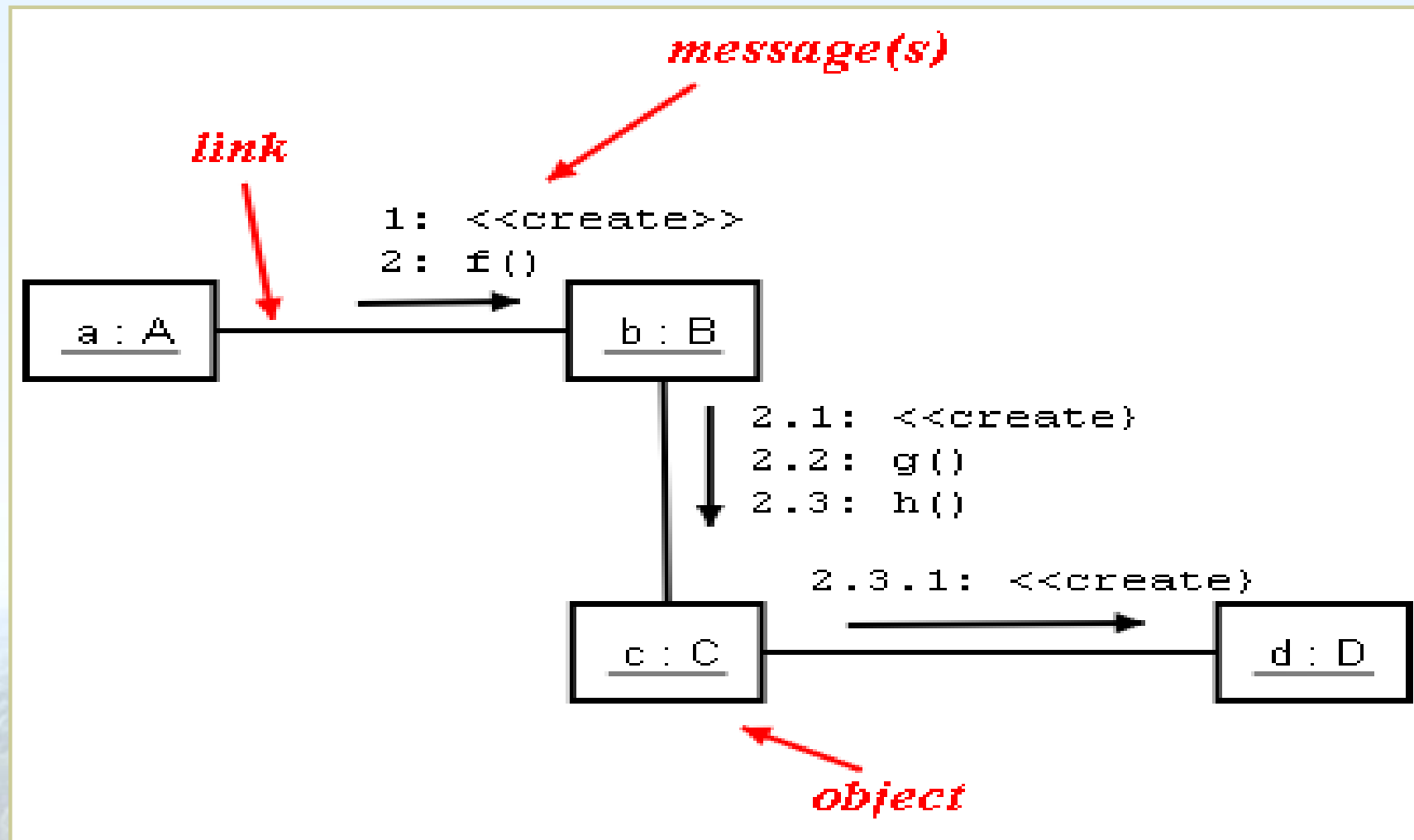
### 4.9.4 协同图

- Very similar to sequence diagrams, but emphasizes structural organization instead of time.
- Elements:
  - **objects**.
  - **links** between objects.
  - multiple **messages** may be sent along each link:
    - <<create>> message models object creation.
  - associate **sequence numbers** with messages:
    - 1, 2, 3, 4, ...
    - 1, 1.1, 1.2, 2, 2.1, 2.1.1, 2.2, 2.3 (makes relationships between methods explicit).



## 4.9 UML动态建模表示

### 4.9.4 协同图





## 4.9 UML动态建模表示

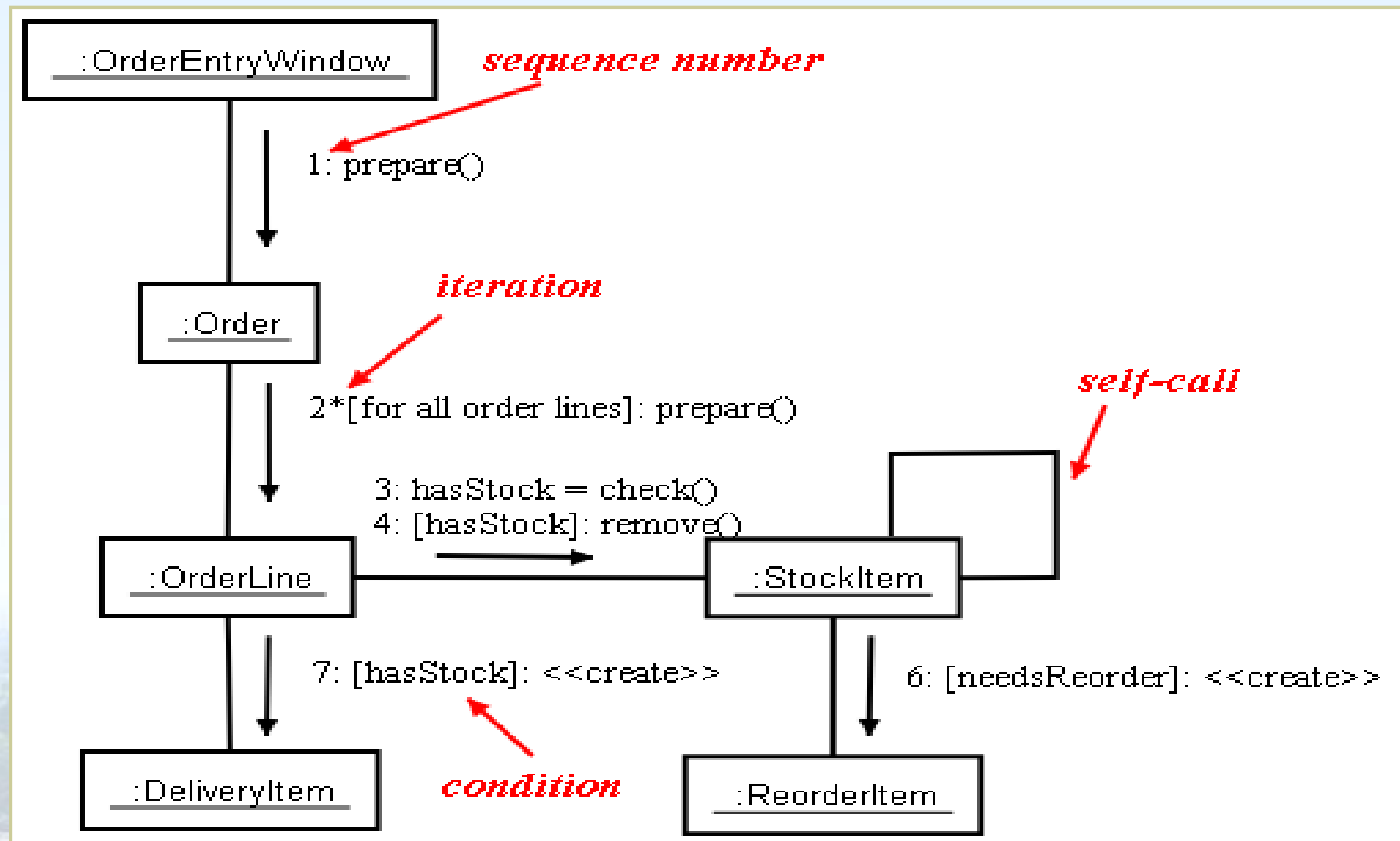
### 4.9.4 协同图

- Collaboration diagrams may also contain:
  - conditions
  - iteration (\*)
  - self-calls
  - asynchronous messages
- Conversion between collaboration diagrams and sequence diagrams always possible:
  - each kind of diagrams has unique advantages.



## 4.9 UML动态建模表示

### 4.9.4 协同图



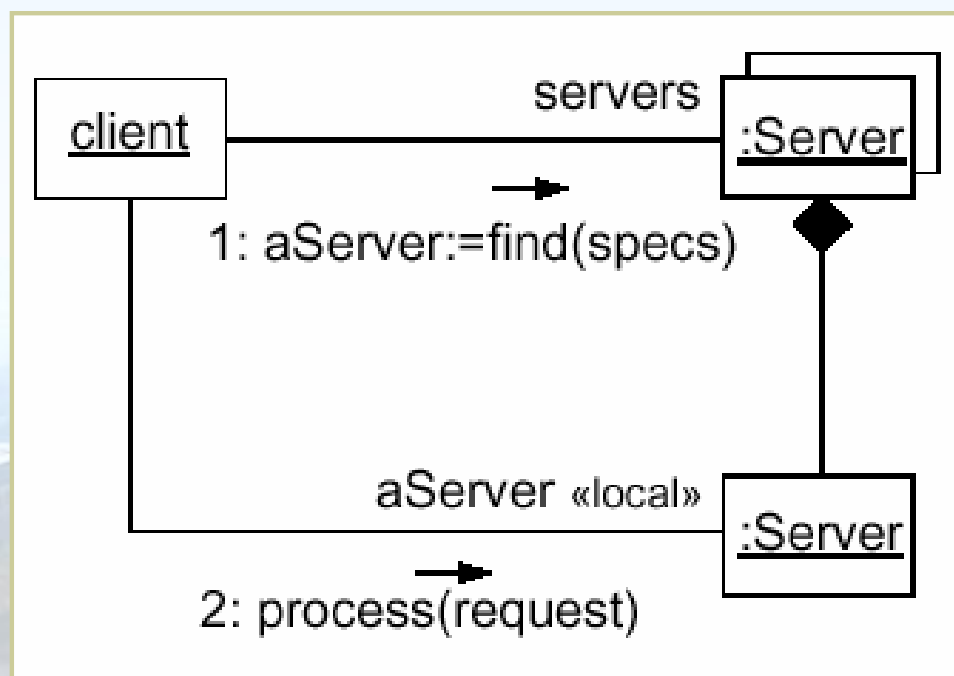




## 4.9 UML动态建模表示

### 4.9.4 协同图

- 涉及多个对象的协同图：
  - 协同图中箭头的涵义与序列图相同。
  - `:=` 表示返回值的赋值。





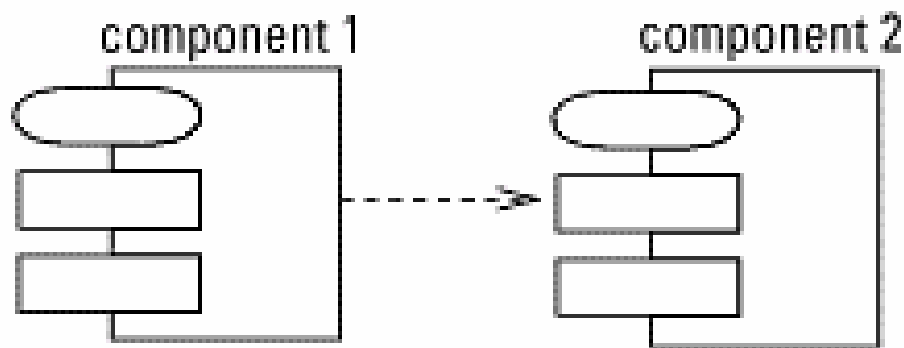
## 4.10 UML系统实现表示

### 4.10.1 组件图 (Component Diagram)

- 组件图用于描述系统中不同的软件组件之间的依赖关系，这些组件包括源码组件、二进制码组件、可执行组件。

#### COMPONENT DIAGRAM

Shows the dependencies between software components

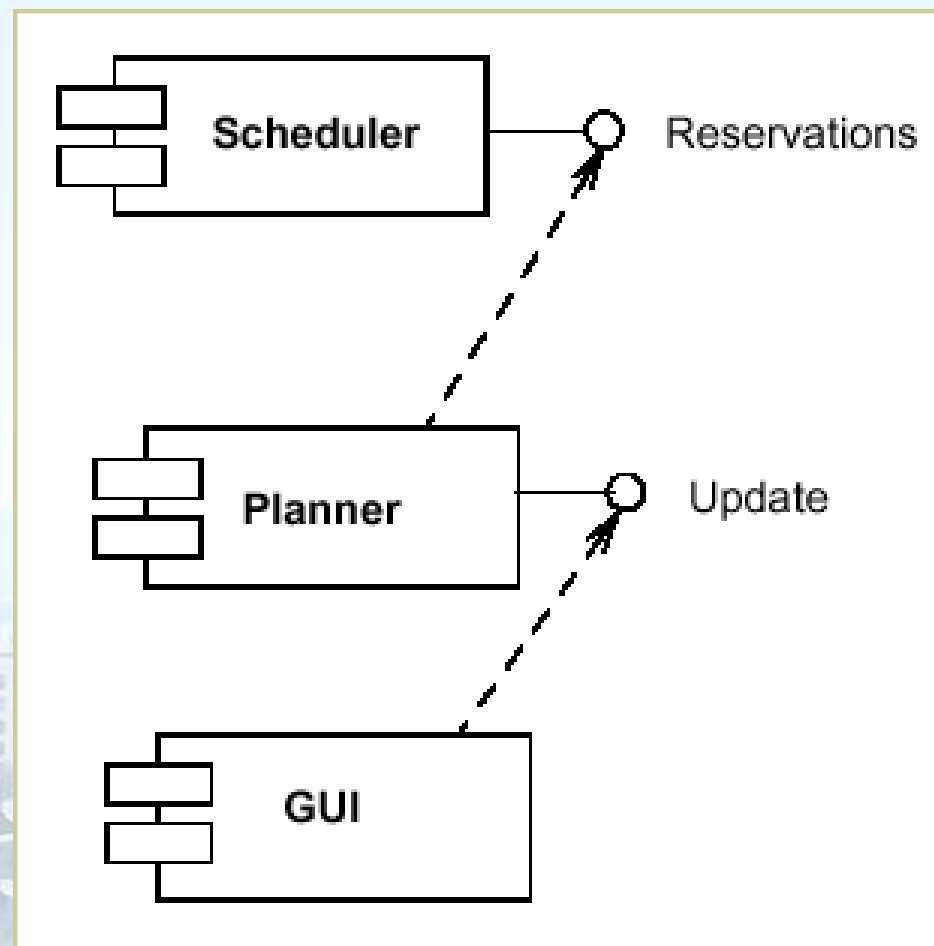




## 4.10 UML系统实现表示

### 4.10.1 组件图

- 表示组件与另一组件的接口之间的依赖关系。





## 4.10 UML系统实现表示

### 4.10.2 部署图 ( Deployment Diagram )

- 部署图用于描述系统在运行时组件、进程以及其中的对象的配置关系。

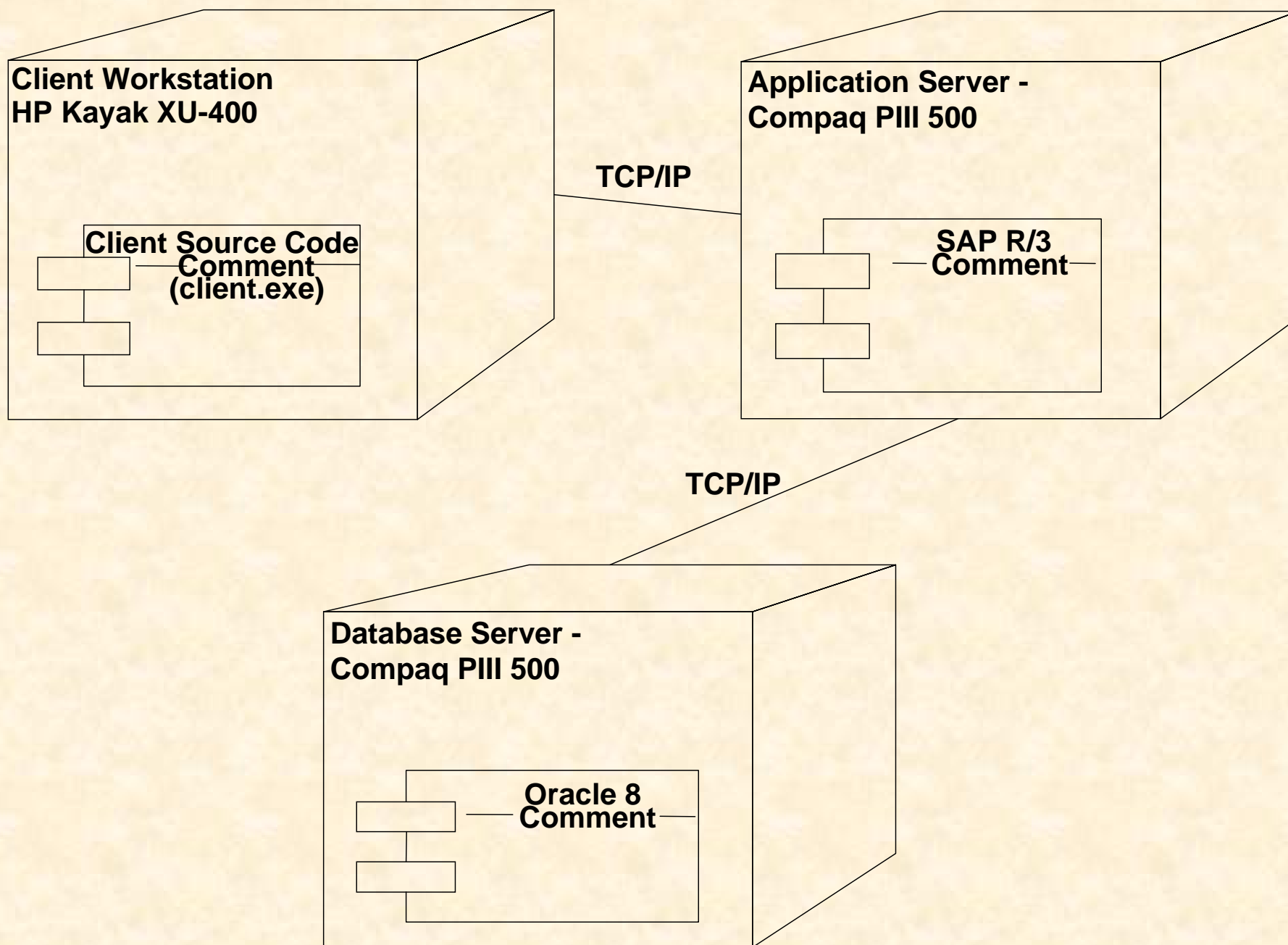
#### DEPLOYMENT DIAGRAM

Shows the configuration of runtime processing elements





## 4.10 UML系统实现表示







## 4.11 代码生成与文档生成

### 4.11.1 从模型生成代码

- 以**Rose**为例，目前能够生成代码的只有类图。
- 代码生成与目标语言的设定有关。
- 用类图可以自动生成类的定义（全部）和类的实现（仅限于方法的框架），后者的内容需程序员自己加入。
- 所生成的代码中，有自动更新和保留两部分，前者在每次生成时都随模型变化而变化，后者（是程序员自己加入的内容）则在生成时保持不变。




## 4.11 代码生成与文档生成


### 4.11.1 从模型生成代码


- 例：用类 **SharedHeap** 的模型产生对应的 .h 文件和 .cpp 文件。


#### SharedHeap


(from SharedData)

 cpHeap : char[HEAP\_SIZE]

 lHasAlloced : long = 0

 cpStartPtr : char \* = NULL

 shmMalloc(iSize : int = 0) : void \*

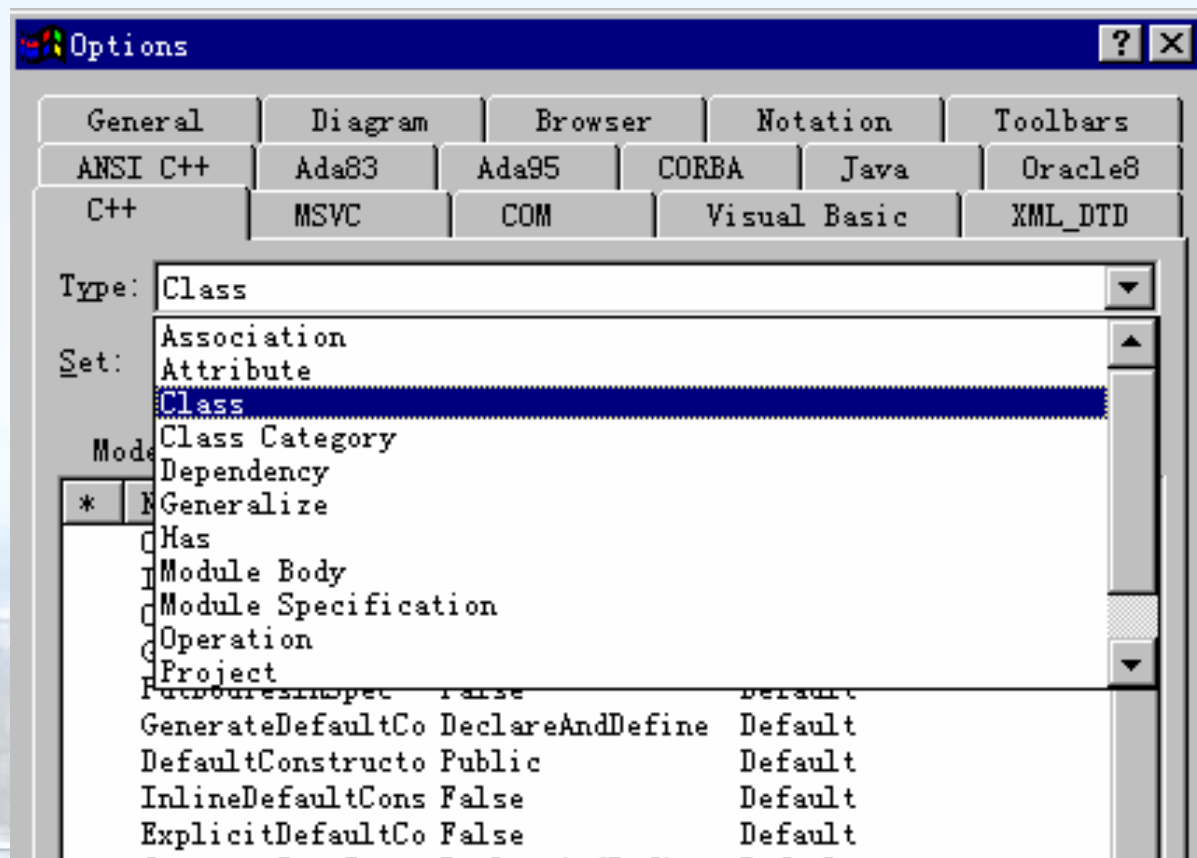
 resetHeap() : void



## 4.11 代码生成与文档生成

### 4.11.2 在Rose中设定代码生成的特性

- 在Rose . Tools . Model Properties . Edit下，可设定代码生成的特性。





## 4.11 代码生成与文档生成

### 4.11.3 文档自动生成的原理

- 以**Rose**为例，所有用**UML**建立的模型都以文本方式存储在对应的**.mdl文件**中。
- 文档自动生成工具（**SoDa**）分析**.mdl**文件，根据欲生成的文档的模板中的要求，过滤出所需的信息，填入该文档模板的一个实例（**Word**文件）中，最后产生以该模板为框架的一份文档。
- 由于是根据模型自动生成文档，所以可始终保持其一致性。



## 4.11 代码生成与文档生成

### 4.11.3 文档自动生成的原理

- SoDa 自带的文档模板有几十种（均为英文），可以用 Word 手工将其改为中文模板。
- 文档模板的格式可以自己定义，但要采用 SoDa 规定的描述语言。







# 要点与引伸

- UML 本身是一种工具，使用的好坏主要取决于使用者。但是，UML 中的确渗透了面向对象的基本思想，使用时要注意挖掘。
- 这些 UML 表示在系统分析和设计中都可以使用。理解这些 UML 表示并不难，难的是应用到具体活动的具体建模过程中。
  - 从已有的范例（特别是分析模式和设计模式）中汲取经验，可能是比较有效的办法。



## 要点与引伸（续）

- 动态模型描述的是目标系统与对象状态（数据值）相关 / 与并发与同步相关 / 与时间相关 / 与处理逻辑相关的、多侧面的动态特征，重点体现了系统中主要对象之间的协同关系，这对于表现和理解并发、分布系统是至关重要的。
- 目前的工具还无法保证动态模型与程序语义的一致性。





# 下一次的內容

- 面向对象系统分析与设计 - 面向对象系统分析
  - 面向对象分析与设计的基本概念
  - 功能性需求的建模
  - 范例：一个图书馆信息系统的需求建模
  - 概念建模
  - 范例：一个图书馆信息系统的概念建模

