UML建模技术

David

Contents

- Ü 基本概念
- Ü 工作流
- Ü UML图示
  - è 用例模型
  - è 序列图
  - è 类图
  - è 协作图
  - è 活动图
  - è 组件图
  - è 部署图
- Ü UML图与模型
- Ü UML图与项目过程

OO基本概念

- Ü对象
- Ü类
- Ü属性
- Ü操作（方法）
- Ü接口（多态）*
- Ü构件*
- Ü关系
- Ü包
- Ü子系统

OOAD是主流技术

- OOAD大部分情况下比结构化设计好：
  - 结构化设计是过时的东西，它强调软件的结构按照功能来组织，一旦功能改变，软件的结构就会不稳定。
  - 而OO设计把数据流和功能统一起来，IT行业绝大部分（70-80%）的软件设计可以采用OO方法，目前国外流行的趋势也是这样，剩下的少部分有特定需求的可能还会用传统方法。

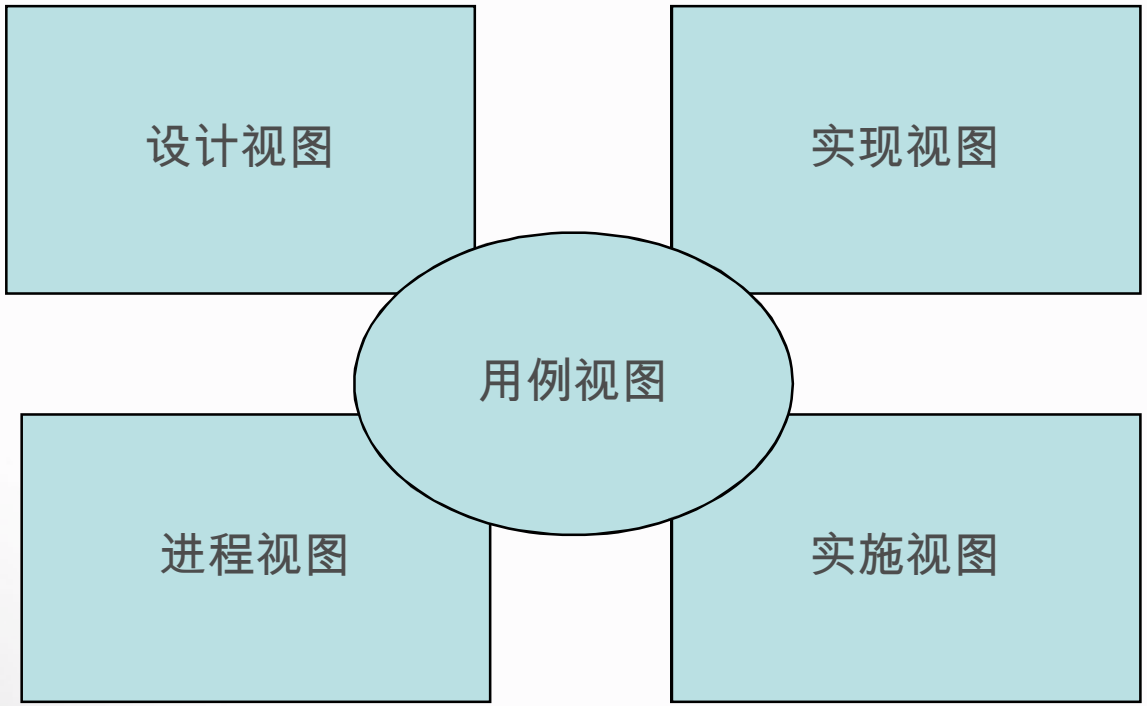
Object Oriented Analysis

- 用面向对象方法分析问题域，建立基于对象、消息的业务模型，形成对客观世界和业务本身的正确认识。
- 生成业务对象的动、静态模型和抽象类。

Object Oriented Design

- 针对OOA给出的问题域模型，用面向对象方法设计出软件基础架构（概要设计）和完整的类结构（详细设计），以实现业务功能。
- 生成对象类的动、静态模型（解决域）。

“4+1”视图



词汇
功能

行为

性能
可伸缩性
吞吐量

系统装配
配置管理

系统拓扑
分布
交付
安装

基于三种模型的分析过程

- Ü 功能模型定义“做什么”
- Ü 动态模型定义“何时做”
- Ü 对象模型定义“对谁做”。

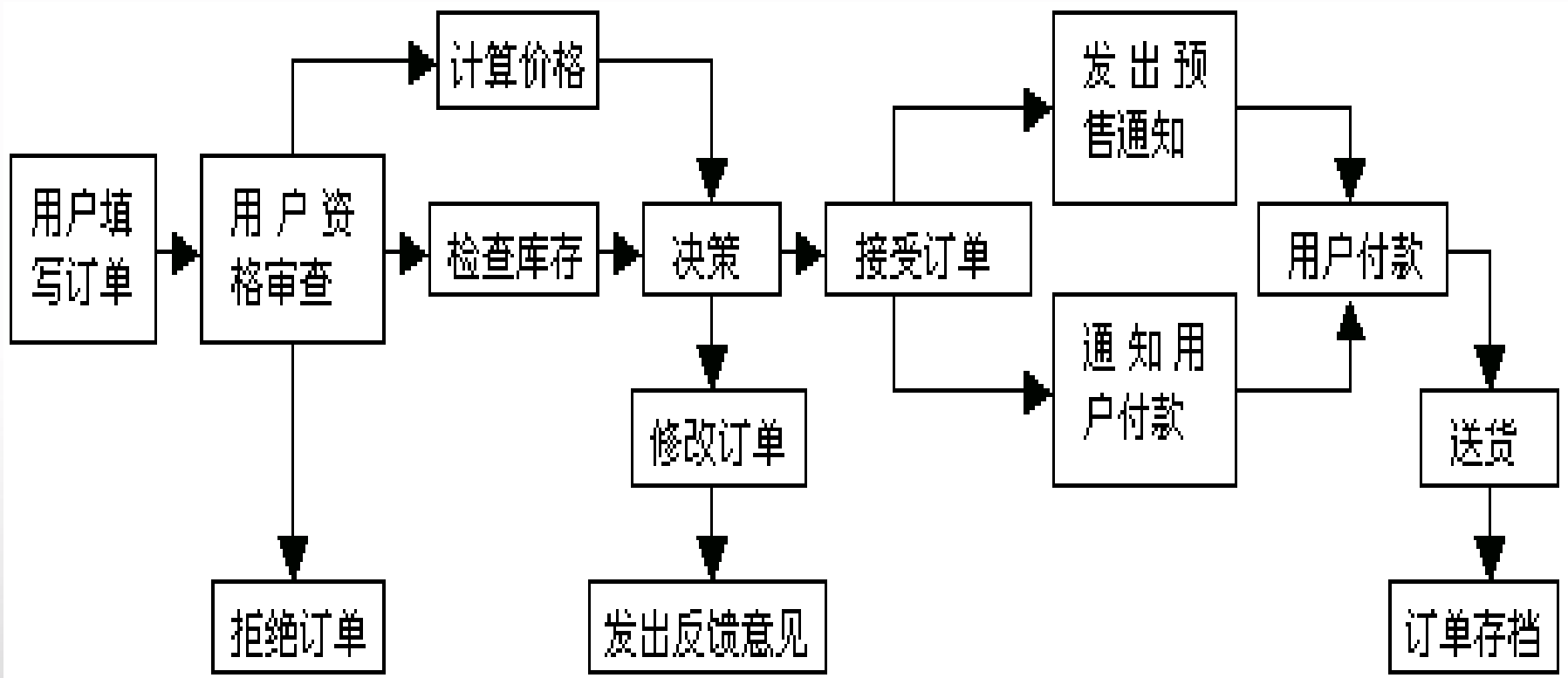
工作流

- 是“业务过程的部分或整体在计算机应用环境下的自动化”。
- 它主要解决的是“使在多个参与者之间按照某种预定义的规则传递文档、信息或任务的过程自动进行，从而实现某个预期的业务目标，或者促使此目标的实现”。

工作流

- 简单地说，工作流就是一系列相互衔接、自动进行的业务活动或任务。
- 一个工作流包括一组任务（或活动）及它们的相互顺序关系，还包括流程及任务（或活动）的启动和终止条件，以及对每个任务（或活动）的描述。

某产品销售的工作流示意图



UML图示

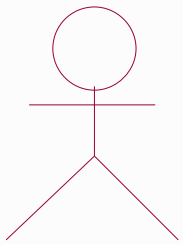
- Use Case Diagram
- Sequence Diagram
- Class Diagram
- Collaboration Diagram
- State Diagram
- Activity Diagram
- Component Diagram
- Deployment Diagram

用例模型

- Ü 用例模型描述的是外部执行者(actor)所理解的系统功能。用例模型用于需求分析阶段
- Ü 首先，它描述了待开发系统的功能需求；
- Ü 其次，它把系统看作黑盒子，从外部执行者的角度来理解系统；
- Ü 第三，它驱动了需求分析之后各阶段的开发工作，不仅在开发过程中保证了系统所有功能的实现，而且被用于验证和检测所开发的系统，从而影响到开发工作的各个阶段和UML的各个模型。
- Ü 在UML中，一个用例模型由若干个用例图来描述，用例图的主要元素是用例和执行者。

Actor -执行者

- "An Actor defines a *coherent set of roles* that users of use cases play when interacting with use cases. An actor has one role for each use case with which it communicates."
- 角色的集合，可以是人或外部系统。
- 定义了"系统边界"。



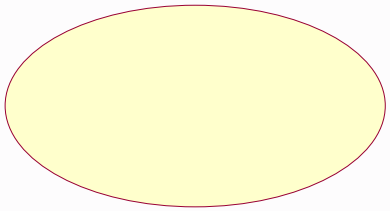
Actor

Actor -执行者

- 执行者是与系统交互的人或物，它代表外部实体，例如，用户、硬件设备或与本系统交互的另一个软件系统。使用用例并与系统交互的任何人或物都是执行者。
- 实践表明，执行者对确定用例是非常有用的。面对一个大型、复杂的系统，要列出用例清单往往很困难，这时可以先列出执行者清单，再针对每个执行者列出它的用例。这样做可以使问题变得容易很多。

Use Case - 用例

- 使用用例视图（UseCase View）是UML的核心，主要作用是用来说明系统功能性的需求，找出系统中的使用案例（Use Case）与角色（Actor,或称动作者），和利用使用案例（Use Case）的模型来充分表达出软件功能的需求。



Use Case

Use Case - 用例

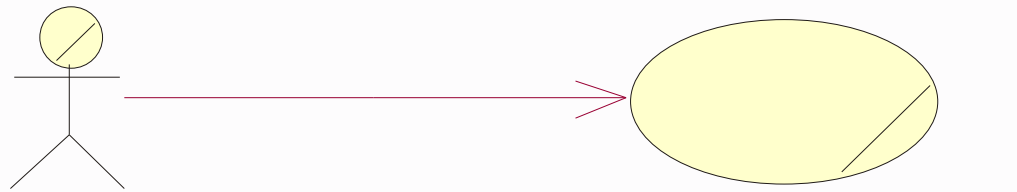
- 大部分用例将在项目的需求分析阶段产生，并且随着开发工作的深入还会发现更多用例，这些新发现的用例都应及时补充进已有的用例集中。用例集中的每个用例都是对系统的一个潜在的需求。

- 发现执行者
  - 为获取用例首先要找出系统的执行者，可以通过请系统的用户回答一些问题的办法来发现执行者。
- 获取用例
  - 事实上，从识别执行者起获取用例的过程就已经开始了。一旦识别出了执行者，就可以对每个执行者提出问题以获取用例。

Use Case - 用例

- Ü 一个用例实质上是用户与计算机系统之间的一次典型的交互作用，它代表的是系统的一个完整的功能。在UML中把用例定义成系统执行的一系列动作，动作的结果能被外部执行者察觉到。
- Ü 在UML用例图中，用例表示为一个椭圆。
- Ü 用例代表某些用户可见的功能，实现一个具体的用户目标。
- Ü 用例由执行者激活，并提供确切的值给执行者。
- Ü 用例可大可小，但它必须是对一个具体的用户目标实现的完整描述。

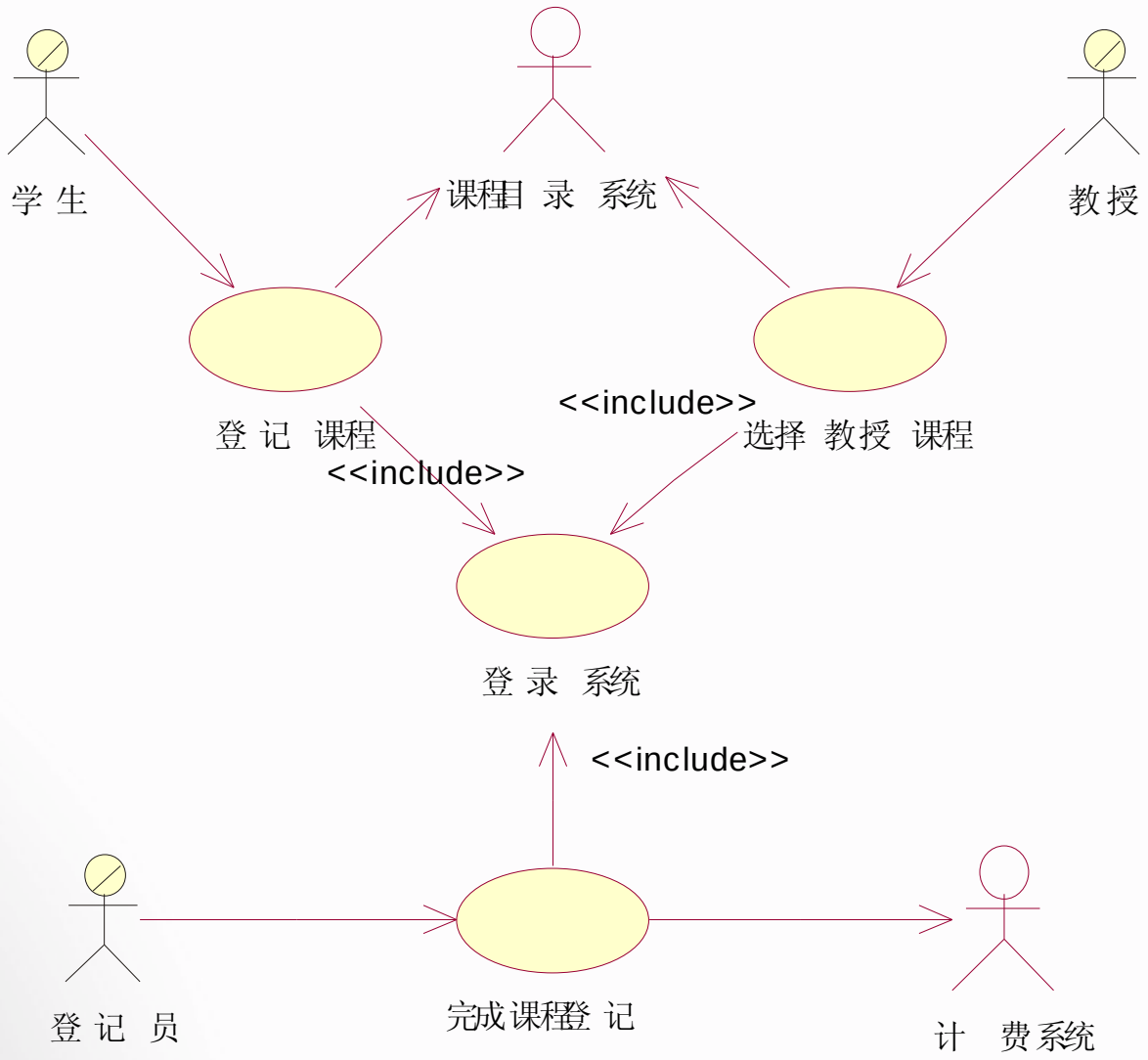
Business Use Case/Actor



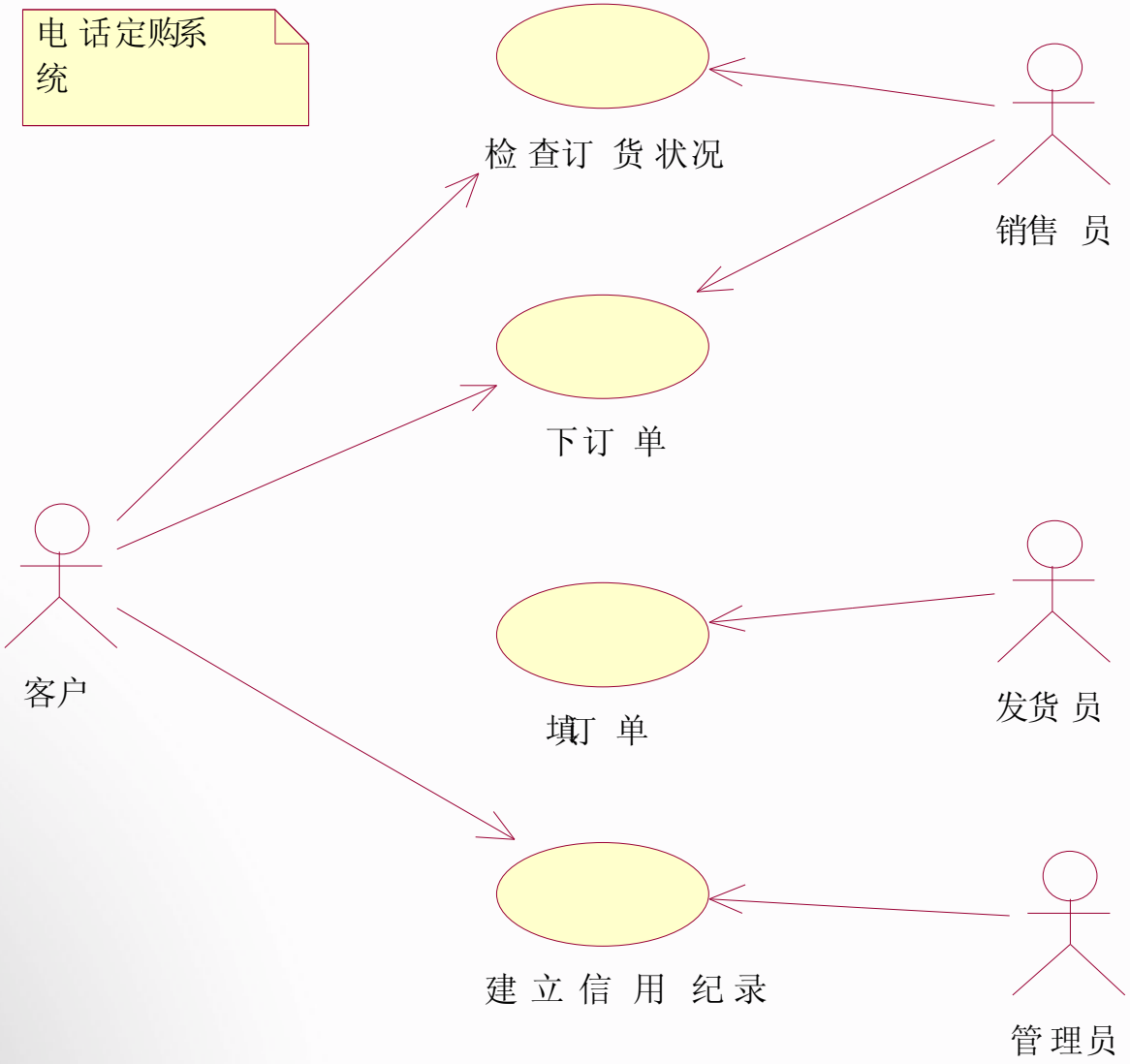
Business Actor

Business UseCase

课程登记系统



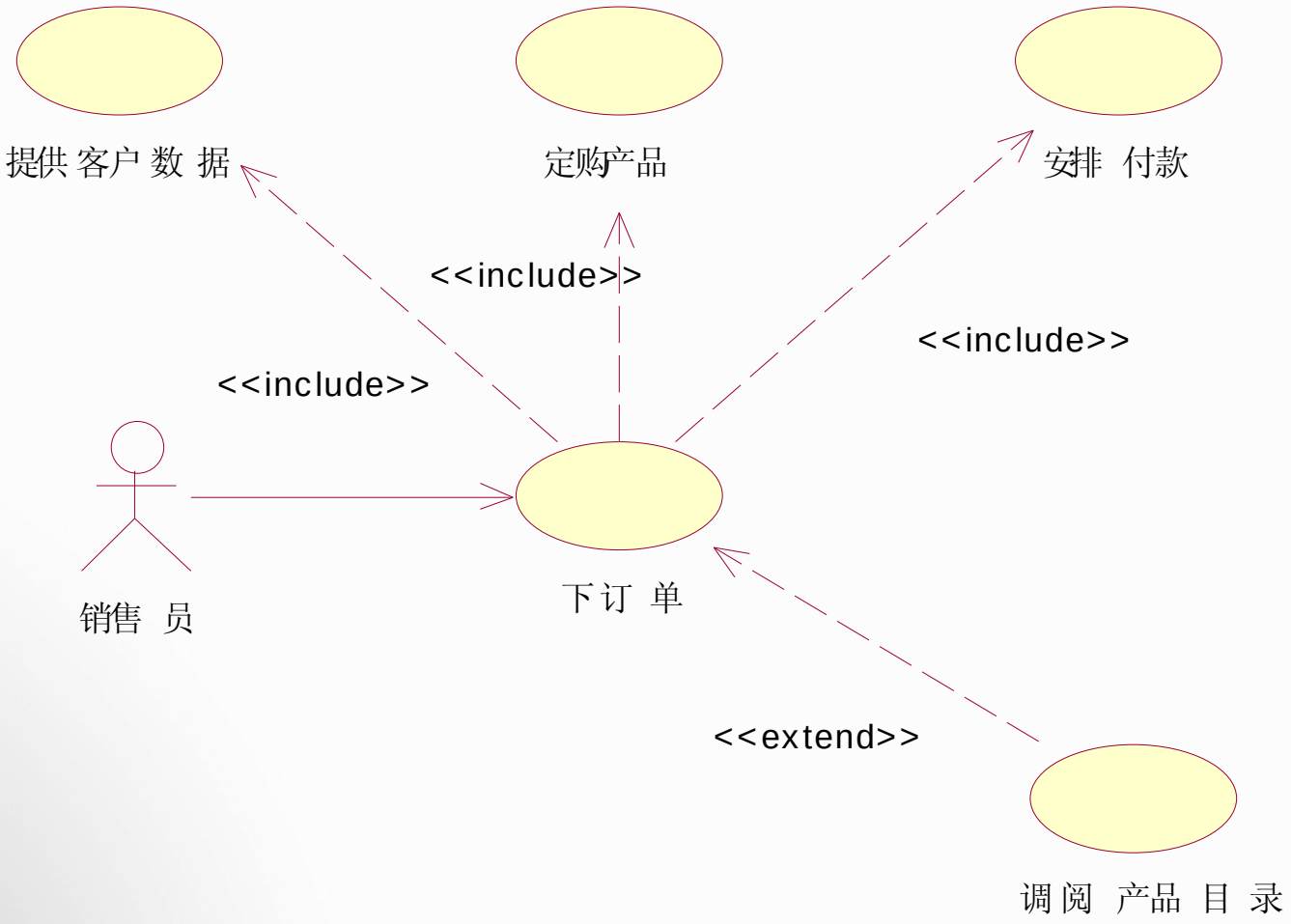
电话定购系统



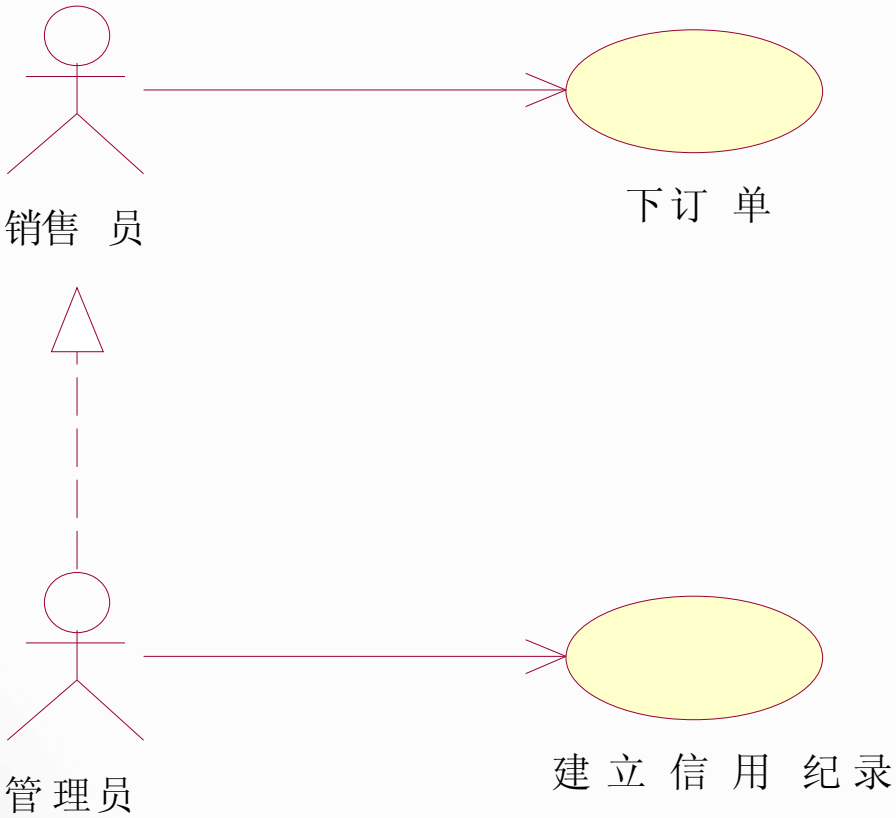
Use Case依赖关系

- 使用关系
  - 当一个用例使用另一个用例时，这两个用例之间就构成了使用关系。
- include/extend

产品定购管理



Actor Generalization



理解Use Case和Actor

- Use Case不是锦上添花的东西，一方面它可以促进与用户沟通，理解正确的需求，另一方面它可以划分系统与外部实体的界限，是系统设计的起点，而最终应该落实到类和实现代码上。
- Use Case View与Logical View应该由明确的相关性。UML中从Use Case到类包的关联可以用依赖（或实现）关系描述。
- Actor不是指人，而是指代表某一种特定功能的角色，因此同一个人可能对应很多个Actor。Actor是虚拟的概念，可以指外部系统和设备。
- Use Case是对系统行为的动态描述，它是OO设计的起点，是类、对象、操作的来源，而通过逻辑视图的设计，可以获得软件的静态结构。

Sequence Diagram

一种动态图，用于按时序展示对象间的消息传递。

- 序列图(Sequence Diagram)是按时间顺序描述了对象间的交互模式；它利用对象的“生命线”和它们之间传递的消息来显示对象如何参与交互。

- 序列图(Sequence Diagram)是一个模型，用于描述对象组如何随着时间在某些行为方面进行协作。序列图捕获单一用例的行为，同时显示在特定用例的时间框架中的对象以及这些对象之间传递的消息。序列图并不显示对象之间的关系。

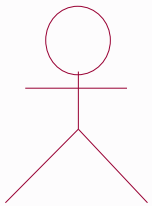
序列图的表示

- 序列图是一种强调消息的时序交互图，它由活动者（Actor）、对象（Object）、消息（Message）、生命线（Lifeline）和控制焦点（Focus of control）组成。
- 在UML中对象表示为一个矩形，其中对象名称标有下划线；消息在序列图中由有标记的箭头表示；生命线由虚线表示。控制焦点由薄薄的矩形表示。

序列图特点

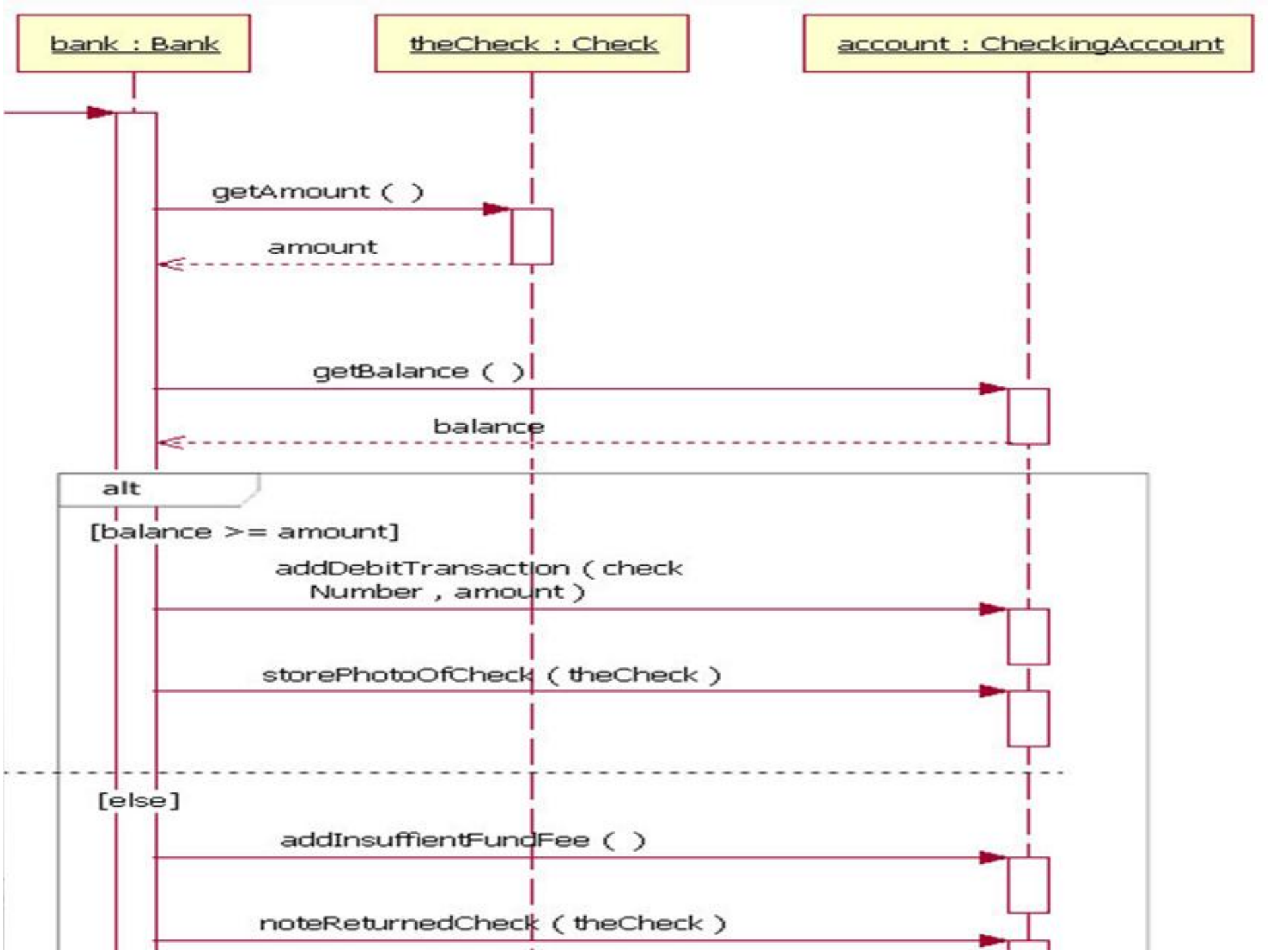
- Ü 反映随时间对象之间的动态协作关系
- Ü 反映对象之间已发送消息的先后顺序
- Ü 说明对象之间的交互过程以及在某一具体位置有何事件发生

序列图元素



Joe : Customer

Banking Screen :
AccountForm



事件流

- 基本事件流：描述该用例的基本流程，指每个流程都“正常”运作时所发生的事情，没有任何备选流和异常流，而只有最有可能发生的事件流；
- 其他事件流：表示这个行为或流程是可选的或备选的，并不是总要执行它们；
- 异常事件流：表示发生了某些非正常的事情所要执行的流程

类图

- 显示系统中类与类之间的交互。
- 显示了模型的静态结构，特别是模型中存在的类、类的内部结构以及它们与其他类的关系等。类图不显示暂时性信息。

类图之定义属性:

- UML描述属性的语法格式如下:

  可见性 属性名: 类型名=初值{性质串}

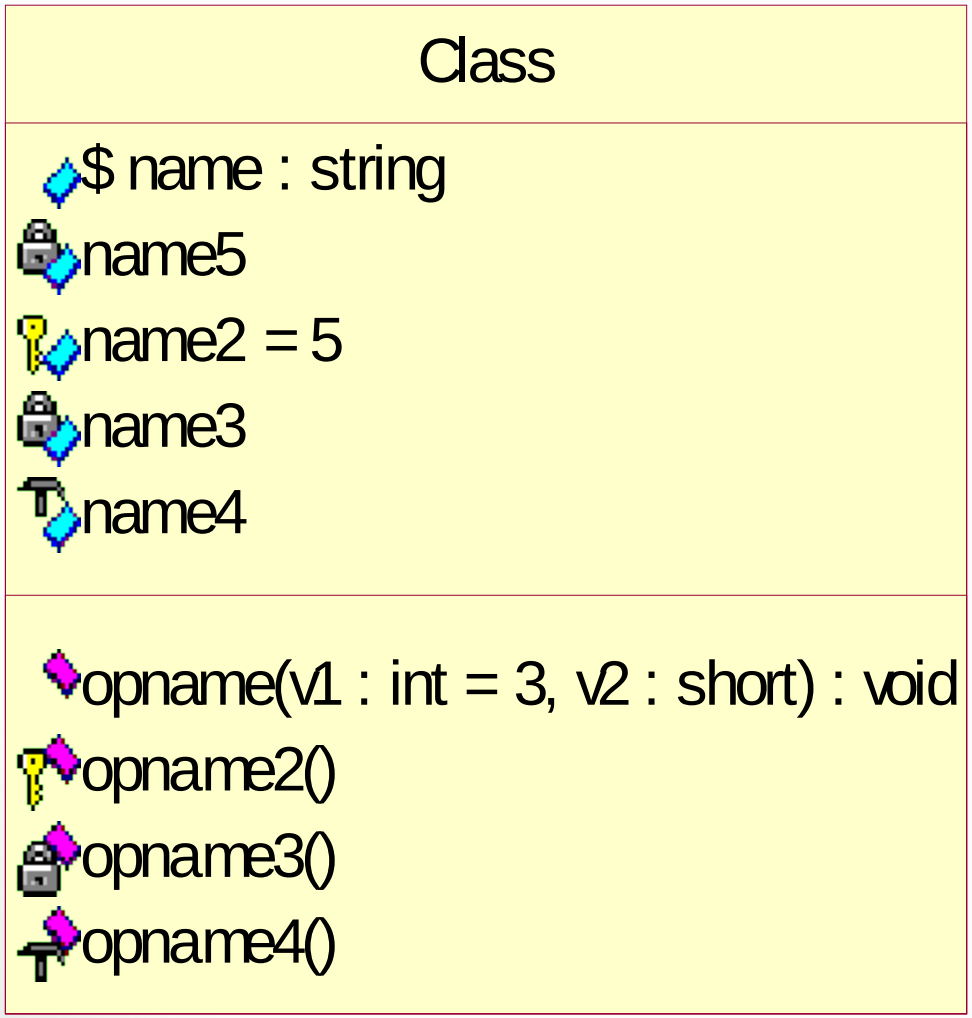
- 属性的可见性通常有三种：

  公有的(public)、私有的(private)和保护的(protected)，分别用+、 - 、 # 号表示。

类图之 定义操作

- UML描述操作的语法格式如下:
  - 可见性 操作名(参数表) ：返回值类型{性质串}
  - 操作可见性的定义方法与属性相同(+-#)。

Rose中的类

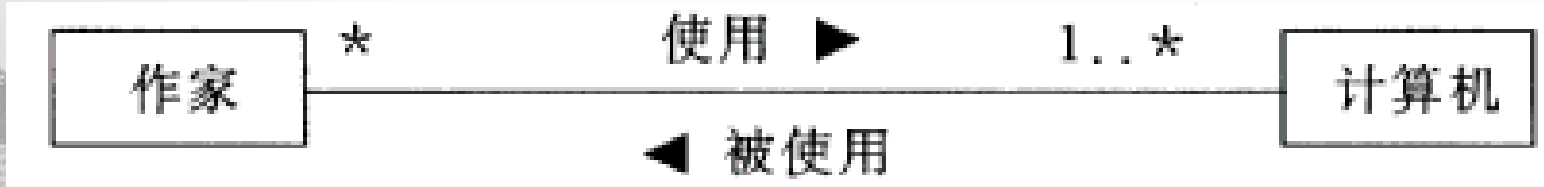


类图之类关系

- Ü类图由类和它们之间的关系组成。定义了类之后，就可以定义类之间的各种关系了。
- Ü类与类之间通常有关联、泛化(继承)、依赖和实现四种关系。
- Ü关联关系（Association）
  - è单向关联关系
  - è双向关联关系
  - è聚合关系（Aggregation）
  - è合成关系（Composition）
- Ü依赖关系（Dependency）
- Ü泛化关系（Generalization）
- Ü实现关系（Implement）

类图之关联关系之双向关联

- 双向关联（普通关联）是最常见的关联关系，只要在类与类之间存在连接关系就可以用普通关联表示。
- 普通关联的图示符号是连接两个类之间的直线。
- A类存在一个B类的全局变量，B类也存在一个A类的全局变量。A类和B类有相同的地位，可以互相传送消息，通信是双方的。
- 经理和职员关系举例



类图之关联关系之单向关联

- 如果关联是单向的，则称为单向关联，其符号是用实线箭头连接两个类。
- A类存在一个B类的全局变量。只有A类向B类发送消息，而不允许B类向A类发送消息。
- 键盘、系统关系举例

类图之关联关系之重数

- Ü 在类图中还可以表示关联中的数量关系，即参与关联的对象的个数。在UML中用重数说明数量或数量范围，例如：
  - è 0..1表示0到1个对象
  - è 0..*或*表示0到多个对象
  - è 1..15表示1到15个对象
  - è 3表示3个对象
- Ü 如果图中未明确标出关联的重数，则缺省重数是1。

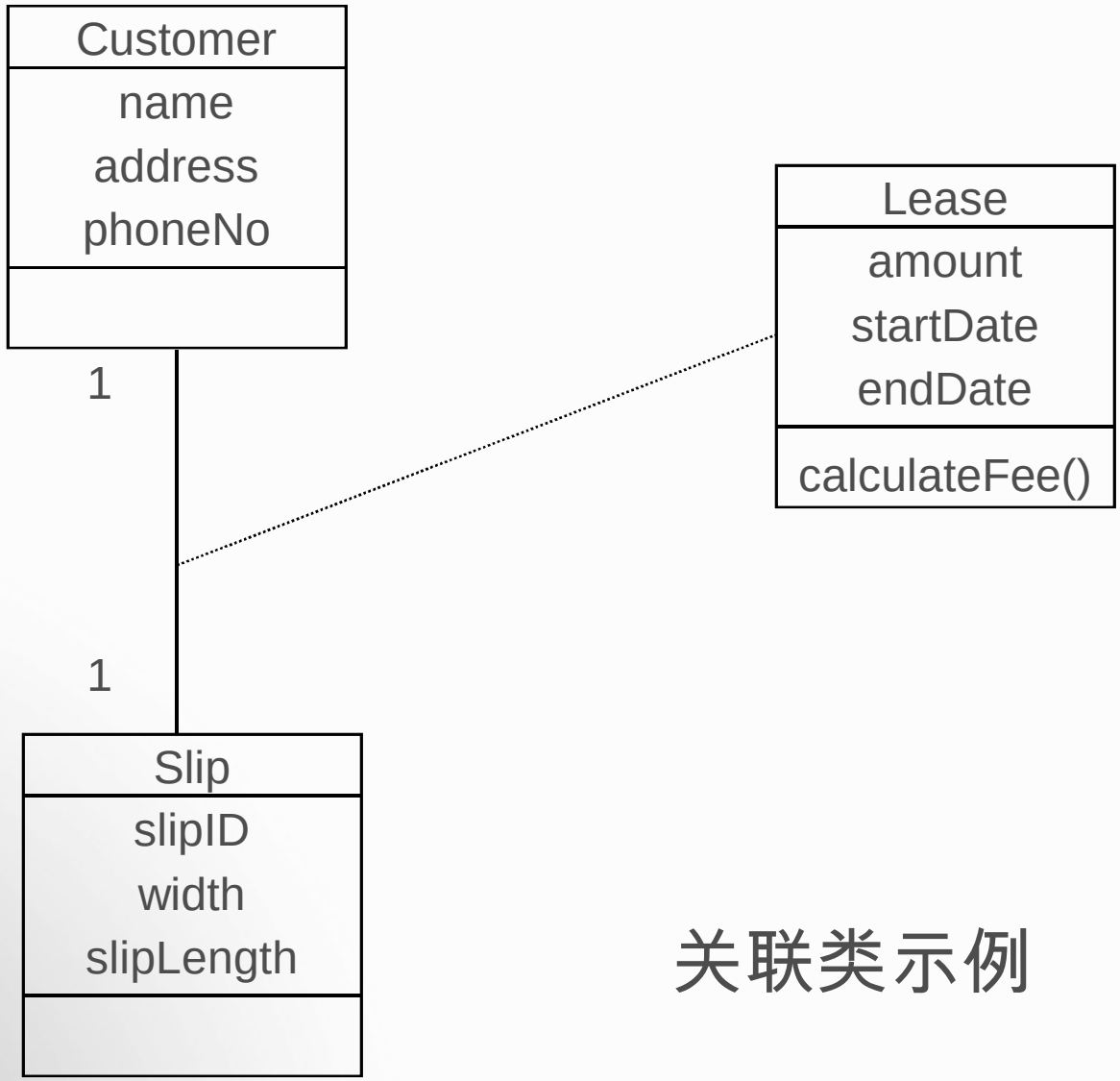
关联的角色

- 在任何关联中都会涉及到参与此关联的对象所扮演的角色(即起的作用)，在某些情况下显式标明角色名有助于别人理解类图。
- 如果没有显式标出角色名，则意味着用类名作为角色名。

关联类

- 为了说明关联的性质可能需要一些附加信息。可以引入一个关联类来记录这些信息。关联中的每个连接与关联类的一个对象相联系。关联类通过一条虚线与关联连接。

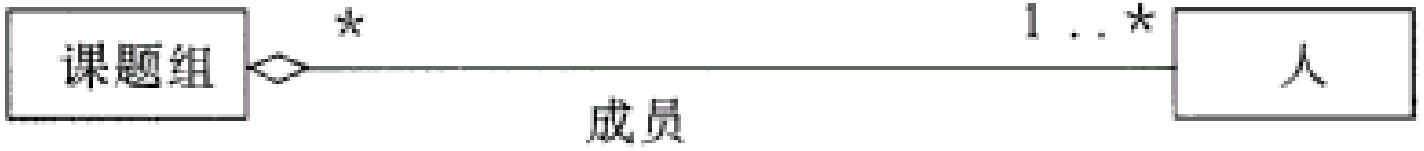
关联类



关联类示例

聚合关系

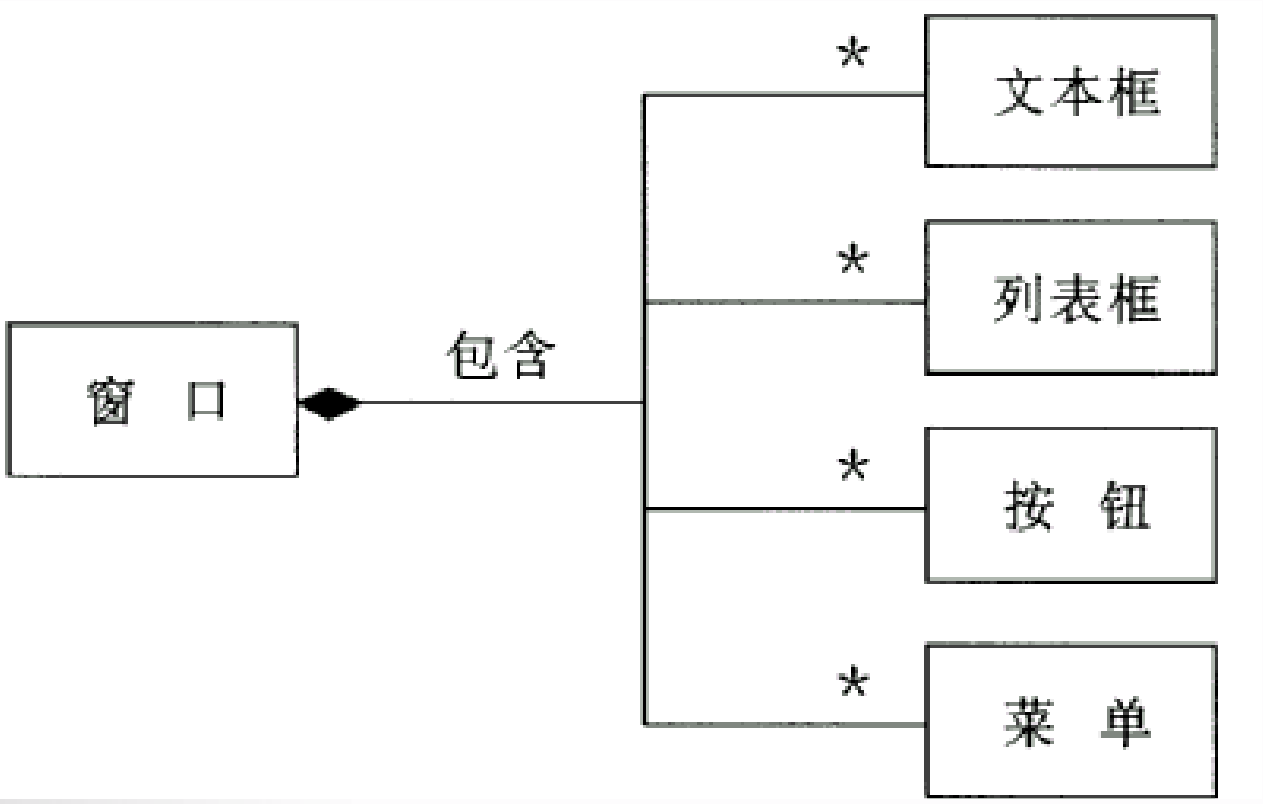
- 聚合关系：是关联关系的一种，是强的关联关系。聚合关系是整体和个体的关系。关联关系的两个类处于同一层次上，聚合关系两个类处于不同的层次，一个是整体，一个是部分。
- A类是整体，B类是部分，A类存在B类的全局变量
- 学校、教室、办公室、操场关系举例



合成关系

- 是关联关系的一种，是比聚合关系强的关系。它要求普通的聚合关系中代表整体的对象负责代表部分的对象的生命周期。
- 表示方法：实心菱形＋实线

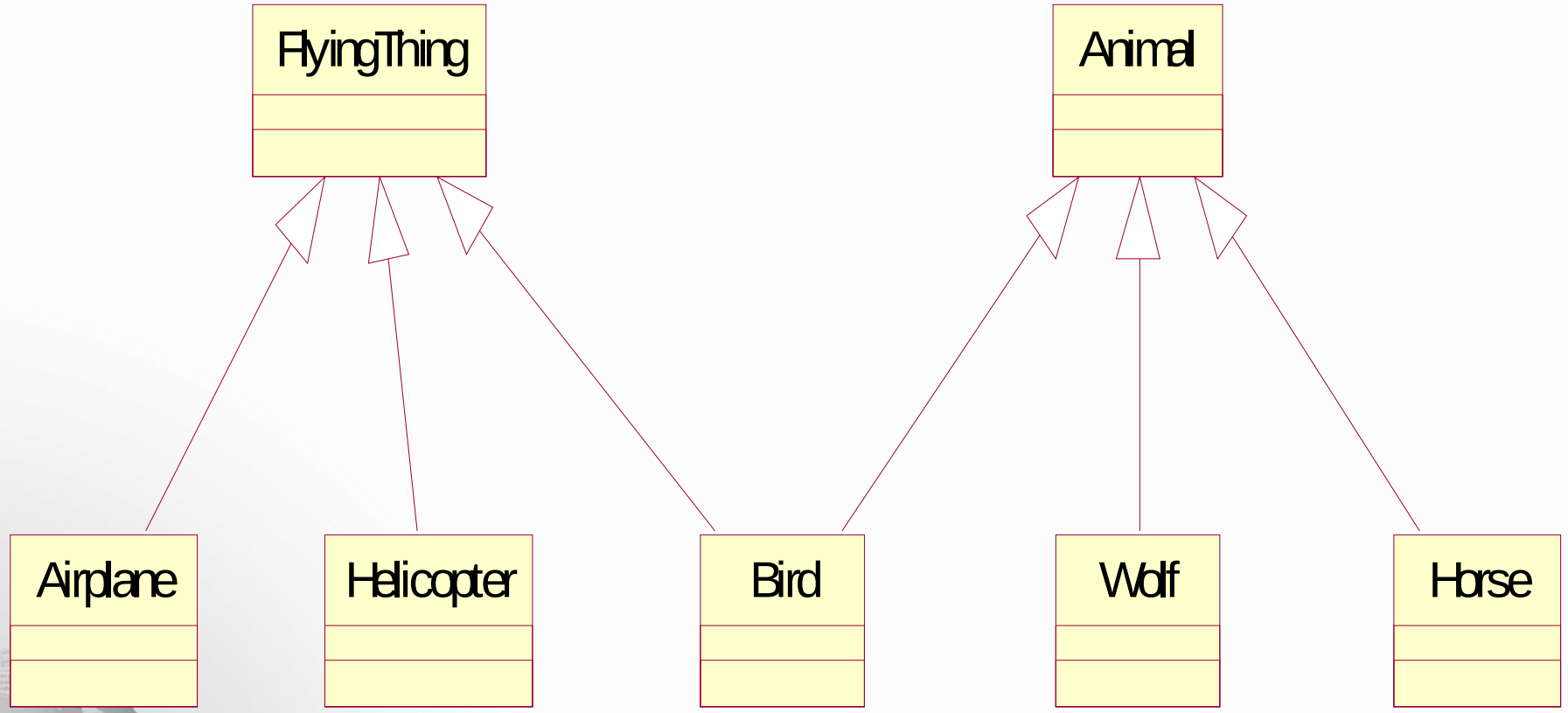
合成关系



泛化关系

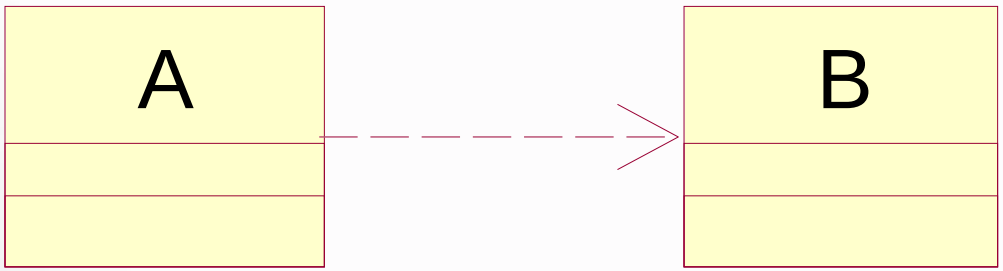
- 表示为类与类之间的继承关系，接口与接口之间的继承。
- 表示方法：用一个空心箭头＋实线，箭头指向父类。或空心箭头＋虚线，如果父类是接口。
- 举例：职员类与经理类

继承



依赖关系

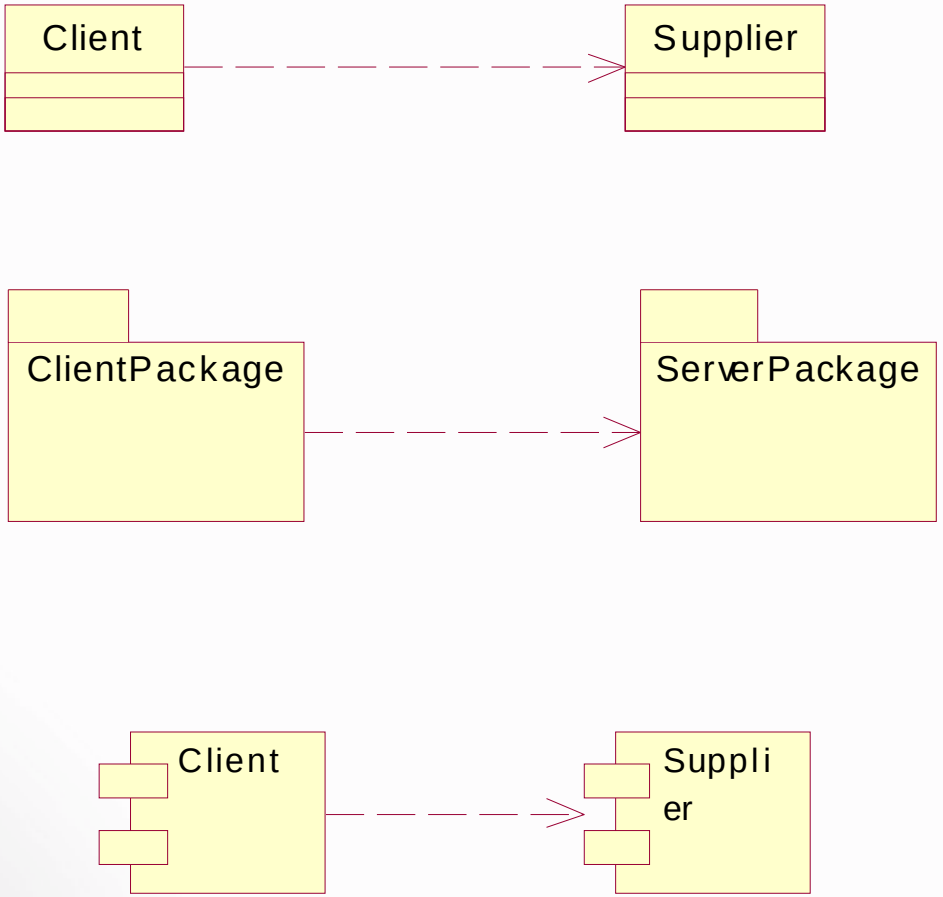
- 是类与类之间的连接，表示一个类依赖于另一个类的定义。
- 例如如果A依赖于B，则B是A中的一个方法中的局部变量，或者B是A中方法的参数，或者A调用了B中的静态方法。
- 表示方法：虚线 + 箭头
- 举例：警察指挥交通、汽车、自行车



依赖关系

Ü 依赖关系描述两个模型元素(类、用例等)之间的语义连接关系：其中一个模型元素是独立的，另一个模型元素不是独立的，它依赖于独立的模型元素，如果独立的模型元素改变了，将影响依赖于它的模型元素。

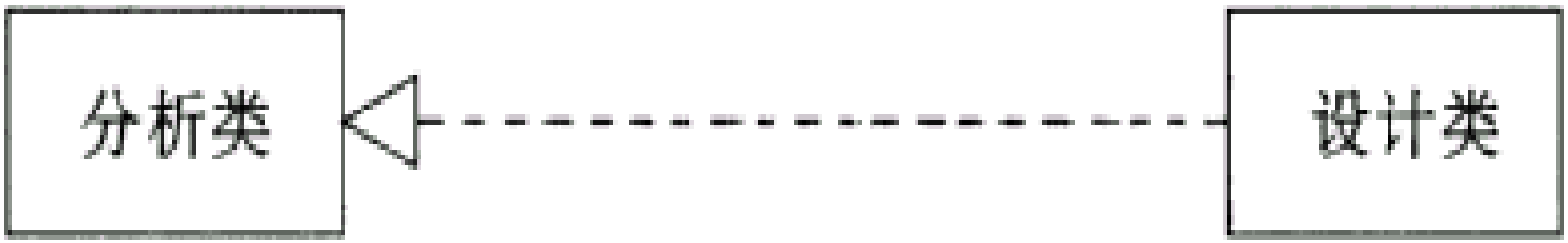
友元依赖关系



实现关系

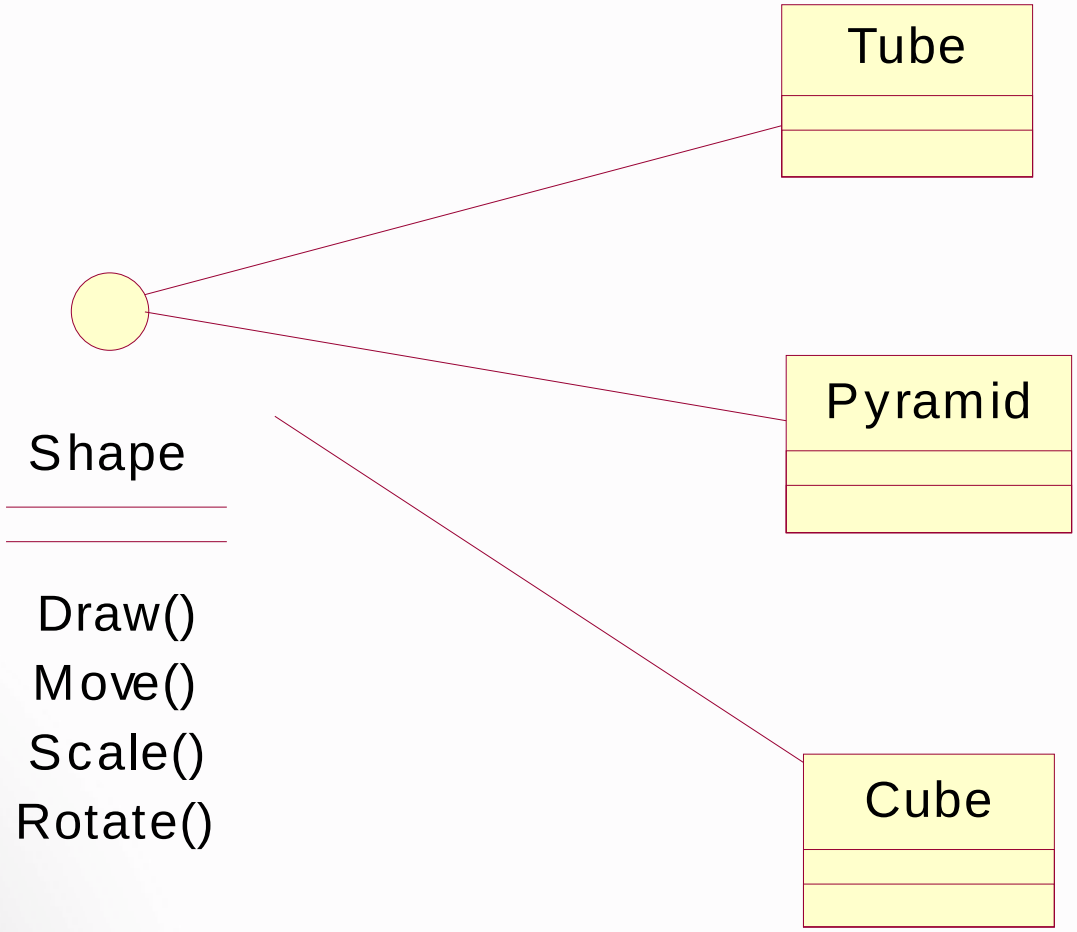
- 当对同一事物在不同抽象层次上描述时，这些描述之间具有实现关系。
- 实现是UML中的术语，表示对事物更详细一层的描述。假设两个元素A和B描述同一个事物，它们的区别是抽象层次不同，如果B是在A的基础上的更详细的描述，则称B实现了A，或称A实现成了B。
- 实现的图示符号为由元素B指向元素A的、一端为空心三角的虚线(不是实线)。实现主要用于模型之间的合作，表示各开发阶段不同抽象层次的模型的相关性，常用于跟踪模型的演变。
- 举例：数据库的原子操作

实现关系



实现关系

Interface



协作图

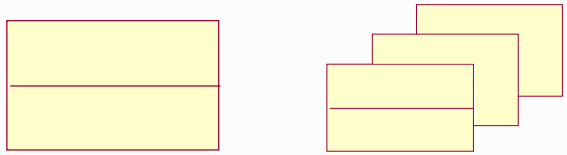
动态协作，显示对象和它们之间的关系（上下文相关）

Sequence diagrams and collaboration

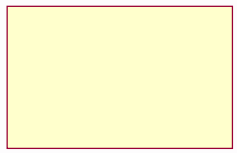
diagrams express similar information, but show it indifferent ways.

协作图元素

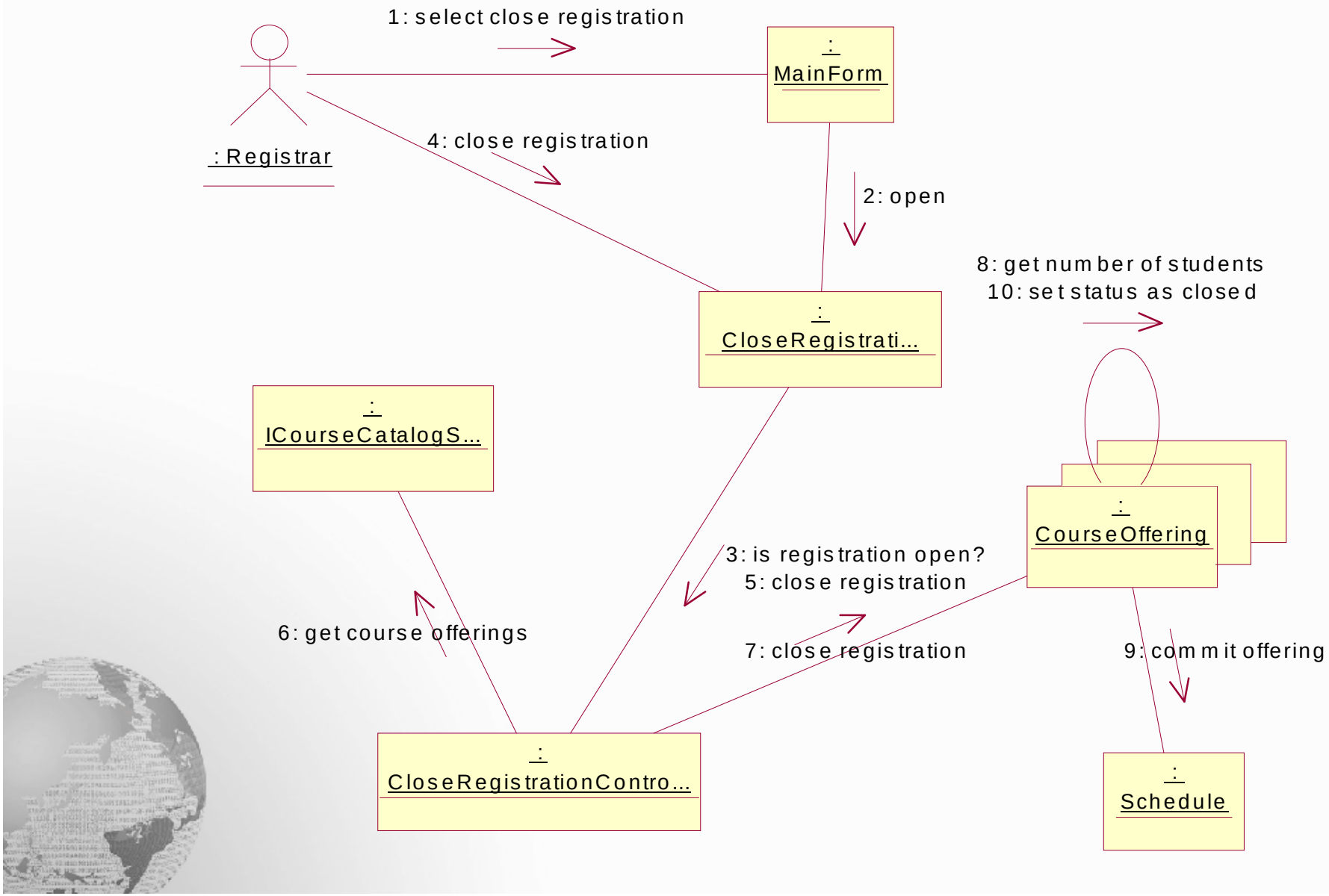
- Object



- Class instance



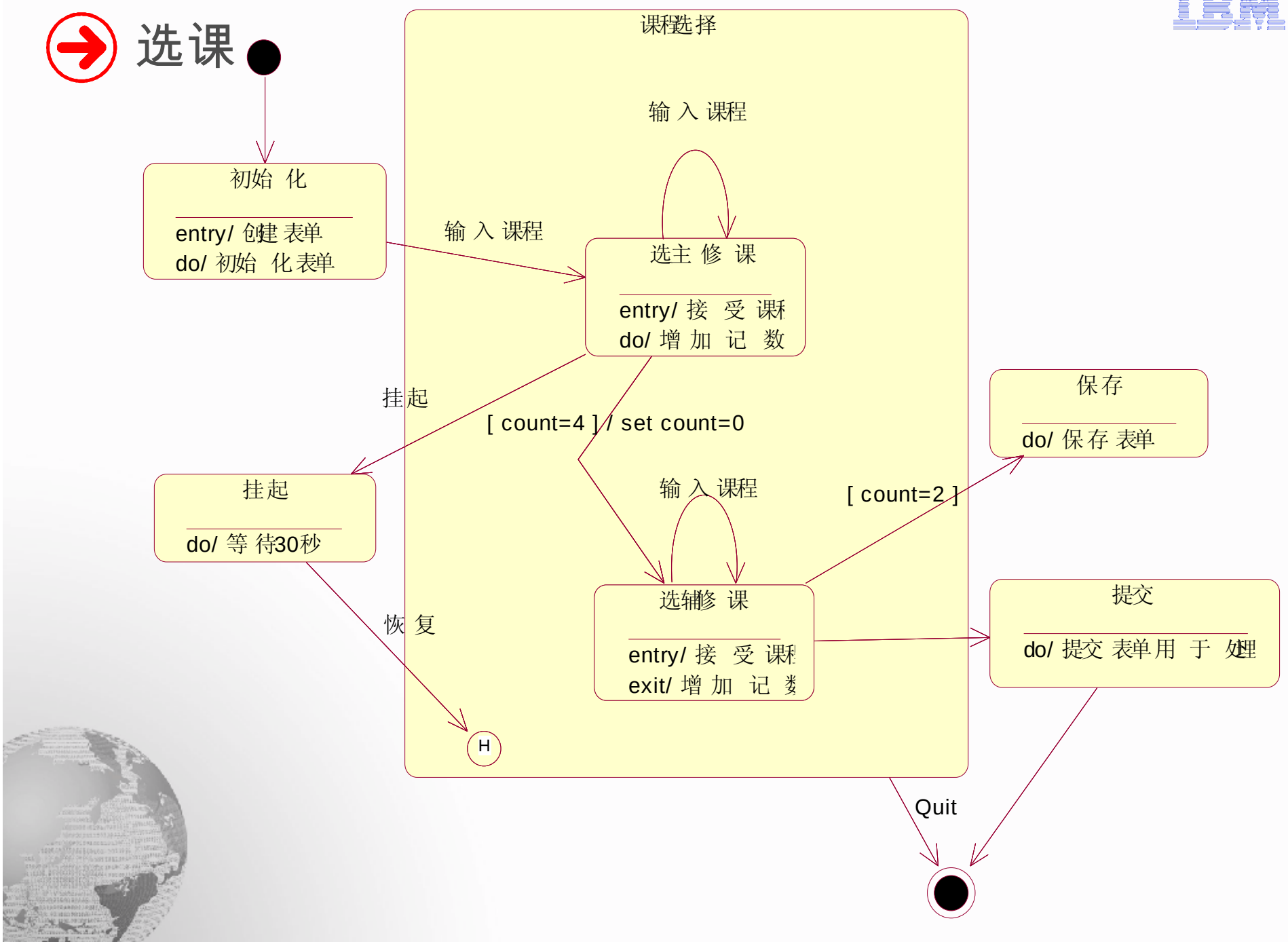
完成课程登记

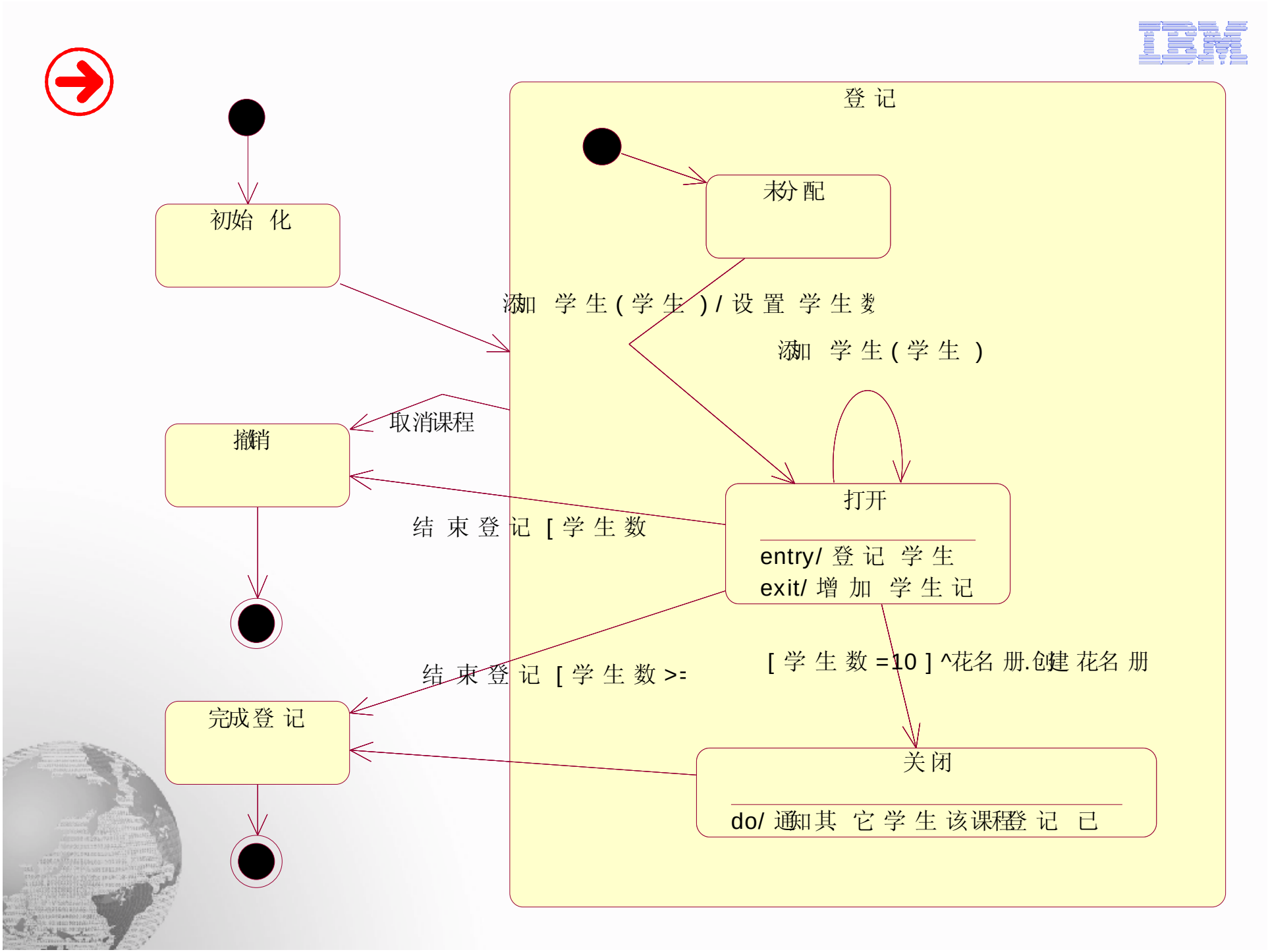


状态图

显示类的所有对象可能具有的状态，以及引起状态变化的事件，可以为系统描述整体状态图

选课





活动图

- Ü 反映一个连续的运动流某个操作的执行时的活动状况
- Ü 描述一个操作执行过程中（操作实现的实例化）所完成的工作（动作）。
- Ü 描述对象内部的工作。
- Ü 显示如何执行一组相关的动作，以及这些动作如何影响它们周围的对象。
- Ü 显示用例的实例是如何执行动作以及改变对象状态。
- Ü 说明一次商务活动中的工人（角色）、工作流、组织和对象是如何工作的。
- Ü 动作、转移、泳道、对象、信号

组件图

- Ü 组件包含逻辑类及逻辑类的实现信息
- Ü 用来反映代码的物理结构源组件、二进制组件、可执行组件链接时的组件、运行时的组件
- Ü Main programs，主程序
- Ü Packages，构件包
- Ü Subprograms，子程序
- Ü Tasks，独立线程
- Ü EXE，可执行文件
- Ü DLL，动态连接库

部署图

- 显示系统中软件和硬件的物理架构
- 节点、连接、组件、对象

UML的图与模型的对应关系

- 用例模型--用用例图、顺序图、协作图、状态图和活 动图描述。
- 分析、设计模型--用类图和对象图、顺序图、协作图、状态图和活动图描述。
- 实现模型--可用组件图、顺序图和协作图描述。
- 实施模型--配置图
- 测试模型--测试模型引用了所有其他模型，所以它使用他们对应的所有图。

UML图与项目过程

- Ü1项目初始需求调研阶段
  - è用例图：利用业务用例和系统用例来框定需求
  - è活动图：作为用例描述的辅助
  - è状态图：作为用例描述的辅助
- Ü2项目初始可行性文档阶段
  - è部署图：描述系统的硬件布局
- Ü3项目初始基础设施搭建阶段
  - è组件图：规范开发人员各种环境的组件描述

UML图与项目过程

- 4建模分析阶段
  - 用例图：利用业务用例描述系统
  - 活动图：作为用例描述的辅助
  - 状态图：作为用例描述的辅助
  - 协作图：作为用例实现的描述，以对象的消息链互动来展示系统如何工作。
  - 序列图：作为用例实现的描述，以对象的生命线和消息焦点来展示系统如何工作。

UML图与项目过程

- 5建模设计阶段
  - 用例图：利用业务用例描述系统
  - 类图：作为面向对象开发的设计
  - 协作图：作为用例实现的描述，以类的方法互动来设计系统如何工作需要类图中图素的辅助，更接近图代码。
  - 序列图：作为用例实现的描述，以类的生命线和消息焦点来设计系统如何工作，需要类图图素的辅助，更接近代码。