

6-2 面向对象的基本概念

2019年5月26日

22:15

3 对象之间的五类关系

4 UML建模语言简述

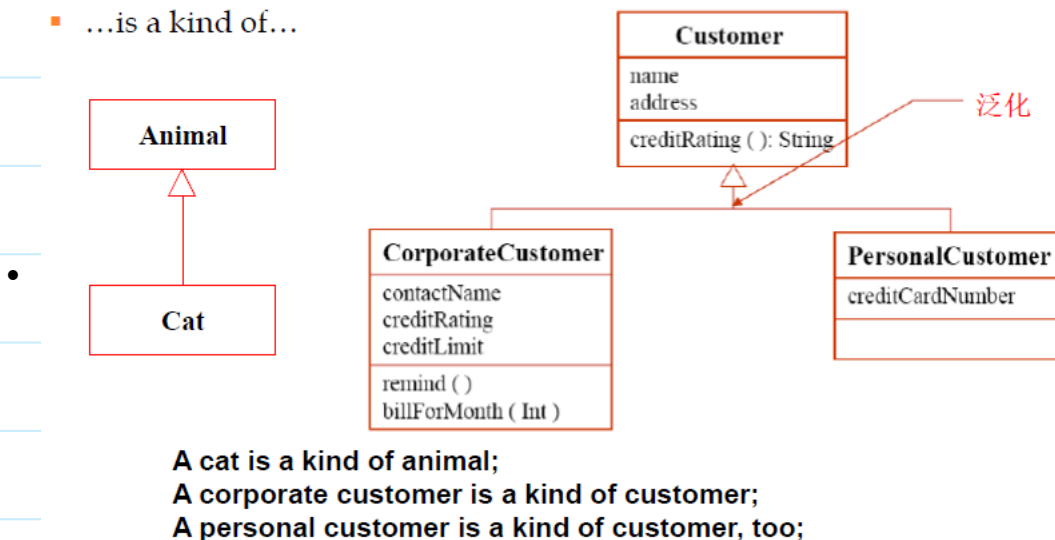
3 对象之间的五类关系

对象之间的联系

- 分类结构：继承/泛化关系一般与特殊的关系
- 组成结构：聚合与组合关系部分与整体的关系
- 实例连接：关联关系对象之间的静态联系
- 消息连接：依赖关系对象之间的动态通信联系

分类结构：继承/泛化关系

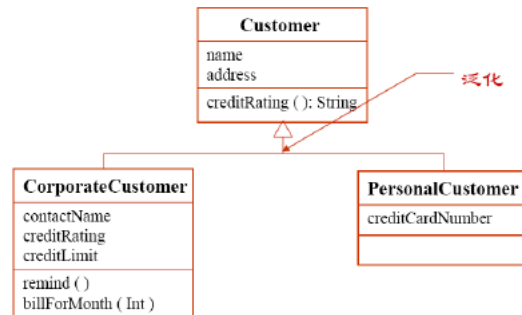
- 分类结构：表示的是事物的“一般 - 特殊”的关系，也称为泛化
 - ...is a kind of...



```
class Customer {
    String name;
    String address;
    String creditRating() {};
}
```

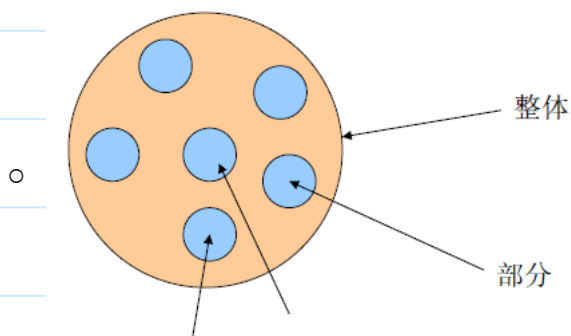
```
class CorporateCustomer : Customer {
    String contactName;
    String creditRating;
    String creditLimit;

    void Remind() {};
    void billForMonth(int bill) {};
}
```

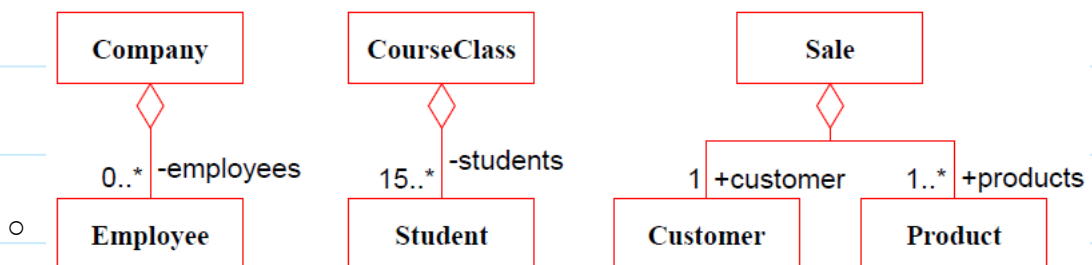


组成结构：聚合与组合关系

- 组成结构：表示对象类之间的组成关系，一个对象是另一个对象的组成部分，即“部分整体”关系。
- 分为两个子类：
 - 聚合(Aggregation)：整体与部分在生命周期上是独立的(…owns a…);
 - 组合(Composition)：整体与部分具有同样的生命周期(…is part of…);



- 聚合(Aggregation)：整体与部分在生命周期上是独立的



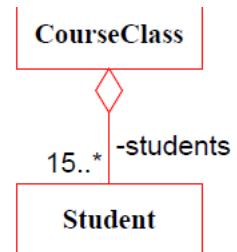
**A company owns zero or multiple employees;
 A course's class owns above 15 students;
 An Sale owns a customer and a set of products;**

定义两个类:

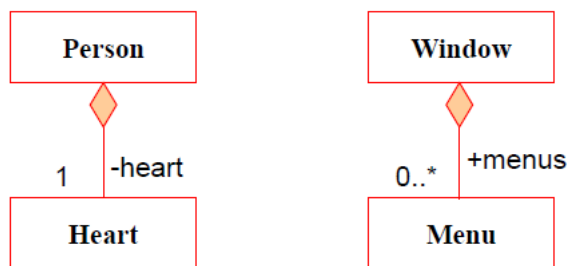
```
class Student {}  
  
class CourseClass {  
    ...  
    private Student[] students;  
    public addStudent (Student s) {  
        students.append(s);  
    }  
    ...  
}
```

使用时的代码:

```
Student a = new Student ();  
Student b = new Student ();  
Student n = new Student ();  
  
CourseClass SE = new CourseClass();  
SE.addStudent (a);  
SE.addStudent (b);  
SE.addStudent (n);
```

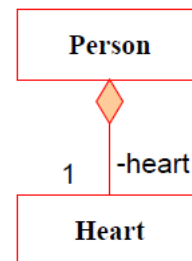


- 组合(Composition): 强调整体与部分具有同样的生命周期;



A heart is part of a person;
A menu is part of a window;

```
class Heart {}  
  
class Person {  
    ...  
    private Heart heart = new Heart();  
    ...  
}
```

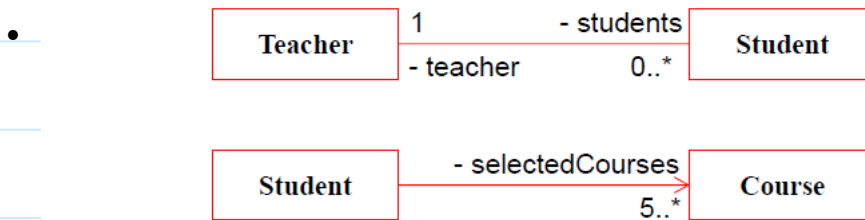


实例连接: 关联关系

- 实例连接: 表示对象之间的静态联系, 通过对对象的属性之间的联系加以展现。
 - 对象之间的实例连接称为链接(Link), 存在实例连接的对象类之间的联系称为关联(Association)。
 - ...has a...

■ 例如：

- “教师”与“学生”是两个类，它们之间存在“教 - 学”关系。
- “学生”与“课程”是两个类，它们之间存在“学习 - 被学习”的关系。

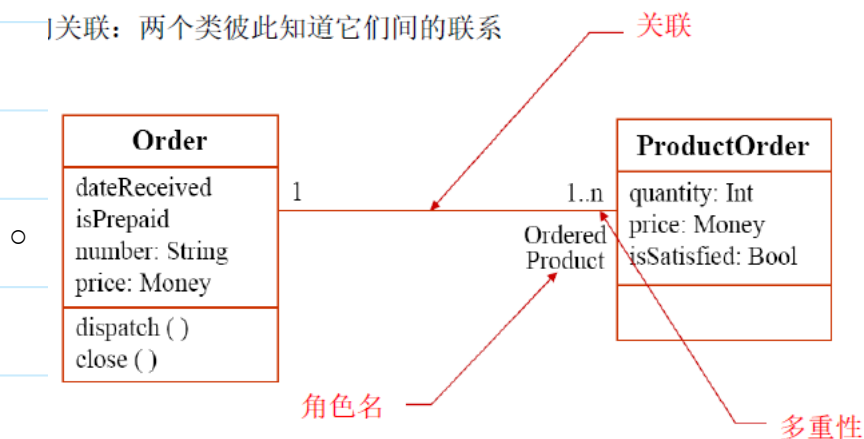


- 关联具有多重性(重数)：表示可以有多少个对象参与该关联

- 关联具有方向性：

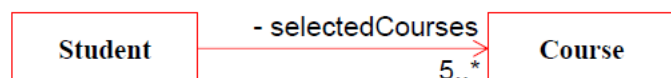
- 单向关联：两个类是相关的，但是只有一个类知道这种联系的存在
- 双向关联：两个类彼此知道它们间的联系

双向关联：两个类彼此知道它们间的联系



```

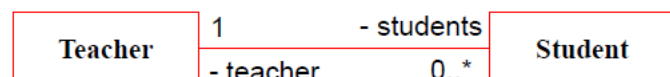
class Course {}
class Student {
    private Course [ ] selectedCourses;
}
    
```



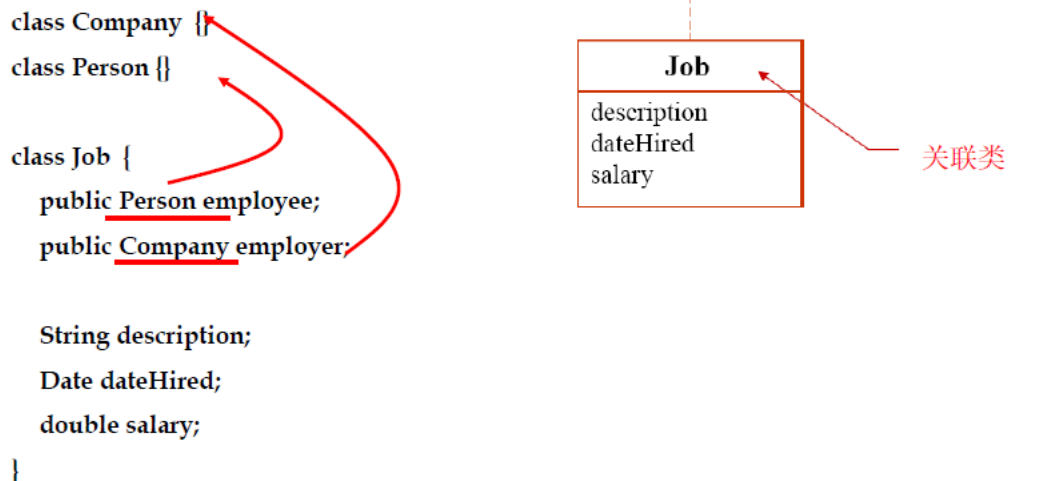
- ```

class Teacher {
 private Student [] students;
}
class Student {
 private Teacher teacher;
}

```

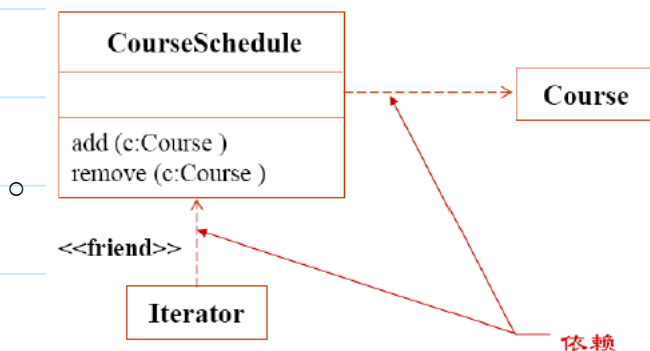


■ 关联类：

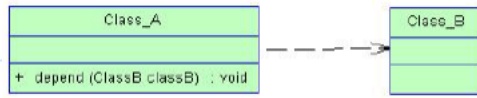


消息连接：依赖关系

- 消息连接
  - 消息连接是对象之间的通信联系，它表现了对象行为的动态联系。
  - 一个对象需要另一个对象的服务，便向它发出请求服务的消息，接收消息的对象响应消息，触发所要求的服务操作。
- 消息连接也称为“依赖关系” (Dependency)。



- 依赖(Dependency): ...use a...
  - 依赖是一种使用关系，一个类A使用到了另一个类B，而这种使用关系是偶然性的、临时性的、非常弱的，但是B类的变化会影响到A。
- 类的依赖可能由各种原因引起，例如：
  - 一个类是另一个类的某个操作的参数
  - 一个类在另一个类的某个操作中被使用



```

class Air {}

class Human {
 public void breath(Air air) {}
}

```



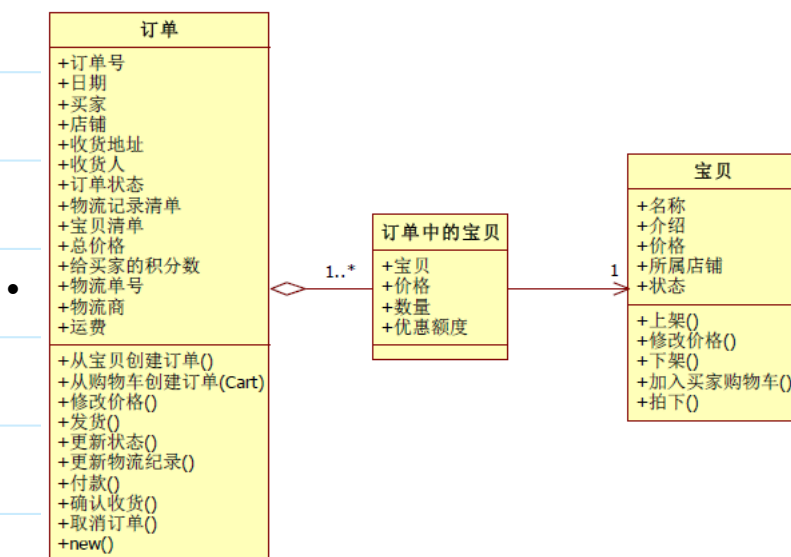
“依赖”关系怎么识别？

- 依赖(Dependency): 最弱的一种类间关系, 临时的、局部的。
- 除非类A的某个操作op使用了类B, 否则不要随意设置依赖关系:
  - op有某个参数param或返回值的类型为B;
  - op的内部业务逻辑中使用了B;
- 判断标准: 作用域范围
  - B是否只在A中的某个操作的作用域范围内才被A所使用? 若是, 则A依赖于B
  - 若A的某个属性的数据类型是B, 那意味着B对A的全部操作都是有意义的, 那么A和B之间至少是关联关系。

如何识别出“关联类”？

- 关联类: 当两个实体类之间产生m:n关联关系, 且这种关联关系导致了新的信息产生并需要对其进行管理时, 就需要关联类。
- 这里所谓的“新的信息”是指: 如果两个实体类不关联在一起, 这些信息就不会存在
- 例如: 订单和商品这两个实体类, 按照常规理解, 似乎是聚合关系, 一个订单包含多种商品。但是在这种聚合发生时, 产生了某些新信息(该商品在该订单中的价格, 可能与卖家给商品设定的价格不同; 本次购买的数量)。这些新信息, 既不属于订单, 也不属于商品本身, 故而拆分出一个关联类“订单项”:
  - 订单: 订单号、买家、卖家、总价格、收货地址、物流流转记录(list)、订单项集合(list)、etc;
  - 商品: 名字、所属卖家、设定价格、etc;
  - 订单项: 商品、本次购买价格、数量。

如何识别出“关联类”？



## 关于“聚合”与“关联”的区别

- 本质上，聚合都是特殊的“关联”关系，只不过关系的强度更大。若两个类之间是聚合关系，其实是可以关联关系来表示的。

例如：

- 对“购物车”和“商品”两个类而言，可以说“多个商品对象聚合成了购物车对象”，商品是购物车的一部分；
- 也可以说“购物车 has some 商品”，二者是多对多的关联关系，一个购物车里有0..\*个商品，一个商品可以在0..\*个购物车内出现，商品对象无需维护自己出现在了哪些购物车对象中，但购物车对象一定要知道自己内部有哪些商品对象，故这是一个从购物车类指向商品的单项关联。
- 何时用聚合，何时用关联？
  - 一个经验：判断两个类的“地位”是否对等。
  - 若二者对等，用关联关系（例如商品和卖家，二者非常独立存在的，彼此地位相同，更适合用关联而不是聚合）；
  - 若二者明显不对等，用聚合关系（例如“买家对商品的评价”和“商品”，商品的地位更高，评价往往看作它的一部分）。

## “依赖”关系怎么识别？

例如：

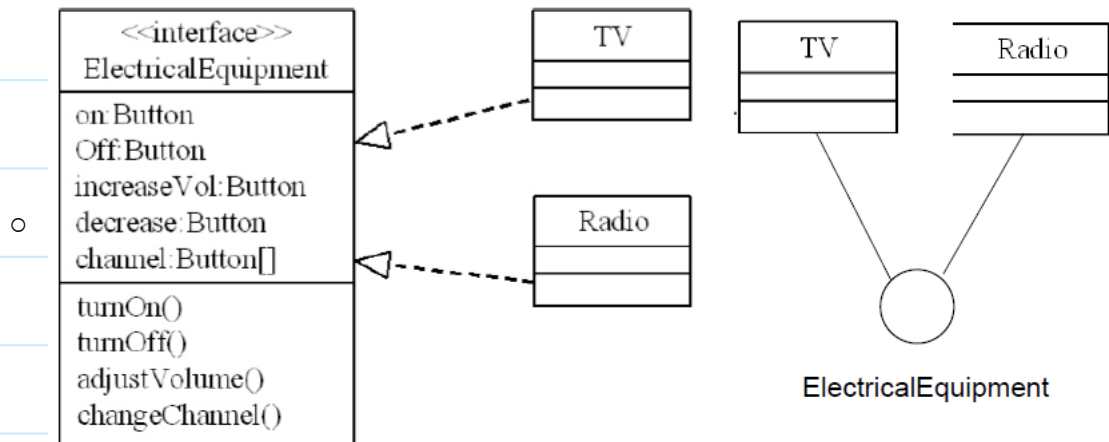
- “订单”类中有一个操作“从购物车生成订单()”，它有一个参数，类型是“购物车”类，该操作在执行时，根据传入的“购物车”对象读取产品信息来生成多个“订单项”对象。这个操作执行完之后，“订单”类和“购物车”类就再无关系，故前者依赖于后者。

一个好理解的例子：

- 一个“路人”类和一个“时钟”类，后者的所有属性和操作均与前者无关，而前者在走到时钟面前时抬头看了看时钟以获取时间，故“路人”类有一个操作“抬头看时间”。只有在这个操作范围内，路人需要与时钟发生关系，而其他操作如“走路”、“吃饭”等均与“时钟”无关。故“路人”依赖于“时钟”。

## 接口连接：实现关系

- 实现关系(realization): 是泛化关系和依赖关系的结合, 通常用以描述一个接口和实现它们的类之间的关系;
- “棒棒糖” 另一种形式。

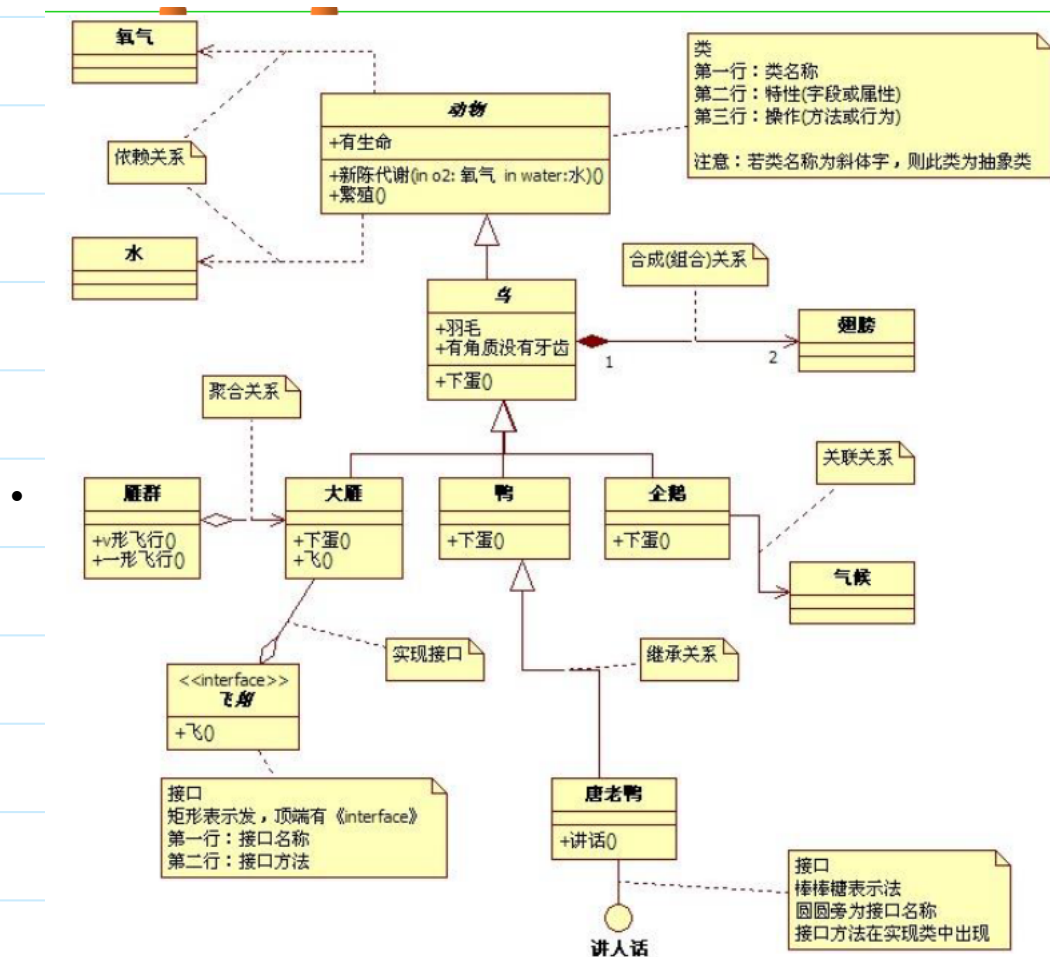


## 小结：对象之间的联系

- 继承/泛化：一般与特殊的关系——isakindof
- 组合：部分与整体的关系，彼此不可分——ispartof
- 聚合：部分与整体的关系，但彼此可分——ownsa
- 关联：对象之间的长期静态联系——hasa
- 依赖：对象之间的动态的、临时的通信联系——usea
- 类间联系的强度：继承>>>组合>>聚合>>关联>>>依赖

## 面向对象概念的一个综合例子





## 4 UML建模语言简述

建模是一种设计技术

- 模型是某个事物的抽象，其目的是在构建这个事物之前先理解它
  - 在构建物理实体之前先验证
  - 通过抽象降低复杂度
  - 可视化，便于与客户和其他小组成员交流
  - 为维护和升级提供文档
- 建模的主要目的是为理解，而非文档
- 软件开发中建模的过程
  - 分析建模
  - 设计建模
  - 实现建模
  - 部署建模

面向对象建模的三个不同视角

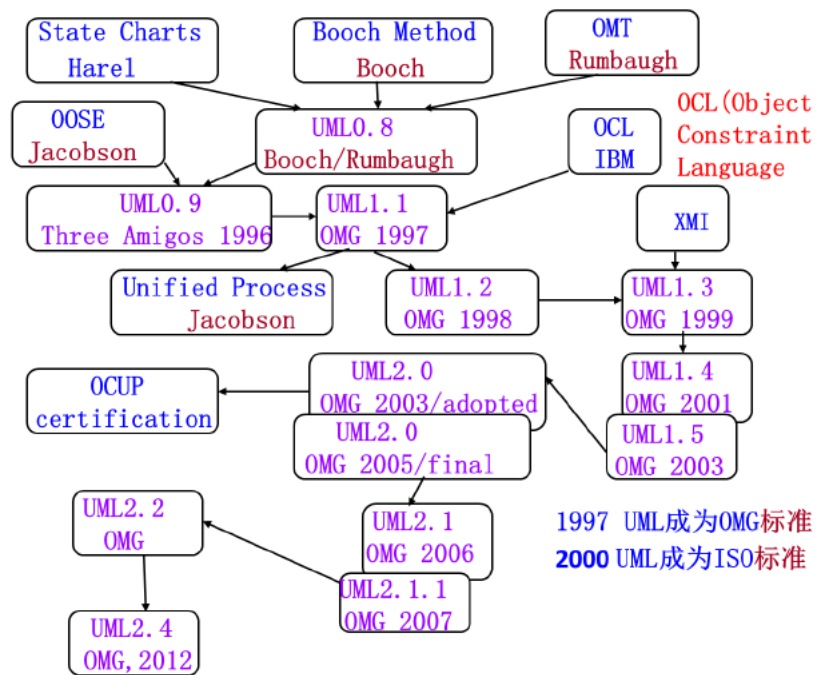
- 类模型：表示系统静态的、结构化的“数据”层面

- 描述系统中对象的结构：它们的标识、属性和操作
- 描述与其他对象的关系
- 状态模型：表示对象时序的、行为的“控制”层面
  - 标记变化的事件
  - 界定事件上下文的状态
  - 一个类有一个状态图
- 交互模型：表示独立对象的协作“交互”层面
  - 系统行为如何完成
  - 对象间如何协作

## 什么是UML?

- 统一建模语言 (Unified Modeling Language, UML) 是描述、构造和文档化系统制品的可视化语言 (OMG03a) 。
  - 由Booch、Rumbaugh和Jacobson合作创建
  - 统一了主流的面向对象的分析设计的表示方法
- UML的定义包括UML语义和UML表示法两个部分。
  - UML语义：UML对语义的描述使开发者能在语义上取得一致认识，消除了因人而异的表达方法所造成的影响。
  - UML表示法：UML表示法定义UML符号的表示法，为开发者或开发工具使用这些图形符号和文本语法为系统建模提供了标准。
- 面向对象建模的图形化表示法的标准
  - UML不是可视化程序设计语言，而是一个可视化的建模语言
  - UML不是OOA/D，也不是方法，它只是图形表示工具

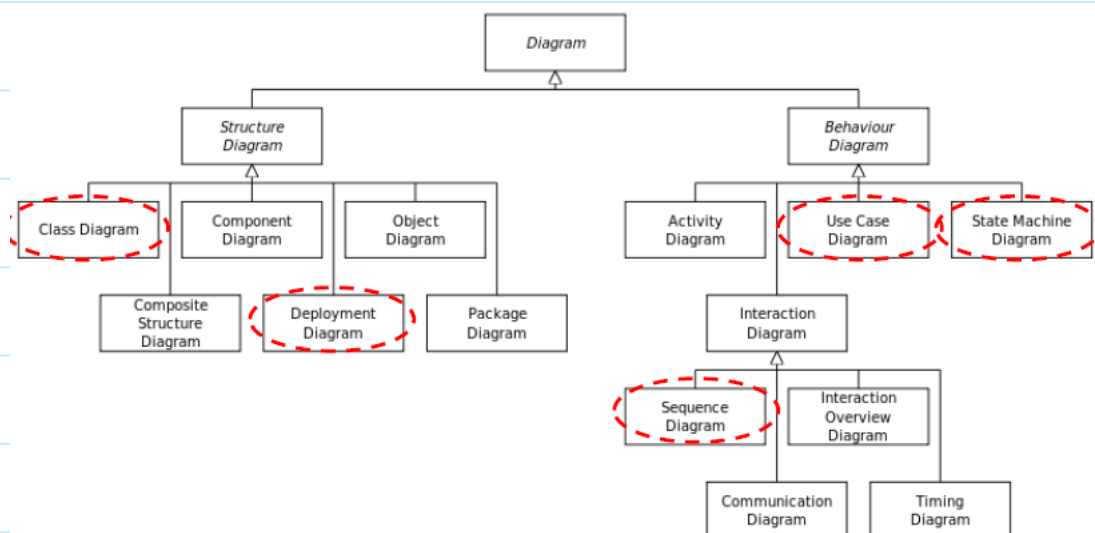
## UML的发展历史



## 应用UML的三种方式

- UML作为草图：非正式、不完整的，用于探讨问题和交流
- UML作为蓝图：相对详细的设计图，用于代码生成和逆向工程
- UML作为编程语言：用UML完成系统可执行规格说明，模型驱动体系结构（MDA）的应用方式，尚在发展阶段

## UML模型



## UML视图和图

|      | 视图    | 图         | 主要概念                    |
|------|-------|-----------|-------------------------|
| 结构分类 | 静态视图  | 类图/对象图/包图 | 类、对象、包、关联、泛化、依赖关系、实现、接口 |
|      | 物理视图  | 构件图       | 构件、接口、依赖关系、实现           |
|      |       | 部署图       | 节点、构件、依赖关系、位置           |
| 行为分类 | 用例视图  | 用例图       | 用例、参与者、关联、扩展、包括、用例泛化    |
|      | 状态机视图 | 状态图       | 状态、事件、转换、动作、            |
|      | 活动视图  | 活动图       | 状态、活动、完成转换、分叉、结合        |
|      | 交互视图  | 顺序图       | 交互、对象、消息、激活             |
|      |       | 协作图       | 协作、交互、协作角色、消息           |

## UML视图

- 结构分类：描述了系统中的结构成员及其相互关系
  - 静态视图对应用领域中的概念以及与系统实现有关的内部概念建模
  - 物理视图对应用自身的实现结构建模。
    - 物理视图有两种：实现视图和部署视图
    - 实现视图为系统的构件模型及构件之间的依赖关系。
    - 部署视图描述位于节点实例上的运行构件实例的安排。
- 行为分类：描述了系统的功能和行为
  - 用例视图是外部用户所能观察到的系统功能的模型图
  - 状态机视图是一个类对象所可能经历的所有历程的模型图。
  - 活动视图是状态机的一个变体，用来描述执行算法或工作流程中涉及的活动
  - 交互视图描述了执行系统功能的各个角色之间相互传递消息的顺序关系。

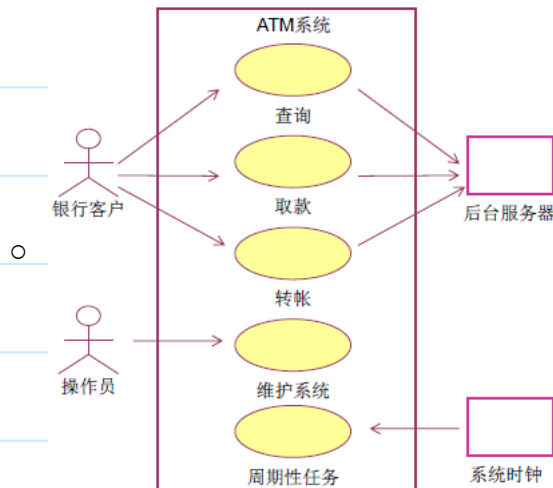
## UML图

- 类图/对象图：描述系统的各个对象类型以及存在的各种静态关系
- 用例图：描述用户与系统如何交互
- 构件图：构件间的组织结构及链接关系
- 部署图：制品在节点上的部署
- 状态图：事件在对象的周期内如何改变状态
- 活动图：过程及并行行为
- 顺序图：对象间的交互；强调顺序
- 协作图：对象间交互，重点在对象间链接关系

## 各UML图及特征

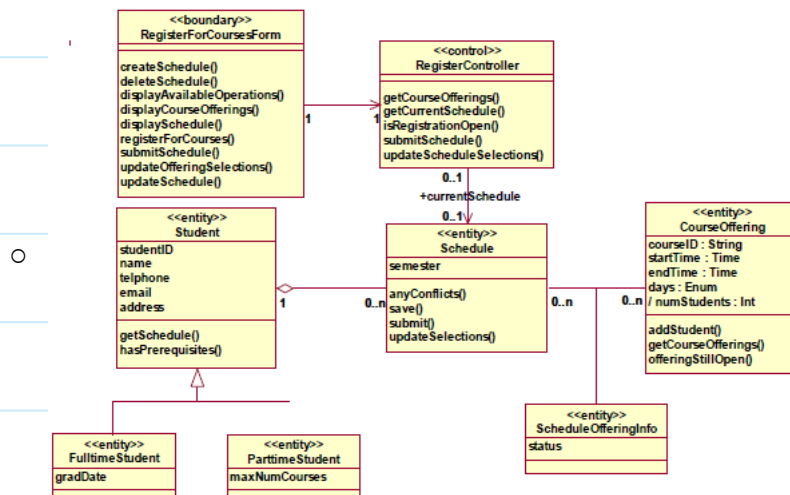
- 用例图 (UseCaseDiagram)

- 描述用户与系统如何交互。从用户角度描述系统功能，是用户所能观察到的系统功能的模型图，用例是系统中的一个功能单元。



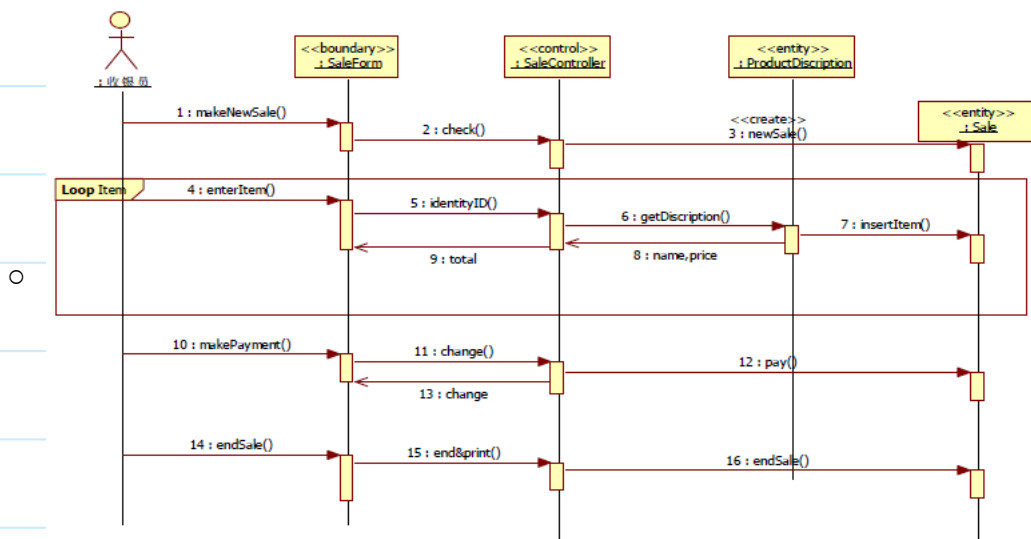
- 类图(ClassDiagram)/对象图(ObjectDiagram)

- 描述系统的各个对象类型以及存在的各种静态关系。对象图是类图的实例，几乎使用与类图完全相同的标识。它们的不同点在于对象图显示类的多个对象实例，而不是实际的类。



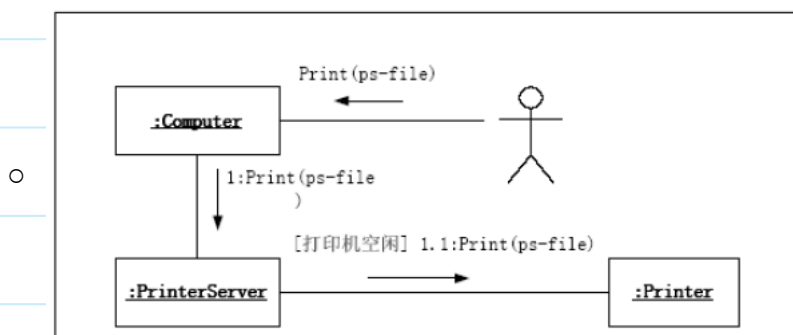
- 顺序图(SequenceDiagram)

- 顺序图显示对象之间的动态合作关系，它强调对象之间消息发送的顺序，同时显示对象之间的交互。



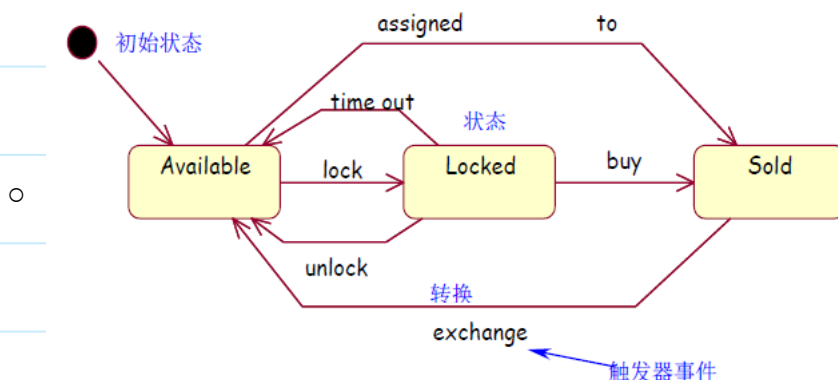
## • 协作图(CollaborationDiagram)

- 对象间交互，重点在对象间链接关系；
- 协作图描述对象间的协作关系，协作图跟顺序图相似，显示对象间的动态合作关系。除显示信息交换外，协作图还显示对象以及它们之间的关系
- 协作图的一个用途是表示一个类操作的实现



## • 状态图(StateChartDiagram)

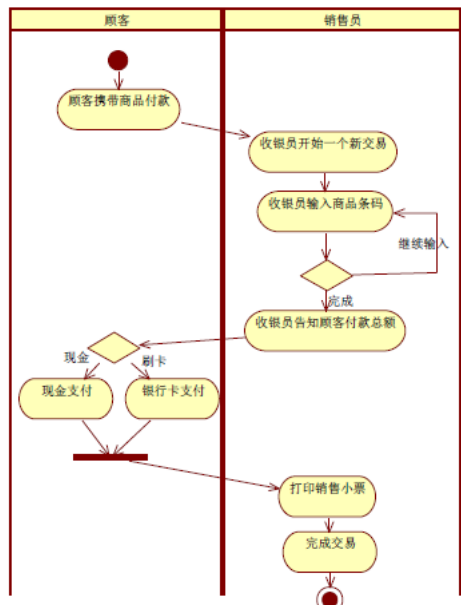
- 事件在对象的周期内如何改变状态；
- 状态图是一个类对象所可能经历的所有历程的模型图。状态图由对象的各个状态和连接这些状态的转换组成



## • 活动图(ActivityDiagram)

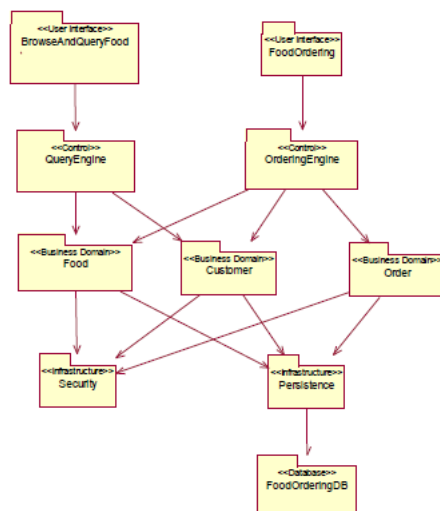
- 活动图是状态图的一个变体，用来描述执行算法的工作流程中涉及的活动
- 活动图描述了一组顺序的或并发的活动

- 过程及并行行为



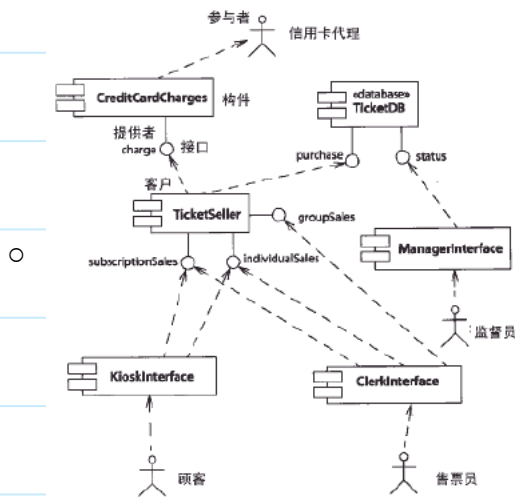
- 包图(PackageDiagram)

- 包图是在UML中用类似于文件夹的符号表示的模型元素的组合。
- 把在语义上接近且倾向于一起变化的类组织在一起形成“包”，既可控制模型的复杂度，有助于理解，而且也有助于按组来控制类的可见性；



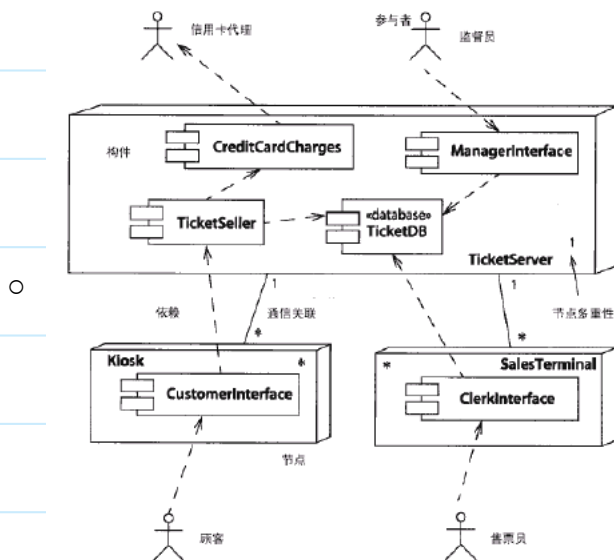
- 构件图(ComponentDiagram)

- 构件间的组织结构及链接关系
- 构件图为系统的构件建模型—构件即构造应用的软件单元—还包括各构件之间的依赖关系，以便通过这些依赖关系来估计对系统构件的修改给系统可能带来的影响





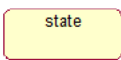






- 部署图(DeploymentDiagram)

- 部署视图描述位于节点实例上的运行构件实例的安排。节点是一组运行资源，如计算机、设备或存储器。这个视图允许评估分配结果和资源分配



## UML语法描述

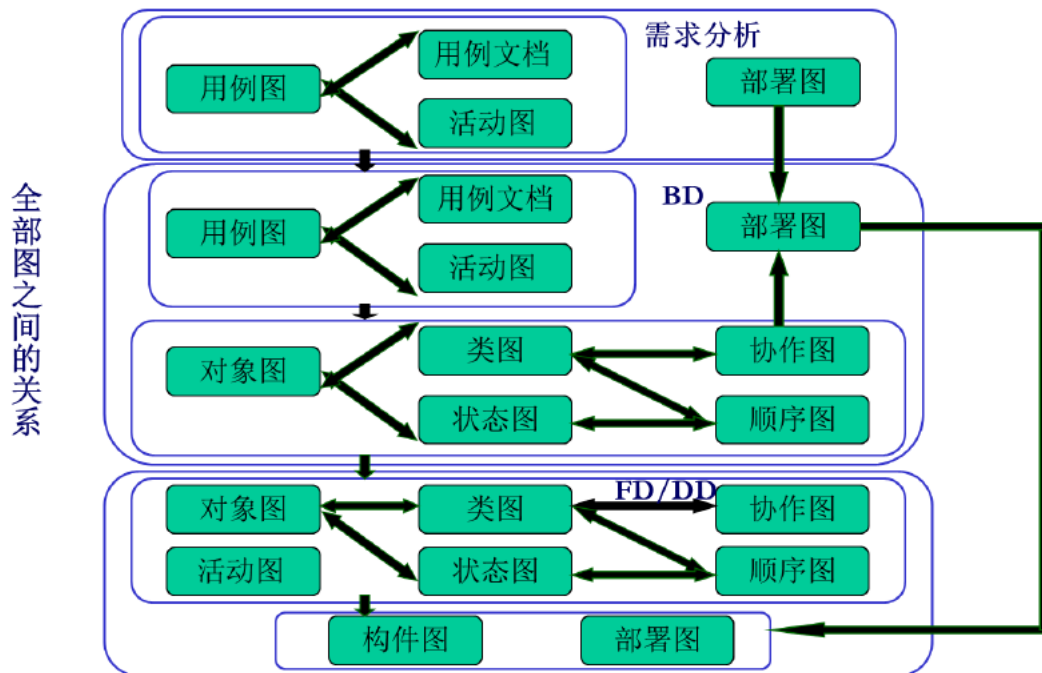
|     |                                                                                                                                                                                                                                                               |                                                                                     |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| 类   | 是对一组具有相同属性、相同操作、相同关系和相同语义的对象的描述                                                                                                                                                                                                                               |  |
| 对象  |  ,   |                                                                                     |
| 接口  | 是描述了一个类或构件的一个服务的操作集                                                                                                                                                                                                                                           |  |
| 协作  | 定义了一个交互，它是由一组共同工作以提供某种协作行为的角色和其他元素构成的一个群体                                                                                                                                                                                                                     |  |
| 用例  | 是对一组动作序列的描述                                                                                                                                                                                                                                                   |  |
| 主动类 | 对象至少拥有一个进程或线程的类                                                                                                                                                                                                                                               |  |
| 构件  | 是系统中物理的、可替代的部件                                                                                                                                                                                                                                                |  |
| 参与者 | 在系统外部与系统直接交互的人或事物                                                                                                                                                                                                                                             |  |

|      |                                 |                                                                                       |
|------|---------------------------------|---------------------------------------------------------------------------------------|
| 节点   | 是在运行时存在的物理元素                    |  |
| 交互   | 它由在特定语境中共同完成一定任务的一组对象间交换的消息组成   |  |
| 状态机  | 它描述了一个对象或一个交互在其生命期内响应事件所经历的状态序列 |  |
| 包    | 把元素组织成组的机制                      |  |
| 注释事物 | 是UML模型的解释部分                     |  |
| 依赖   | 一条可能有方向的虚线                      |  |
| 关联   | 一条实线，可能有方向                      |  |
| 泛化   | 一条带有空心箭头的实线                     |  |
| 实现   | 一条带有空心箭头的虚线                     |  |



## UML图的适用场合

- 需求获取
  - 用例图：建立应用场景，如何使用系统
  - 活动图：明确组织机构的工作流程，软件如何与人交互
- 分析与设计
  - 从概念视角绘制的概念类图：用于领域分析
  - 软件视角的设计类图：设计软件中的类及相互联系
  - 常用用例的顺序图：重要用例的对象交互顺序
  - 构件图：构件设计及构件间关系
  - 包图：设计软件的组织结构
  - 具有复杂生命周期的类的状态图
- 实施
  - 部署图：描述软件系统在硬件平台和运行环境中的物理分布



## 主流的UML建模工具

- 主流的UML建模工具

- IBM Rational Rose / IBM rational software architect(RSA)

- Enterprise Architect

- Visual Paradigm

- Microsoft Visio

- Sybase Powerdesigner

- - Together

- StarUML

- Eclipse

- Microsoft Visual Studio

- JBuilder