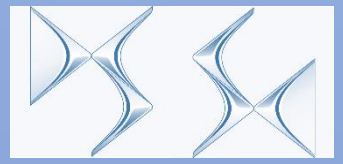
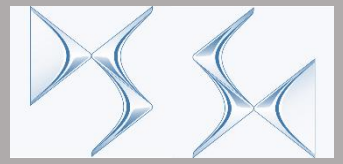


软件工程 模型与方法
Models & Methods of SE

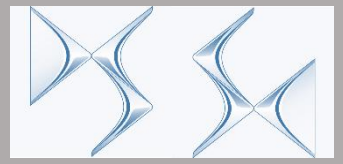
面向对象需求分析方法

老师邮箱: dxiao@bupt.edu.cn

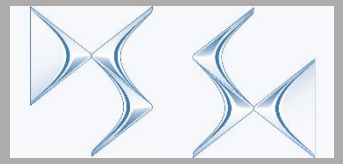


本章内容

- UML：统一建模语言简介
- 面向对象的需求分析建模
  - 领域建模
    - 领域模型的定义和表示
    - 业务背景：概念类及关系，类图
    - 业务流程：活动图
  - 用例建模
    - 用例图
    - 用例说明
    - 系统顺序图
    - 操作契约



OO建模方法的发展历程

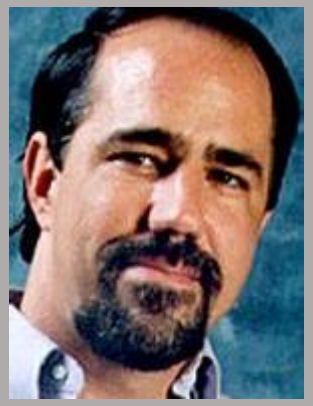


UML 简介

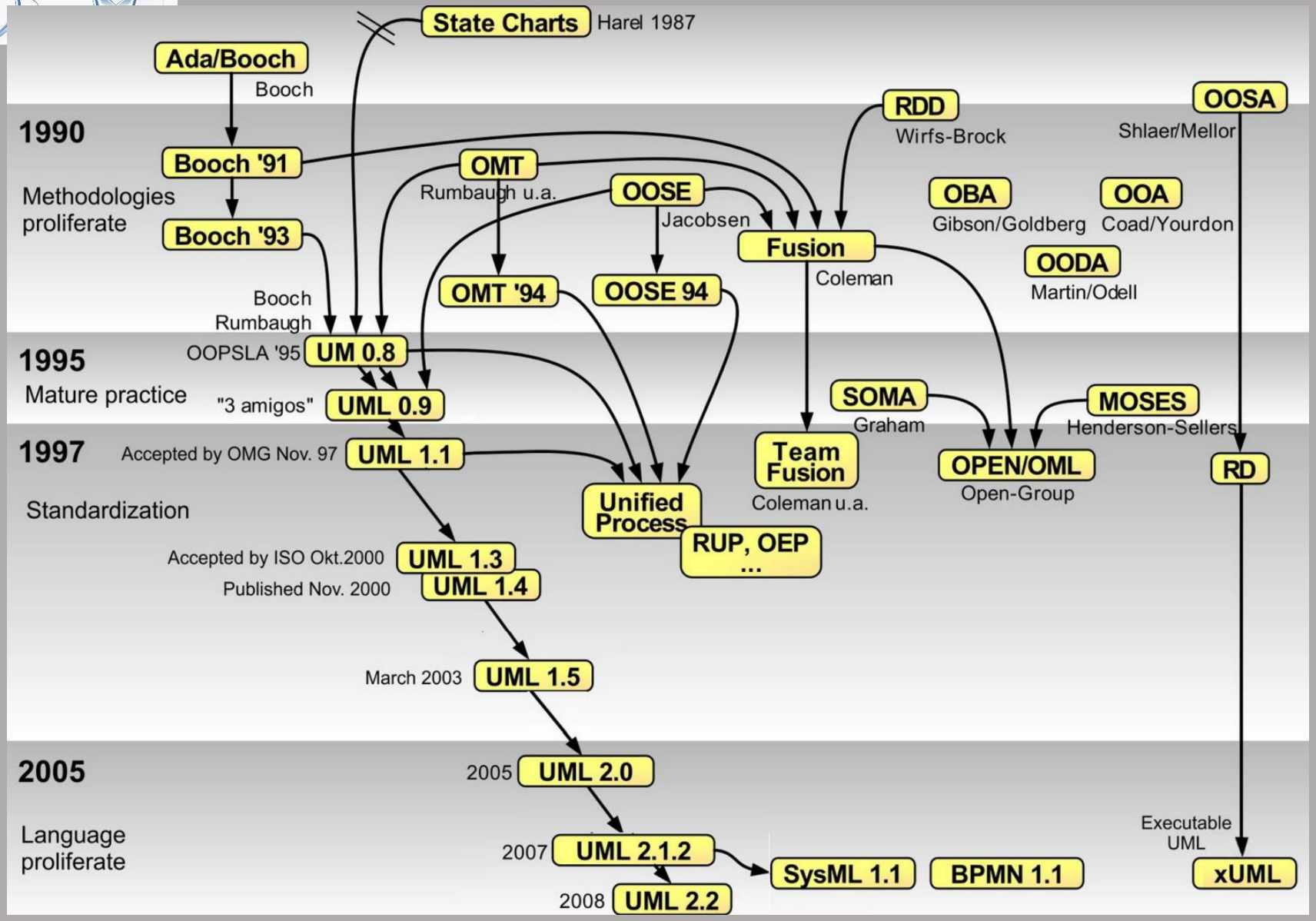
- UML的发展历程
- UML概述
- UML中的组成
- UML中的图

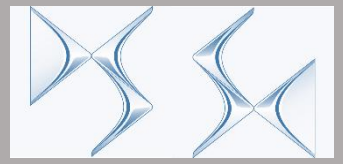
UML的发展历程

- 统一建模语言UML是由Grady Booch、Ivar Jacobson和James Rumbaugh发起，在Booch方法、OOSE方法和OMT方法基础上，广泛征求意见，集众家之长，几经修改而成的一个面向对象的建模语言。
- 该建模语言得到了“UML 伙伴联盟”的应用，并得到工业界的广泛支持，由OMG 组织采纳作为业界标准，是软件界第一个统一的建模语言。



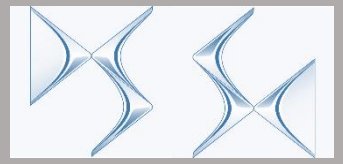
UML的发展历程





UML发展的四个阶段

- 各自为政：上世纪80年代到1993年期间，面向对象方法出现了百家争鸣的局面，但是不同方法的模型相互转换几乎不可能；
- 统一阶段：1994年10月开始，Rational 公司的Booch、Rumbaugh和Jacobson 在Booch、OMT和OOSE方法的基础上进行研究，于1996年发布了统一建模语言UML（Unified Modeling Language）；
- 标准化阶段：OMG为了使UML标准更加完善，发布了征求建议书（RFP），随后，Rational软件有限公司建立了UML Partners联盟，各软件开发商和系统集成商共同努力，1997年制定出UML1.1标准，被OMG采纳；
- 工业界应用：1998年OMG接管了UML标准的维护工作，UML已成为软件工业界事实上的标准。

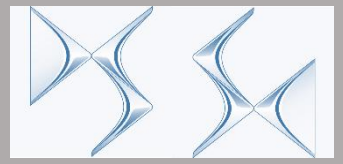


UML的主要特点

- UML 是一种标准的图形化建模语言，它是面向对象分析与设计的一种标准表示，它
  - 不是一种可视化的程序设计语言，而是一种可视化的建模语言；
  - 不是工具或知识库的规格说明，而是一种建模语言规格说明，是一种表示的标准；
  - 不是过程，也不是方法，但允许任何一种过程和方法使用它。

UML 的基本结构

- 基本构造块 Basic building block
  - 事物 Thing
  - 关系 Relationship
  - 图 Diagram
- 语义规则 Rule
  - name、scope、visibility、integrity、execution
- 通用机制 Common mechanism
  - specification、adornment、common division、extensibility mechanism



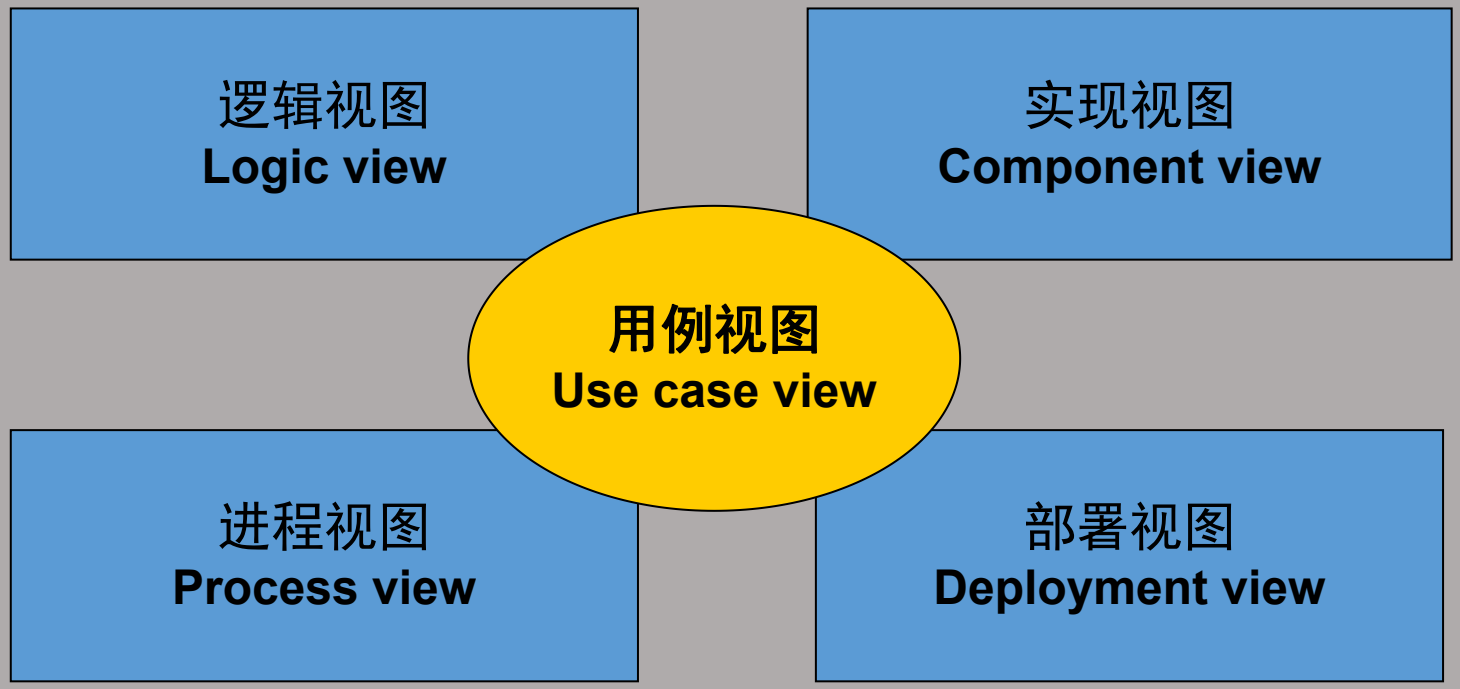
UML 的基本结构

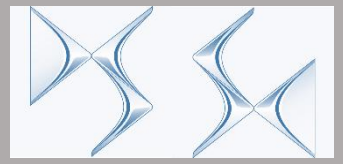
- 事物 Thing 及关系 Relationship
  - Structural thing
    - Class, interface, collaboration, use case, component, node
  - Behavior thing
    - Interaction, state machine
  - Group thing
    - package
  - Annotation thing
    - note

- 依赖 Dependency
- 关联 Association
- 泛化 Generalization
- 实现 Realization

UML的4+1视图

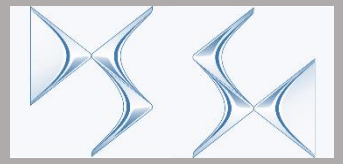
- UML 用模型来描述系统的结构（静态特征）以及行为（动态特征）。从不同的视角为系统的架构建模，形成系统的不同视图（view）， 称为4+1视图





UML 4+1视图的作用

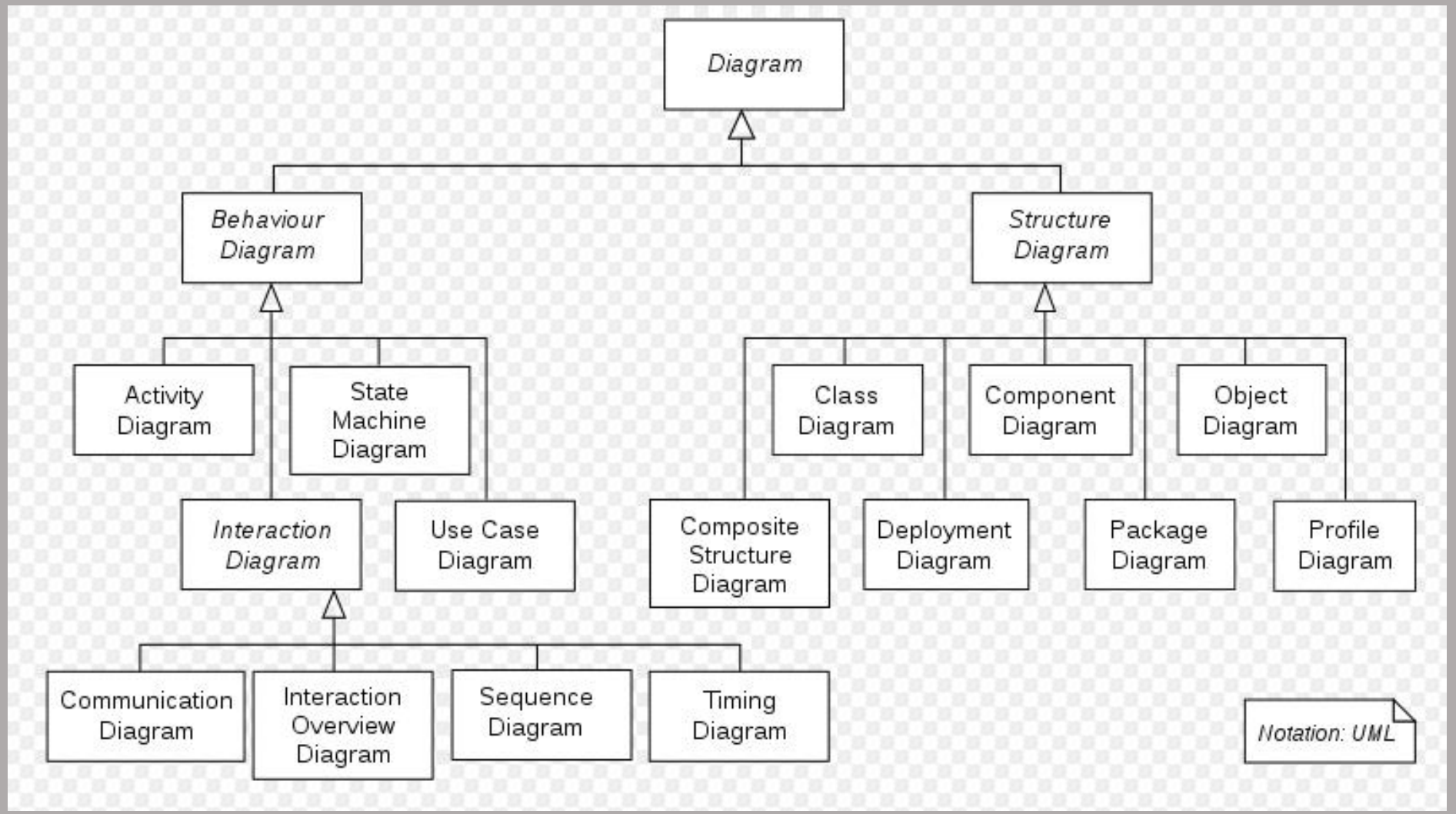
- 用例视图：强调从用户的角度看到的或需要的系统功能，这种视图也叫做用户模型视图（user model view） 或场景视图（scenario view）；
- 逻辑视图： 展现系统的静态或结构组成及特征，也称为结构模型视图（structural model view） 或静态视图（static view）；
- 进程视图：描述设计的并发和同步等特性，关注系统非功能性需求，也称为行为模型视图（behavioral model view）、过程视图（process view）、 协作视图（collaborative view）和动态视图（dynamic view）；
- 构件视图：关注软件代码的静态组织与管理，也称为实现模型视图（implementation model view ）和开发视图（development view）；
- 部署视图：描述硬件的拓扑结构以及软件和硬件的映射问题，关注系统非功能性需求（性能、可靠性等），也称为环境模型视图或物理视图（physical view）；

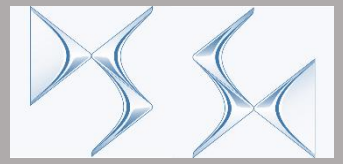


UML的9个基本图

- 用例图（Use case diagram）：（从用户的角度）描述系统的功能；
- 类图（Class diagram）：描述系统的静态结构（类及其相互关系）；
- 对象图（Object diagram）：描述系统在某个时刻的静态结构（对象及其相互关系）；
- 顺序图（Sequence diagram）：按时间顺序描述系统元素间的交互；
- 协作图（Collaboration diagram）：按照时间和空间的顺序描述系统元素间的交互和它们之间的关系；
- 状态图（State diagram）：描述了系统元素（对象）的状态条件和响应；
- 活动图（Activity diagram）：描述了系统元素之间的活动；
- 构件图（Component diagram）：描述了实现系统的元素（类或包）组织；
- 部署图（Deployment diagram）：描述了环境元素的配置并把实现系统的元素映射到配置上。

UML 图的关系



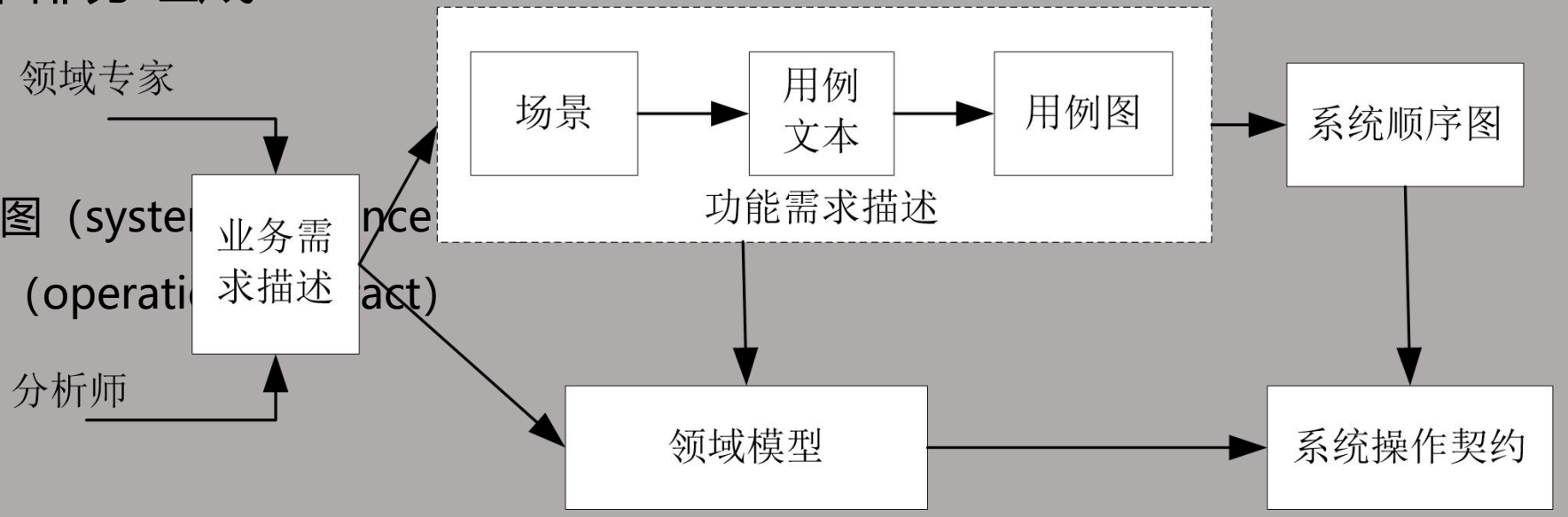


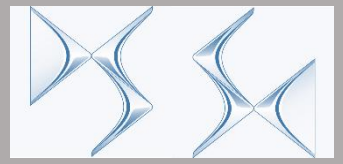
UML视图与图的关系

- 用例视图：使用用例图；
- 逻辑视图：使用类图、对象图，顺序图/协作图；
- 进程视图：使用状态图和活动图；
- 构件视图：使用构件图；
- 部署视图：使用部署图。

面向对象的需求分析建模

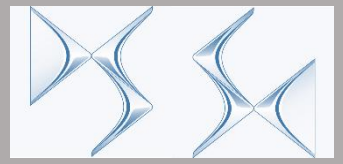
- 面向对象分析方法中的需求分析包含两个模型：领域模型和用例模型。
  - 领域模型表示了需求分析阶段“当前系统”逻辑模型的静态结构及业务流程；
  - 用例模型是“目标系统”的逻辑模型，定义了“目标系统”做什么的需求。由以下四个部分组成：
    - 用例图
    - 用例说明
    - 系统顺序图（syste nce
    - 操作契约（operati act）





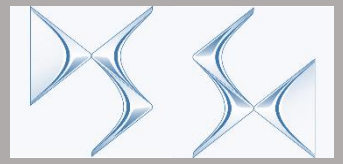
领域模型

- 领域模型：针对某一特定领域内概念类或者对象的抽象可视化表示。
- 主要用于概括地描述业务背景及重要的业务流程，并通过UML的类图和活动图进行展示，帮助软件开发人员在短时间内了解业务。
  - 业务背景：可由用户需求说明书或者用例说明中具有代表业务概念或者业务对象的词汇获得，这些词汇可统称为“概念类”；并通过能够代表关系的词汇建立概念类之间的关系，表示成能够代表业务知识结构的类图；
  - 业务流程：一般由角色及其执行的活动（活动及任务节点）构成，活动的输出一般有数据对象和传给另一个活动的消息组成，建议使用UML的活动图进行描述。



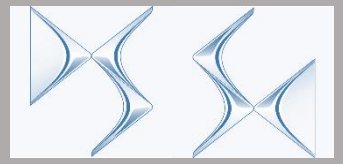
领域模型与软件模型的区别

- 领域模型所关注的仅仅是客观世界中的事物并将其可视化，而非诸如Java或C#类的软件对象。
- 以下元素不适用于领域模型
  - 软件制品，例如窗口、界面、数据库
  - 软件模型中具有职责或方法的对象
- OO的关键思想：逻辑层（Logic Layer）中软件类的名称要源于领域模型的概念类和职责，减小人们的思维与软件模型之间的表示差异。
  - 逻辑层的命名也取自于实际的业务逻辑，表明这些软件类替换了某些业务职责



识别概念类

- 通过对用例描述中的名词或名词短语寻找和识别概念类；
- 需要注意的是名词可以是概念类，也可能是概念类中表示特征的属性；
  - 属性一般是可以赋值的，比如数字或者文本。
  - 如果该名词不能被赋值，那么就“有可能”是一个概念类。
  - 如果对一个名词是概念类还是属性举棋不定的时候，最好将其作为概念类处理。
  - 需要注意的是：不存在名词到类的映射机制，因为自然语言具有二义性
- 这种方法的弱点是自然语言的不精确性，建议初学者使用

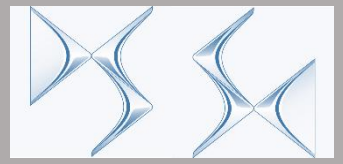


创建领域模型的步骤

- 理解领域模型对理解系统需求至关重要，领域模型的创建步骤如下：
  - 第1步，找出当前需求中的候选概念类；
  - 第2步，在领域模型中描述这些概念类。用问题域中的词汇对概念类进行命名，将与当前需求无关的概念类排除在外。
  - 第3步，在概念类之间添加必要的关联来记录那些需要保存记忆的关系，概念之间的关系用关联、继承、组合/聚合来表示。
  - 第4步，在概念类中添加用来实现需求的必要属性。

识别名词短语

1.	学生在系统主窗口中选择参加考试。
2.	系统列出该学生该时段能参加的所有考试课程名称。
3.	学生选择其中的一门课程，要求考试。
4.	系统弹出登陆框，要求学生输入考试密码。
5	学生输入从监考老师处获取的考试密码，登录。
6.	系统显示欢迎界面，展示考试课程名称和考试时长，询问学生是否开始考试。
7.	学生选择开始考试。
8.	系统按照预先设计好的考卷生成规则自动生成一套考卷。
9.	系统显示考题。
10.	学生答题，并提交该题答案。
11.	系统记录答案，并使上一题或下一题按钮生效。
12.	学生选择上一题或者下一题。 系统重复步骤9～12，直到学生选择结束考试或者考试时间到。
13.	系统显示学生的选择题得分。
14.	系统询问学生是否退出考试。
15.	学生选择退出。
16.	系统回到系统主窗口。



在线考试系统中的概念类

学生

老师

课程

课程规格说明

考试

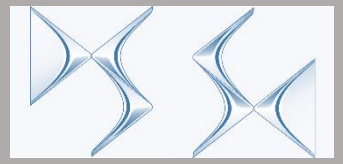
考卷生成规则

考卷生成规则项

考卷

考题

考题规格说明



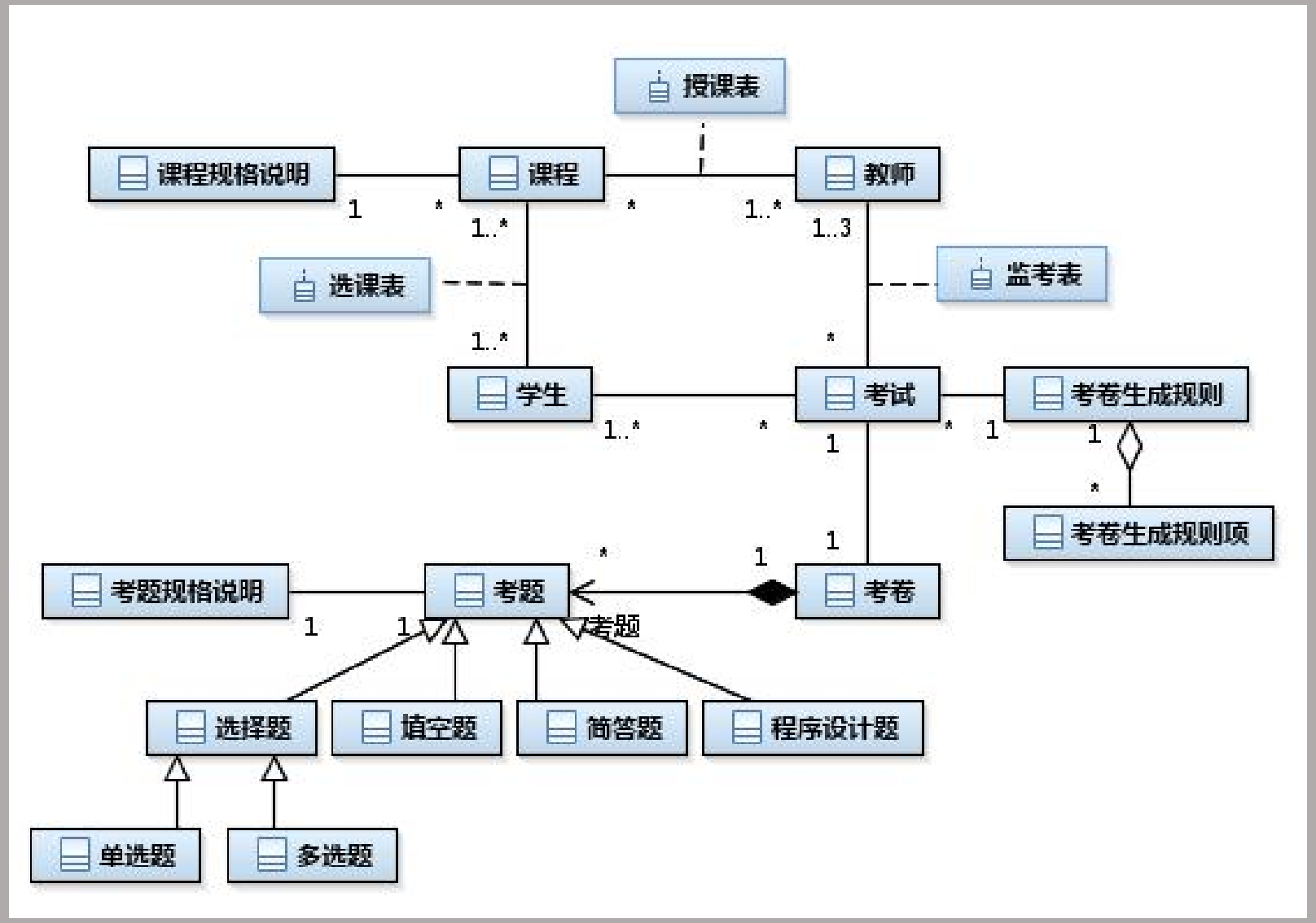
添加关联

- 领域模型中的关联可分为两种：
  - “需要知道”型关联：需要将概念之间的关系信息保持一段时间的关联。领域模型中需要着重考虑。
  - “只需理解”型关联：有助于增强对领域中关键概念的理解的关联。
- 寻找关联时要遵循下述指导原则：
  - 将注意力集中在需要知道型关联。
  - 识别概念类比识别关联更重要，因此领域模型创建过程中应该更加注重概念类的识别。
  - 太多的关联不仅不能有效地表示领域模型，反而容易使领域模型变得混乱。
  - 避免显示冗余或导出关联。

考试系统中“需要知道型”关联

关联	含义
学生“参加”考试	为了知道学生是否需要参加该项考试
老师“监考”考试	为了知道由哪（几）位老师监考
考卷“记录着”考题	为了知道一份考卷由哪些考题组成
考卷生成规则“对应于”课程	为了知道是哪门课程的考卷生成规则
考卷生成规则“应用于”考试	为了知道某次考试使用哪套考卷生成规则
考卷生成规则“记录着”考卷生成规则项	为了知道一套考卷生成规则由哪些细项组成
考题规格说明“详细描述”考题	为了知道一道考题的详细描述

在线考试系统部分领域模型

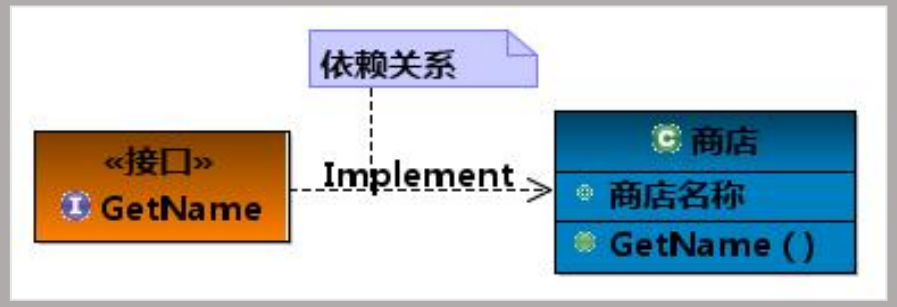


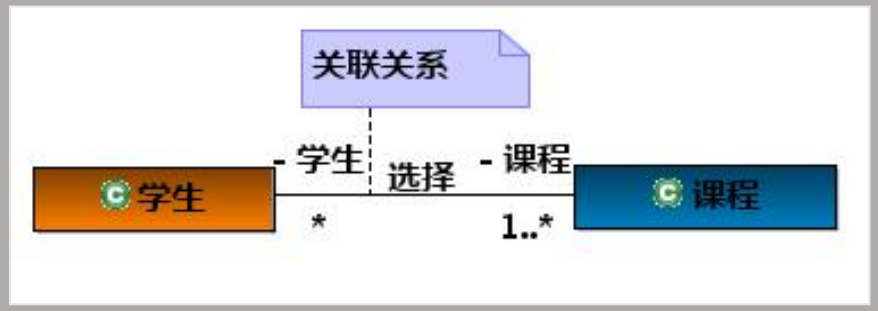
UML 类图的组成

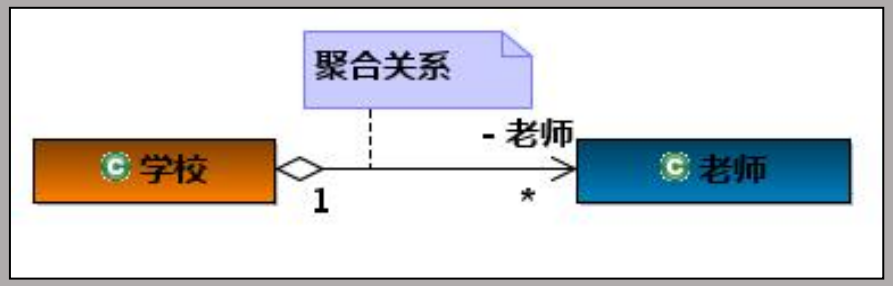
- UML类图用于描述类以及类之间的关系。
- 类包含三个部分：
  - 类名：表示问题域中的概念，含义清晰准确
  - 属性：**可见性 属性名：类型名= 初始值 {性质串}**
  - 操作：**可见性 操作名（参数表）：返回值类型 {性质串}**
- 类的关系有：
  - 关联：普通关联、导航关联、递归关联
  - 组合与聚合
  - 依赖和继承

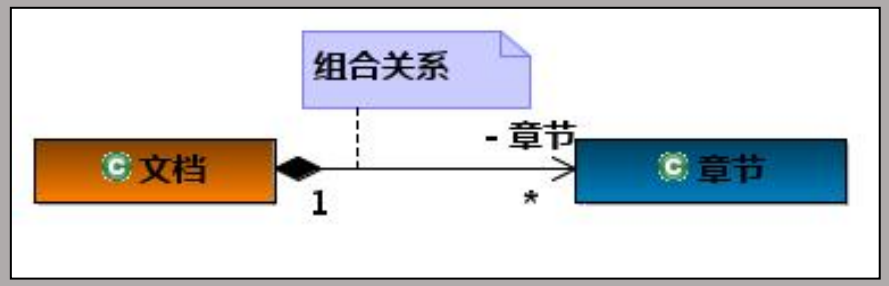
类的关系

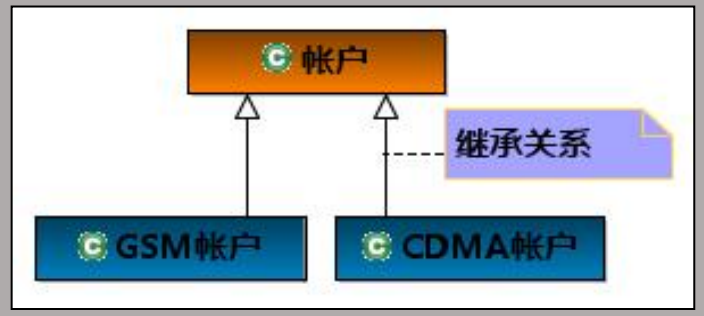
- 以下按照由松散到紧密地的关系进行说明







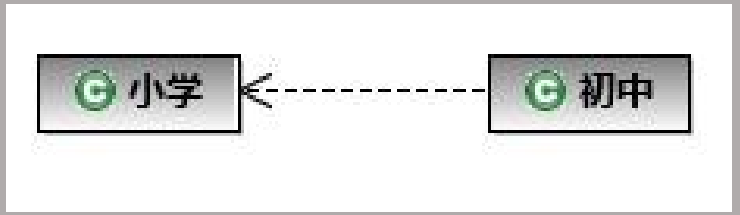


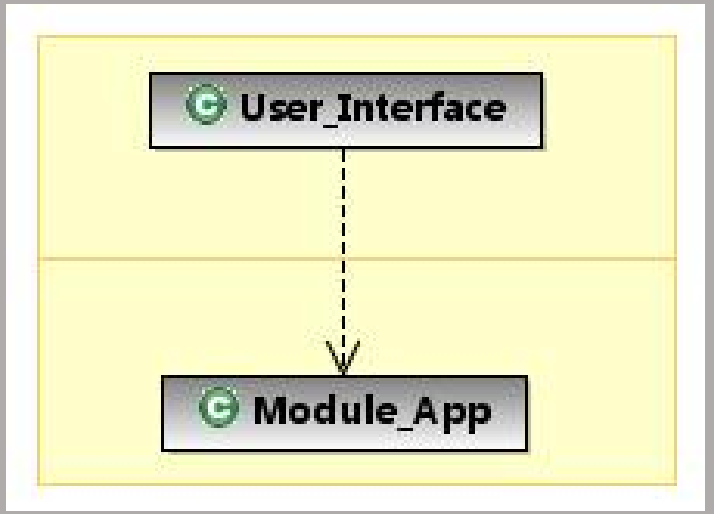


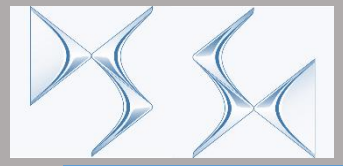
类的依赖关系

- 类A 把 类B 的实例作为方法里的参数使用；
- 类A 的某个方法里使用了类B 的实例作为局部变量；
- 类A 调用了 类B的静态方法

- 那么 类A依赖于类B





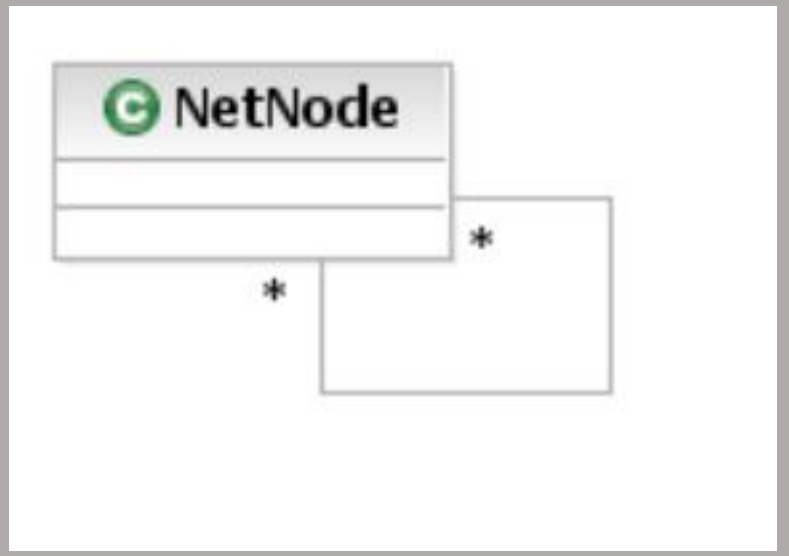


依赖关系与Java

```
public class Man
{
        public void drive(Car car)
        {
                car.start();
        }

        public void sleep()
        {
                Light light = new Light();
                light.off();
        }

        public int getMoney(int amountOfNeed)
        {
                return ATM.fetch(amount OfNeed);
        }
}
```

```
public class Car
{
    public void start()
    {
        system.out.println("car is start");
    }
    public void stop()
    {
    .......
    }
}

public class Light
{
    public void off()
    {
        .......
     }
}

public class ATM
{
    public static int fetch(int amountOfMoney)
    {
        return amountOfMoney;
    }
}
```

类的关联关系

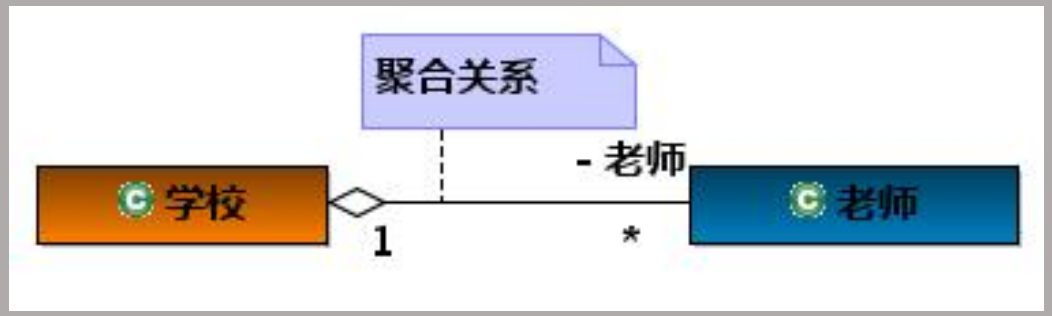


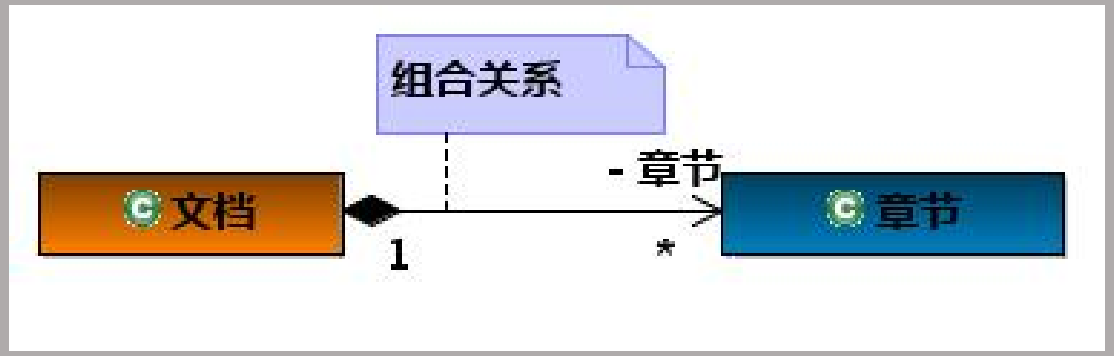
关联关系与Java


```
Public class student {
  private Book [ ] book;
  … // other attributes &
methods
}

Public class Book {
  private Student student;
  … // other attributes &
methods
}
```

```
Public class student {
  private Book [ ] book;
  … // other attributes &
methods
}

Public class Book {
  //  private Student student;
  … // other attributes &
methods
}
```

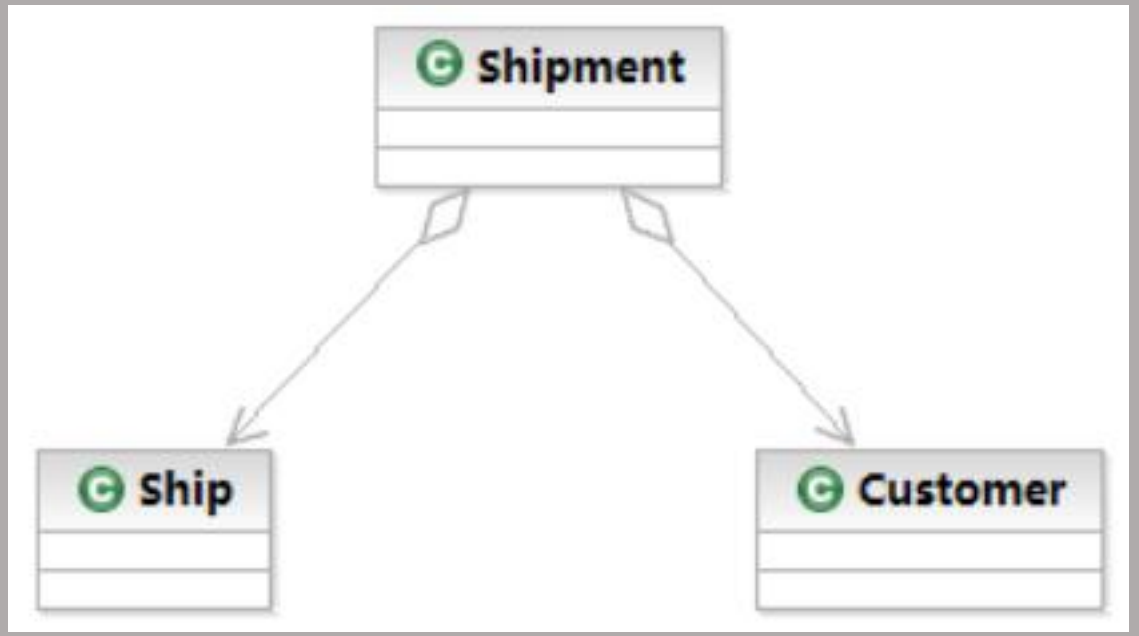
类的聚合与组合





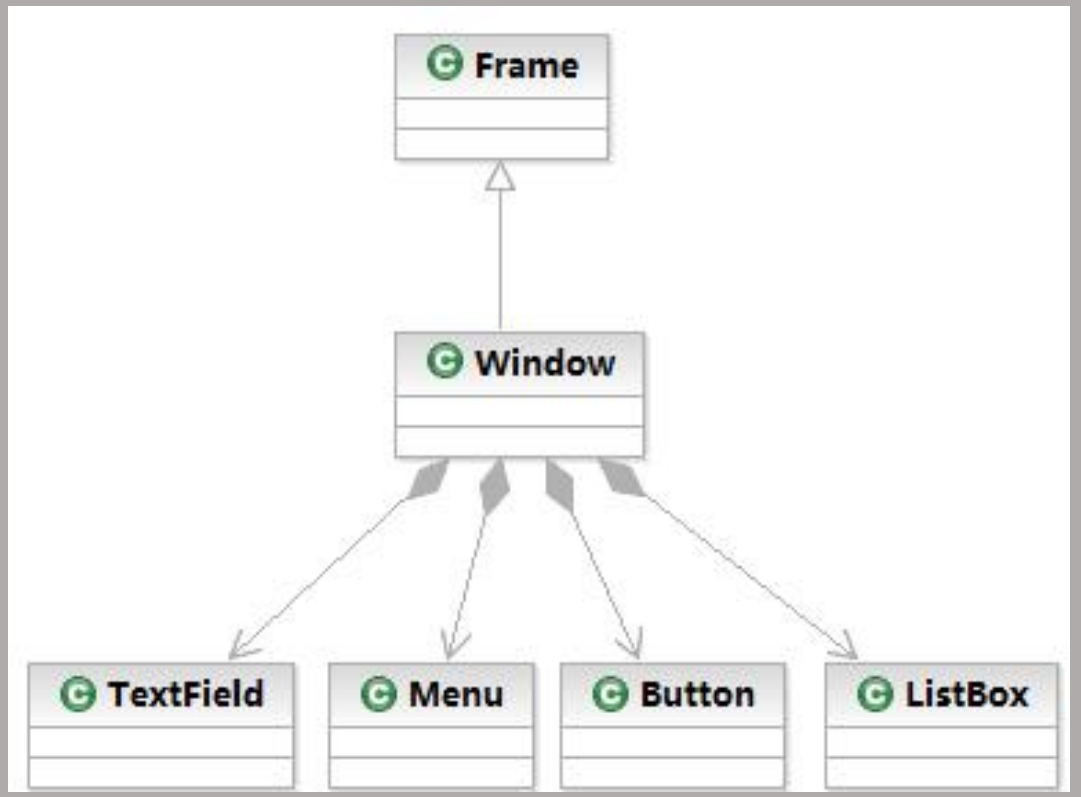
聚合关系与Java

```
public class Shipment
{
        private Ship ship;
        private Customer[] customer;
}

public class Ship
{
        private String id;
        private String owner;
}

public class Customer
{
        private String Name;
        private String creditCardNumber;
}
```

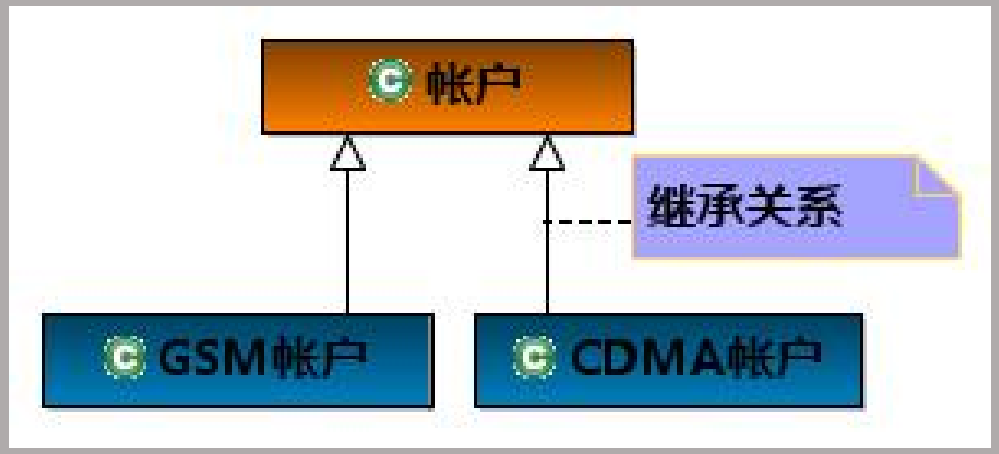


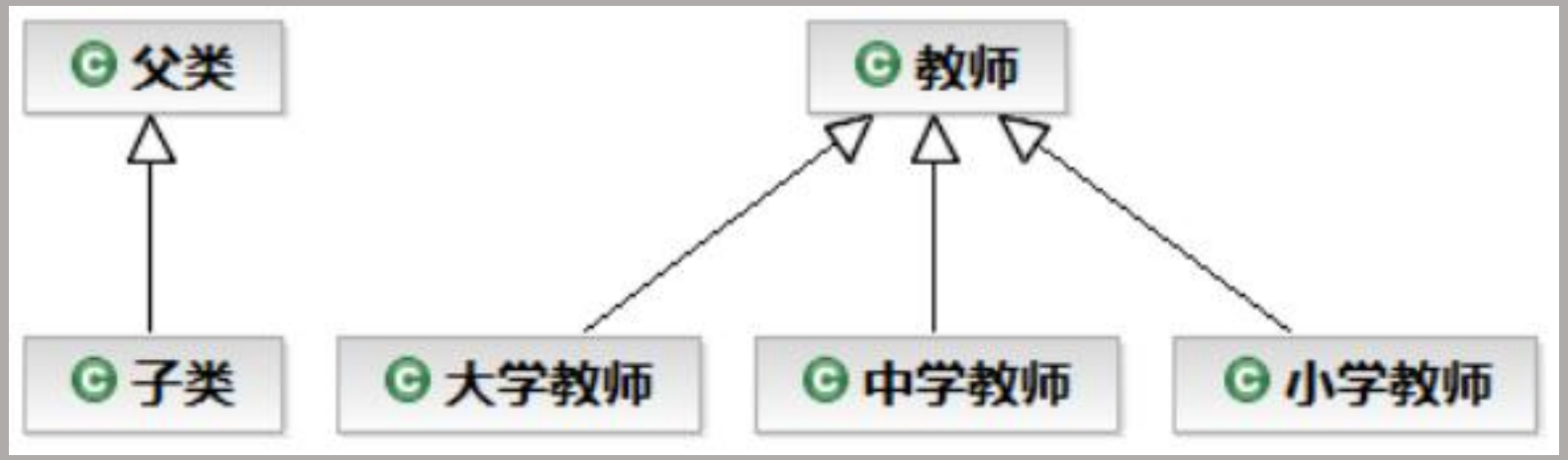
组合关系与Java

```
public class Window extends Frame
{
        TextField txt = new TextField();
        Button btn = new Button();
        MenuBar mbr = new MenuBar();
        MenuItem item = new MenuItem();
        Menu m = new Menu();
.....
}
```



类的继承



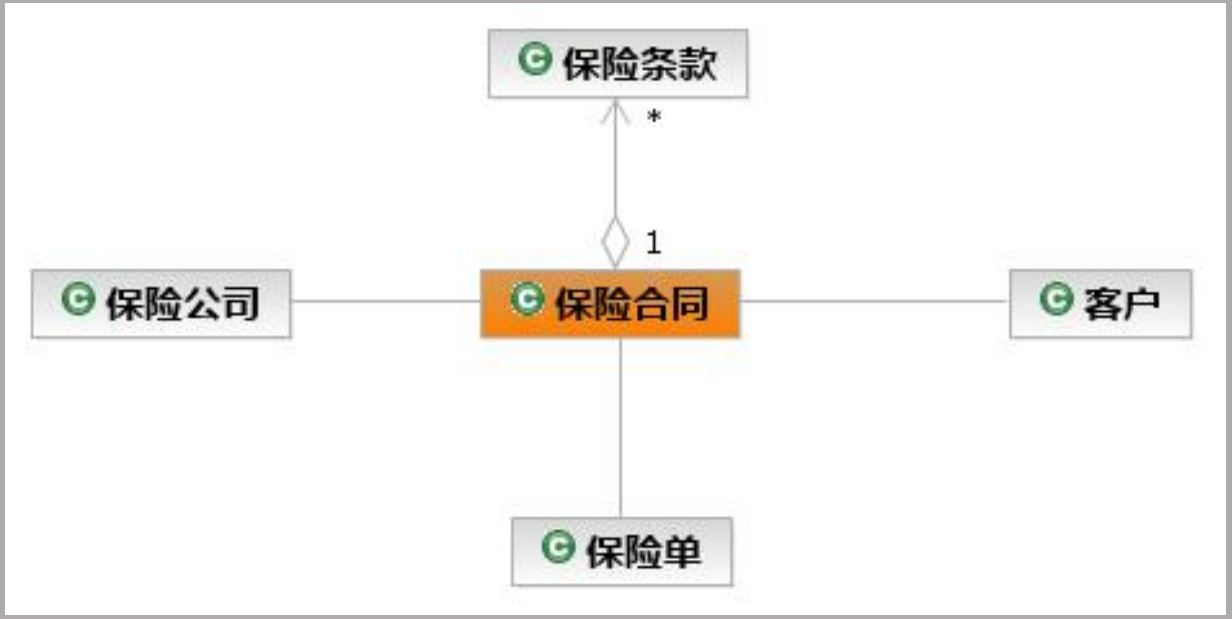


关联类

- 在关联建模中，存在一些情况下，需要包括其它类，因为它包含了关于关联的有价值的信息。
- 对于这种情况，使用关联类来绑定这些基本关联。关联类和一般类一样表示。不同的是，主类和关联类之间用一条相交的点线连接。

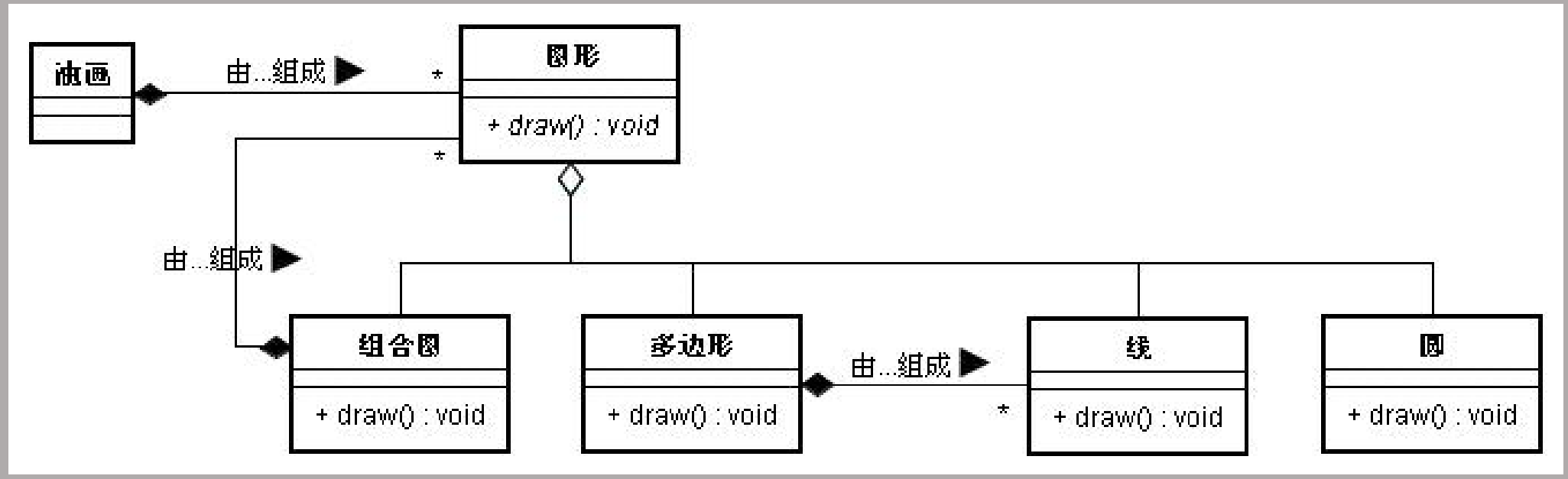
保险业务 案例

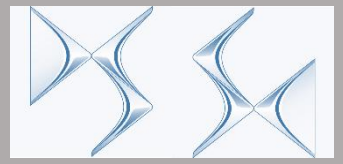
- 保险公司使用保险合同来代表保险业务，这些合同与客户有关。客户可没有或者具有多个保险合同，这些合同至少与一个保险公司有关。
  - 保险合同位于一家保险公司和一个或多个客户之间，它建立保险公司和客户之间的保险关系
  - 保险合同使用保险单表示，也就是合同的书面表示,一个保险单表示一份保险合同。



例子：油画

- 一幅油画由许多图形组成，图形可以由直线、圆、多边形和各种线型混合而成的组合图等。

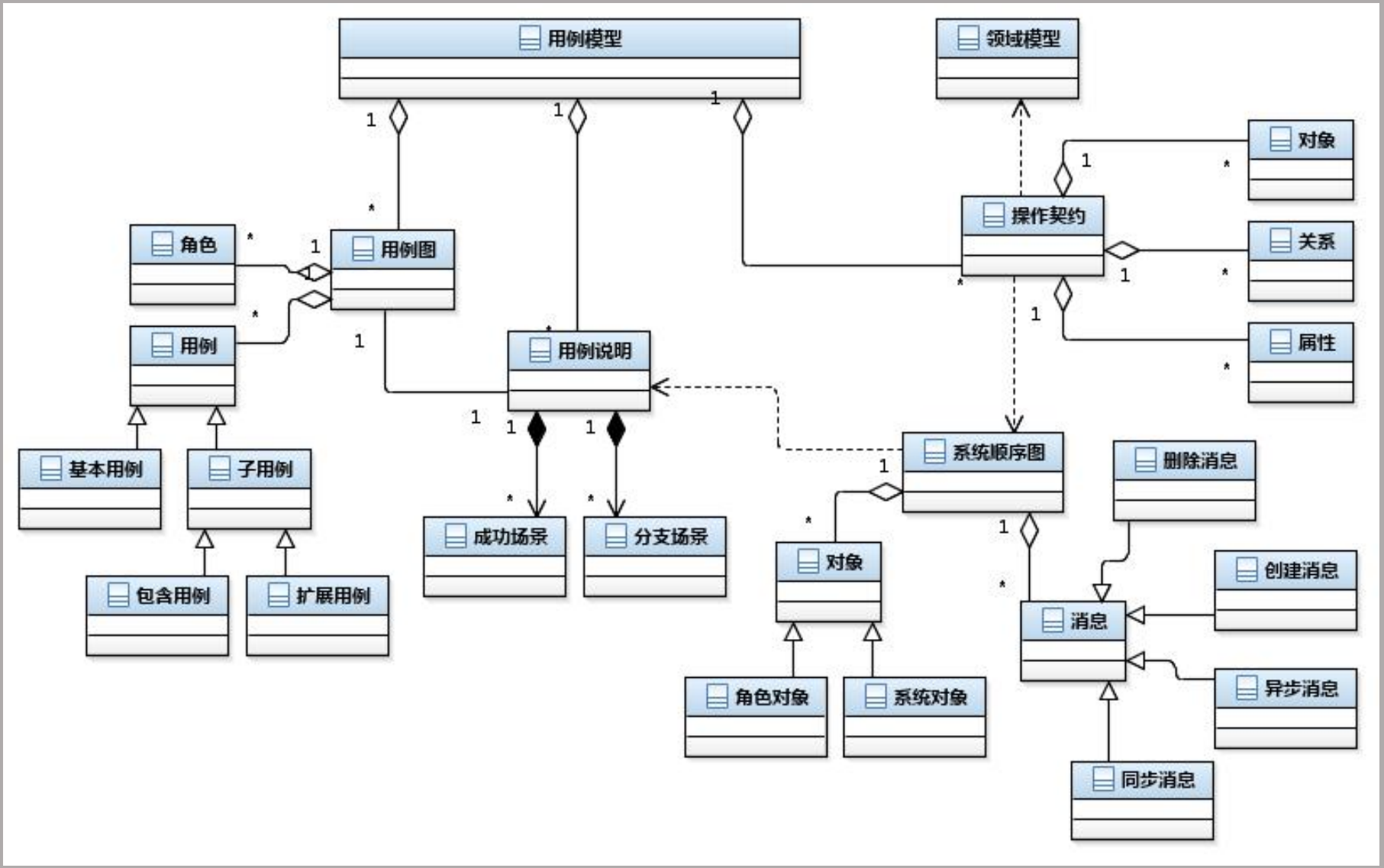




用例模型

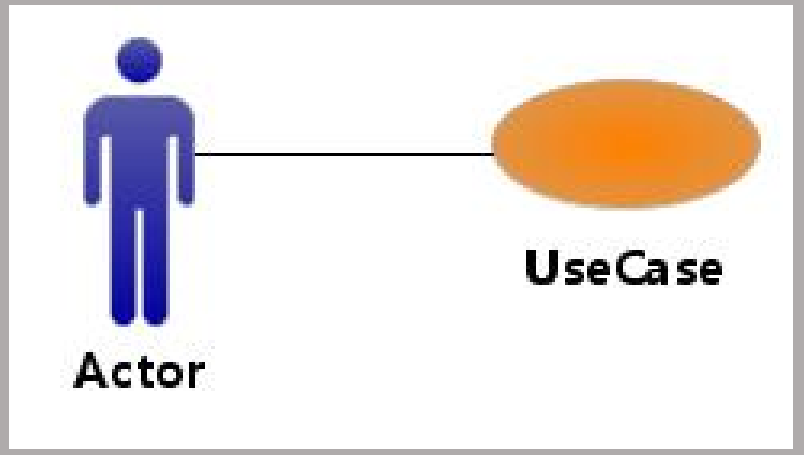
- 用例模型由以下四个部分组成：
  - 用例图；
  - 用例说明；
  - 系统顺序图（system sequence diagram，option）；
  - 操作契约（operation contract，option）；
- 以用例为核心从使用者的角度描述和解释待构建系统的功能需求

用例模型的基本结构



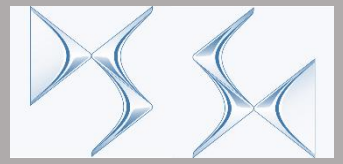
用例图

- 用例图由三个基本元素组成
  - Actor：称为角色或者参与者，表示使用系统的对象，代表角色的不一定是人，也可以是组织、系统或设备；
  - Use_case：称为用例，描述角色如何使用系统功能实现需求目标的一组成功场景和一系列失败场景的集合；
  - Association：表示角色与用例之间的关系，以及用例和子用例之间的关系；



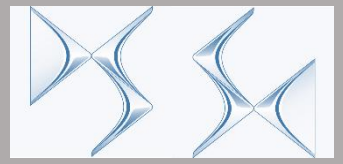
案例-1：银行柜台取款场景

- 银行柜台取款的场景：
  1. 某储户到银行柜台取钱，人多排队；
  2. 等到柜台窗口时，将存折或银行卡交给窗口的营业员；
  3. 营业员通过刷卡器读入账号信息，并提示储户输入密码；
  4. 密码验证通过后营业员询问储户需提取多少钱？
  5. 储户告知具体的提款金额；
  6. 营业员通过系统输入取款金额，并打印本次取款的操作记录，交给储户签字认可；
  7. 营业员核对无误后将取款的现金和银行卡交给储户；
  8. 储户接收现金和银行卡，起身离开；
  9. 完成一次银行柜台取钱的交易。
- 由于柜台人数限制，该银行希望开发ATM系统，并要求具有余额查询、取款、更改登录密码的功能



案例-1：ATM系统取款场景分析

- 根据银行柜台的取款场景，可以设计如下使用ATM取款的场景：
  1. 储户将银行卡插入ATM；
  2. 储户根据提示输入密码；
  3. 储户根据系统的操作提示选择取款；
  4. 储户根据提示确定取款金额；
  5. 储户从取款箱中取出现金；
  6. 储户选择是否打印操作凭据；
  7. 储户根据系统提示选择退出；
  8. ATM退出银行卡；
  9. 储户取卡离开；
  10. ATM完成一次取款交易。

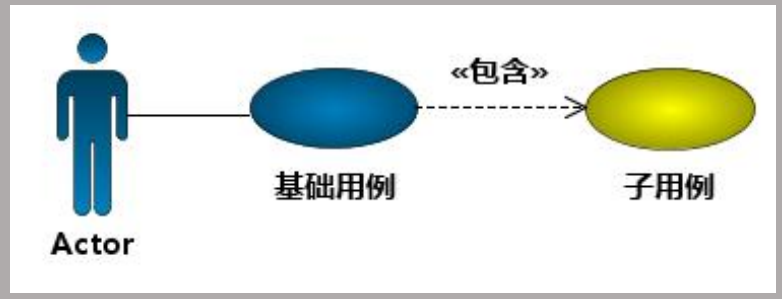


用例获取

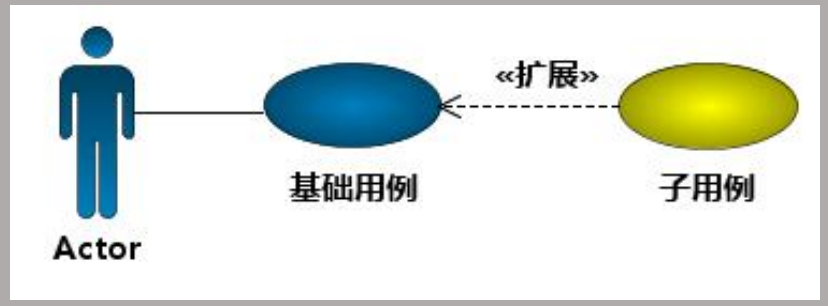
- 根据ATM取款的场景描述，有以下可以表示功能的动词：
  - 取款，插入，输入，选择，确定，取出，退出、取卡和完成；
  - 插入银行卡：表示验证银行卡的有效性；
  - 输入密码：表示进行身份验证；
  - 选择操作：表示储户本次操作的目的性，取款；
  - 确定金额：表示目的性的具体数值；
  - 取出现金：表示本次操作是否正确；
  - 退出银行卡：表示系统已经确认本次操作的结束；
  - 储户取卡：表示本次交易结束；
- 代表该角色 目的性 功能的用例：取款
- 请大家练习描述查询余额以及更改密码的场景！

基本用例与子用例

- 基本用例：与角色直接相关的用例，表示系统的功能需求；
- 子用例：通过场景描述分析归纳出的用例，也表示了系统的功能，但这些用例与角色无直接关系，而与基本用例存在关联关系；
  - 包含子用例：多个基本用例中的某个与角色交互的场景具有相同的操作，且这些场景都是基本用例中必须执行的步骤，可以将其抽取出来作为基本用例的子用例；

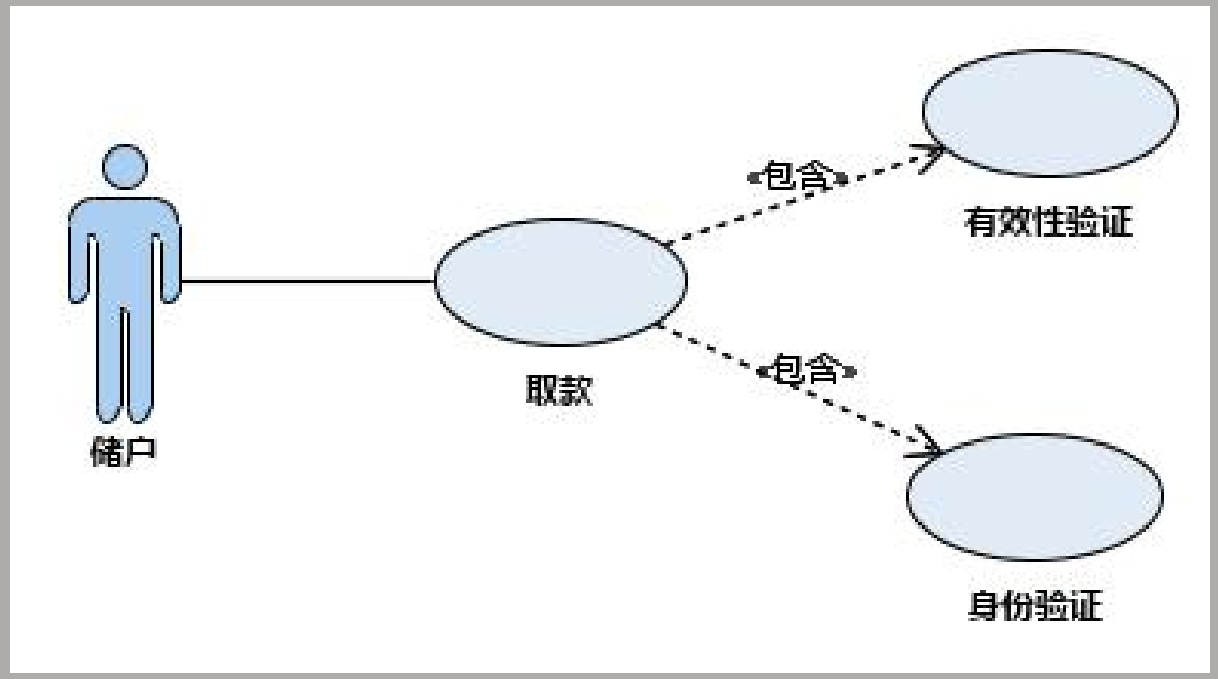


  - 扩展子用例：（多个）基本用例中的某些场景存在相同的条件判断的情况，可以将其抽取出来作为基本用例的子用例；



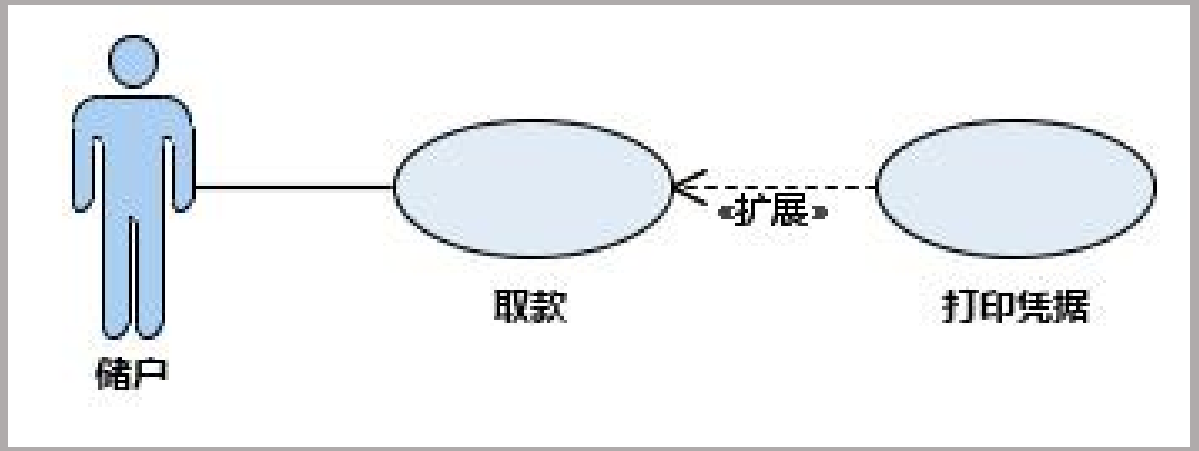
案例-1：包含子用例

- 在ATM的取款、查询余额、更改密码的操作场景中，经分析都存在一段验证银行卡有效性及身份验证的场景，而且都是基本用例必须执行的操作，为此可以将其抽取出来作为基本用例的包含子用例。



案例-1：扩展子用例

- 在取款和查询余额的操作场景中，存在一段是否打印操作凭据的情景，经分析该功能可以成为一个子用例，且符合在某种条件下可以执行的用例，为此该子用例应为基本用例的扩展子用例。



- 请同学们考虑更改密码与取款用例的关系

用例说明模板

- 基于已经找到的用例和子用例，并参考之前的需求定义以及场景描述的内容，将用例交互的成功场景和失败场景以标准的格式归纳描述。

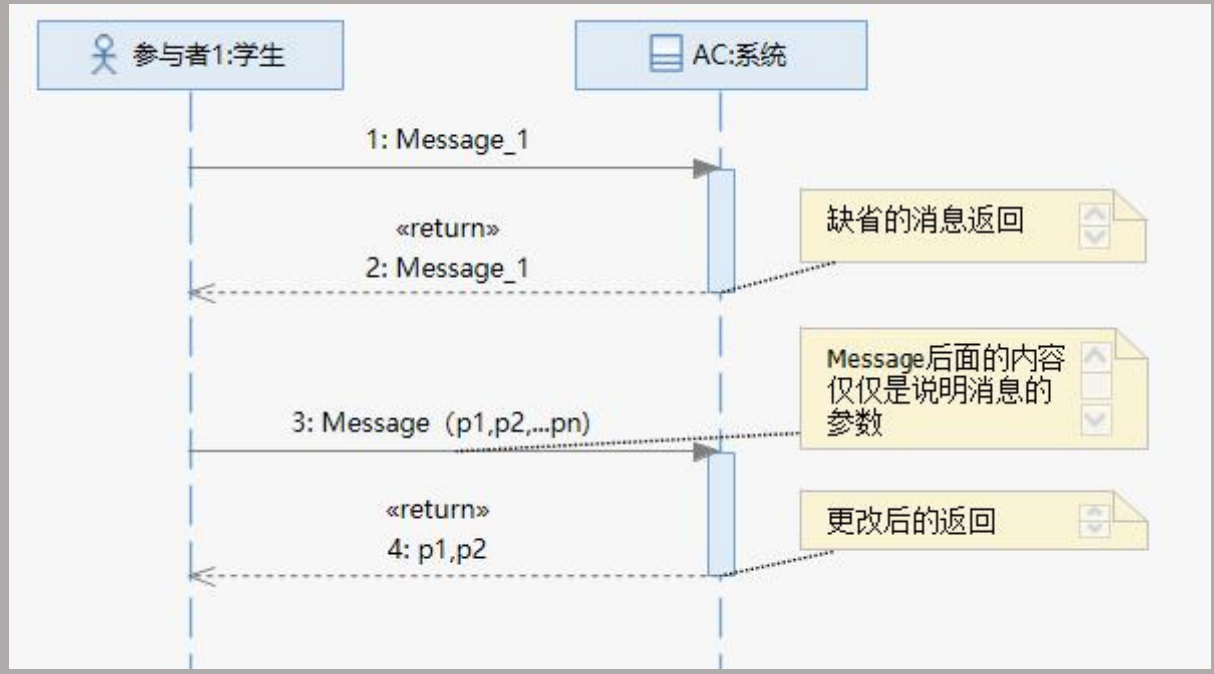
用例编号:	**每一个用例一个唯一的编号，方便在文档中索引。**
用例名称:	**（状语+）动词+（定语+）宾语，体现参与者的目标。**
范围:	**应用的软件系统范围**
级别:	**用例/子用例**
参与者:	**参与者的名称**
项目相关人员及其兴趣:	**用户应包含满足所有相关人员兴趣的内容**
前置条件:	**规定了在用例中的一个场景开始之前必须为“真”的条件。**
后置条件:	**规定了用例成功结束后必须为“真”的条件。**

用例说明模板

主要成功场景:描述能够满足项目相关人员兴趣的一个典型的成功路径。不包括条件和分支	
1.	
.....	
.....	包含子用例的名称 或者 扩展子用例的名称
n.	
扩展（或替代流程）：（备选路径）说明了基本路径以外的所有其他场景或分支	
*a.	描述任何一个步骤都有可能发生的条件，前边加*
5a.	对基本路径中某个步骤的扩展描述，前边加基本路径编号
特殊需求:	与用例相关的非功能性需求
技术与数据的变化列表:	输入输出方式上的变化以及数据格式的变化。
发生频率:	用例执行的频率。
待解决的问题:	不清楚的、尚待解决的问题可集中在此进行罗列

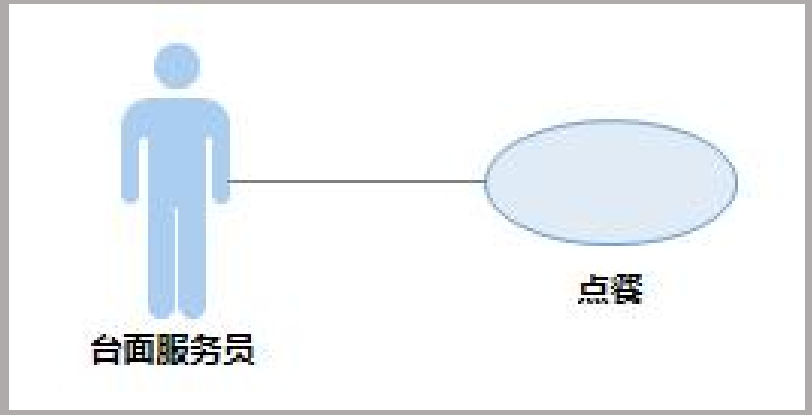
系统顺序图

- 使用UML的sequence diagram描述角色与系统之间的交互
- 在用例描述的基础上需进一步确定角色与系统之间的交互信息，并以可编程的方式将其命名；
- 系统顺序图中“一般”只需要三个UML的符号元素
  - 角色；
  - 代表软件系统的对象，一般使用system或者系统命名；
  - 角色与system之间的交互信息，简称消息或操作；

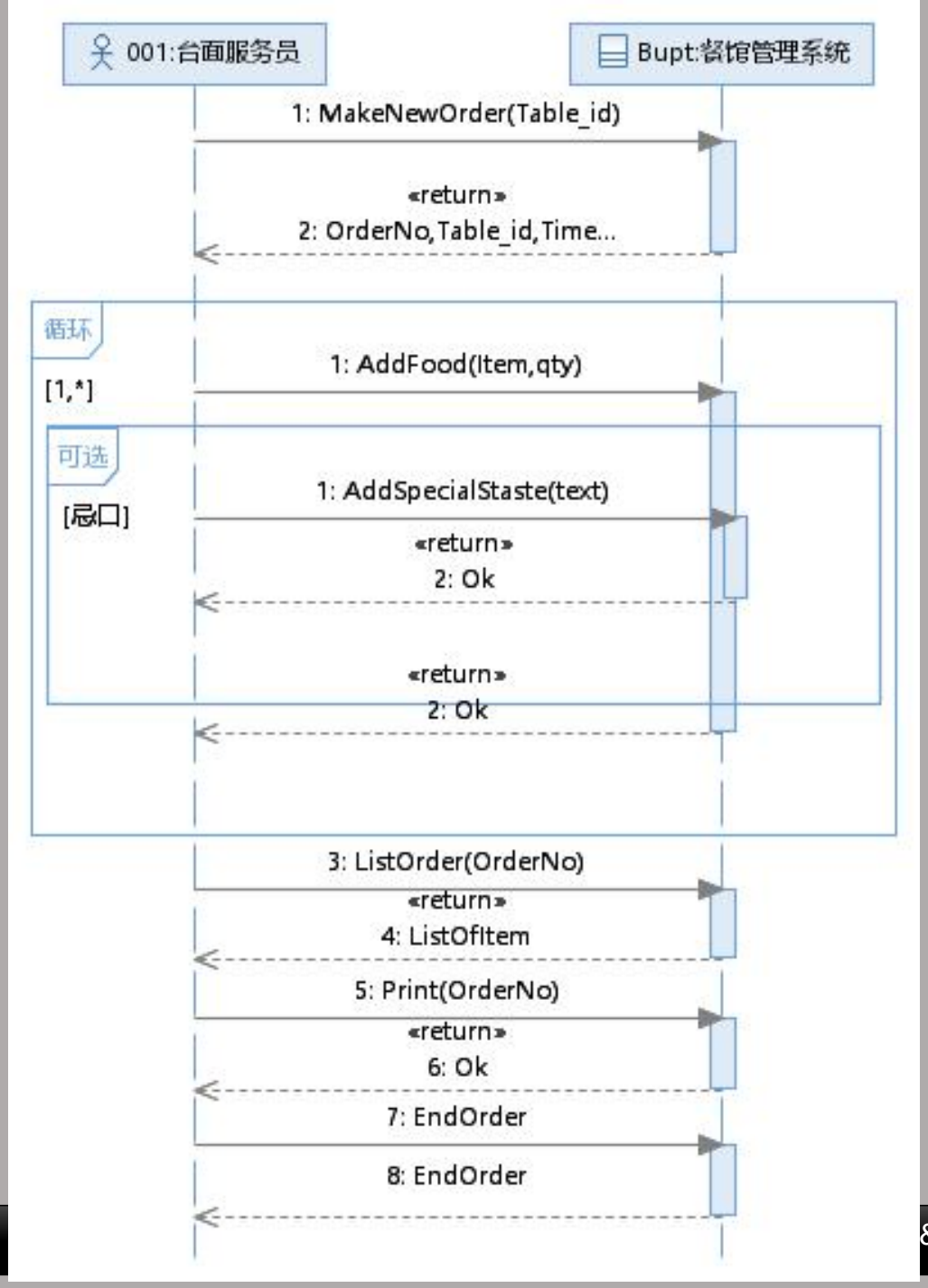


案例-2：餐馆点餐场景

- 角色：台面服务员，负责台面客人的点菜，生成订单；
  1. 客人进入餐馆，安排就坐；
  2. 服务员将菜单递给客人阅览；
  3. 客人呼叫台面服务员点菜；
  4. 服务员手持点菜POS机（或者点菜小本）开始记录菜品信息；
  5. 同时记录菜品的忌口信息；
  6. 直到客人表示完成点菜，生成一张订单并打印；
  7. 完成一次点菜。
- 考虑该场景中的领域模型中的概念类



系统顺序图（SSD）



注意：

1.SSD是用于替代用例说明文本的一种方式；

2.图中只有两个对象，表示角色对象与系统对象；

3.图中的消息名称及参数要求以可编程的方式命名；

4.消息名称和参数可以通过一个列表使用中文说明具体含义；

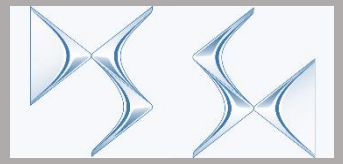
5.用例图中的每个用例都应该对应一张SSD；

6.角色发给系统的指令（系统事件）是操作契约关注的元素

操作契约

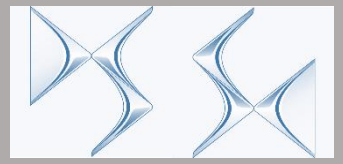
- 系统操作：处理系统事件的操作，也称为系统事件；
- 操作契约是为系统操作（指令）而定义的，参考领域模型中业务对象（**概念类**）接收到相同的系统事件后，执行必须的业务处理时各业务对象的状态以及系统操作执行的结果，以便软件设计时进行参考。模板如下表所示：

操作：	操作以及参数的名称
交叉引用：	（可选择）可能发生此操作的用例
前置条件：	执行该操作之前系统或领域模型对象的状态
后置条件：	操作完成后领域模型中对象的状态： **1、对象的创建和删除；** **2、对象之间“关联”的建立或消除；** **3、对象属性值的修改；**



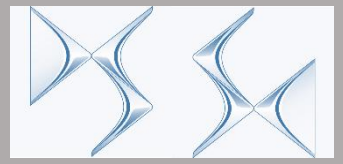
创建操作契约

- 创建操作契约的指导原则如下：
  - 根据系统顺序图识别进入到系统内的所有系统事件，即操作；
  - 针对每一个系统操作结合对应的领域模型，找到与此操作相关的概念类；
  - 定义概念类响应该操作的以下三项内容；
    - 对象实例创建和删除；
    - 对象关联形成和断开；
    - 对象属性修改。
- 后置条件中至少有一项存在，该操作才有存在的必要性！
- 后置条件的陈述应该是声明性的，以强调系统状态所发生的变化，无需考虑如何设计和实现的。



点菜用例的系统事件

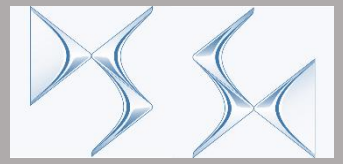
- MakeNewOrder（Table_id）：开始一个点菜服务请求；
- AddFood（Item_id, quantity）：添加菜品；
- AddSpecialTaste（text）：添加特殊忌口信息；
- ListOrder（OrderNo）：列出所选菜品清单；
- Print（OrderNo）：打印所选菜品订单；
- EndOrder（）：结束一次点菜服务；



MakeNewOrder（Table_id)

- 参考领域模型分析并说明：开始一次点菜服务，说明要创建一个新的订单。
  - 根据领域模型，该订单必须与台面相关联；
  - 还必须与管理该台面的服务员相关联；
  - 除此之外，该订单流水号、订单创建时间必须被动态创建；
  - 为了能够记录多项菜品信息，还必须初始化一个“集合”项来添加多个菜品；

系统事件	MakeNewOrder（TableId）
交叉引用	订单处理
前置条件	服务员身份验证通过，开始订单处理
后置条件	1. 一个新的（概念类）订单被创建； 2. 订单与（概念类）台面建立关联； 3. 订单与（概念类）台面服务员建立关联； 4. 订单的属性初始化：订单流水号、订单时间、存储菜品的数组等

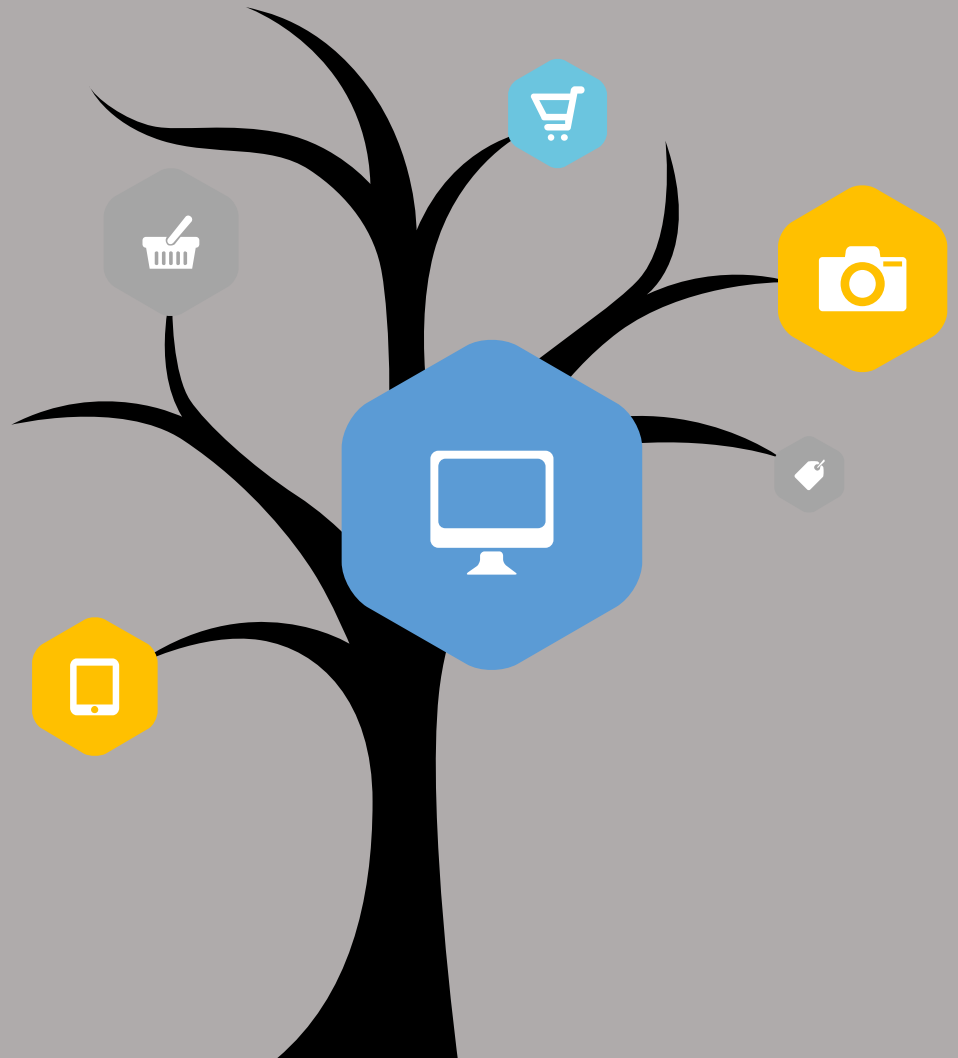


AddFood（Item_id, quantity）

- 参考领域模型分析并说明：在记录顾客所点菜品的同时，说明菜品的对象被创建；
  - 同时隐含说明订单与菜品之间的关联被创建；
  - 除此之外，为了能显示该菜品的详细信息和价格，订单还必须通过菜品Id与菜品描述概念类建立关联；
  - 顾客所点菜品的数量，比如5碗米饭等，将被赋值；

系统事件	AddFoodItem（ItemId, quantity）
交叉引用	订单处理
前置条件	服务员正在处理订单
后置条件	1. 一个新的（概念类）菜品被创建； 2. 菜品与订单建立关联； 3. 订单与菜品描述建立关联； 4. 菜品属性被修改：quantity；

课后练习

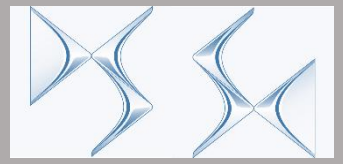


请各位同学练习以下系统事件的操作契约

AddSpecialTaste(text);

ListOrder(OrderNo);

PrintOrder(OrderNo);



总结

- 面向对象的需求分析结果：需求规格说明书，由以下两个部分组成
- 用例模型由以下四个部分构成：
  - 用例图，UML use-case diagram
  - 用例说明
  - 系统顺序图，UML sequence diagram
  - 操作契约
- 领域模型由以下两个部分构成：
  - 业务背景知识：概念类及概念类之间关系构成的类图，UML class diagram
  - 业务流程：由UML活动图表示的业务对象之间为了完成某个活动所执行的一系列子活动和动作序列，参见附录附录三