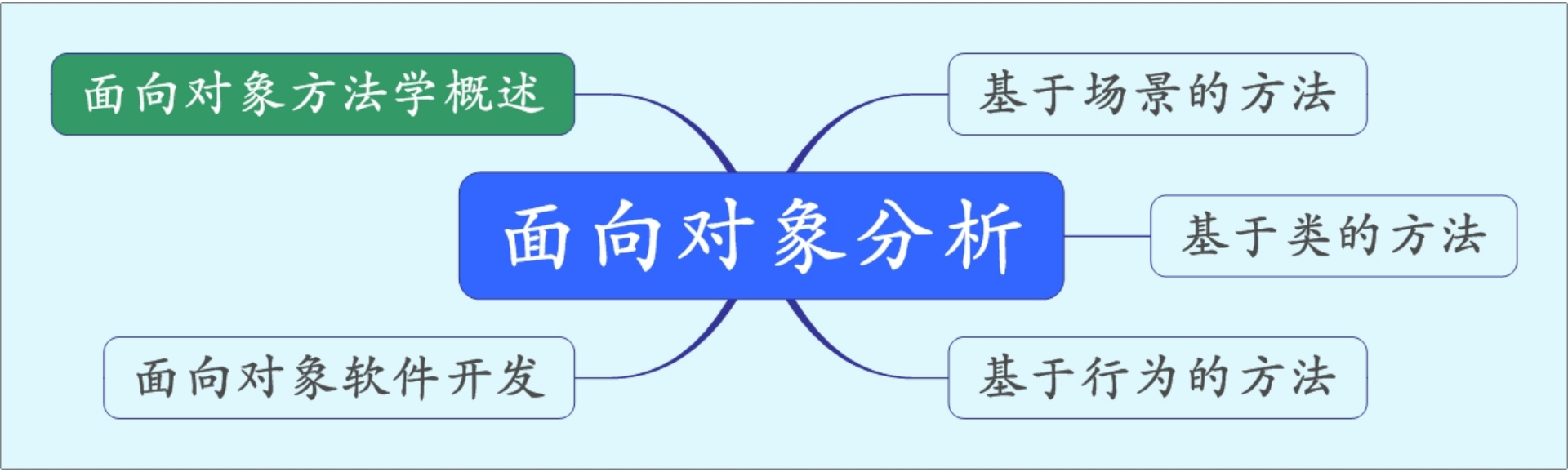
面向对象分析
(Object-Oriented Analysis, OOA)

outline



传统方法的缺点

- 结构化分析方法：**面向功能**
  - 软件结构严重依赖于功能
  - 功能是软件开发中最不稳定的因素
  - 数据和操作相分离

面向对象的发展历史

- 雏形阶段
  - 1960’s Simula-67编程语言
  - 1970’s Smalltalk编程语言
- 完善阶段
  - 1980’s：理论基础，OO 编程语言（C++, Objective-C, Object C等）
- 繁荣阶段
  - 1990’s:面向对象分析和设计方法（Booch, OMT, OOSE）Java 语言
  - 1997: OMG 组织的统一建模语言（UML）
  - 逐渐替代了传统的结构化方法

面向对象方法学的要点

- 尽可能模拟人类习惯的思维方式，使开发软件的方法与过程尽可能接近人类认识世界解决问题的方法与过程。
- 使描述问题的问题空间(问题域)与实现解法的解空间(求解域)在结构上尽可能一致。

面向对象方法学的要点

- 认为客观世界是由各种**对象**组成的，任何事物都是对象。
- 把所有对象都划分成各种对象**类**(类，class)，每个对象类都定义了一组数据和一组方法。
- 按照子类与父类的关系，把若干个对象类组成一个层次结构的系统(**继承**)。
- 对象彼此之间仅能通过传递**消息**互相联系。
- **OO =objects+class+inheritance+communication**

面向对象方法组成

- OOSD（Object-Oriented Software Development）
  - OOA(Object-Oriented Analysis)面向对象的分析

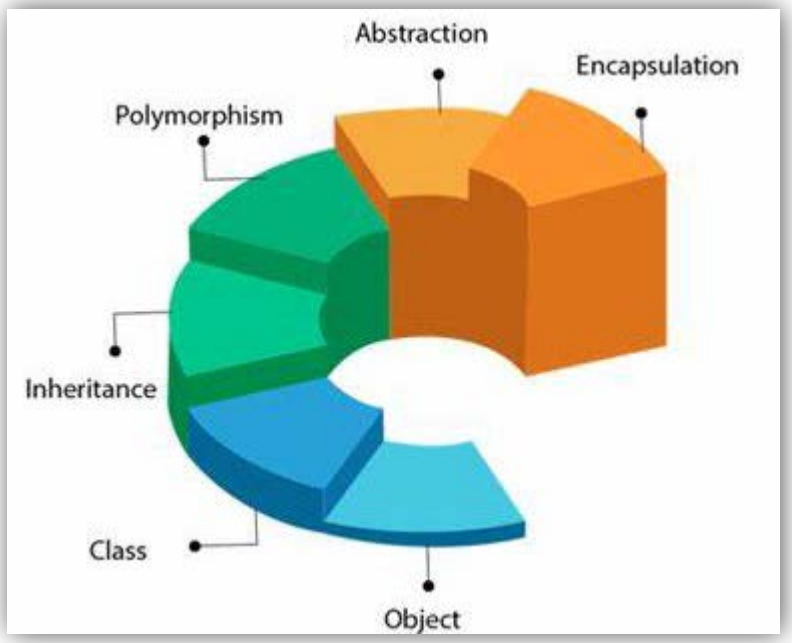
    强调对一个系统中的对象特征和行为的定义,建立系统的三类模型。
  - OOD(Object-Oriented Design)面向对象的设计

    与OOA密切配合顺序实现对现实世界的进一步建模。
  - OOP (Object-Oriented Program)面向对象的程序设计

    是面向对象的技术中发展最快的，使用面向对象的程序设计语言，进行编码。

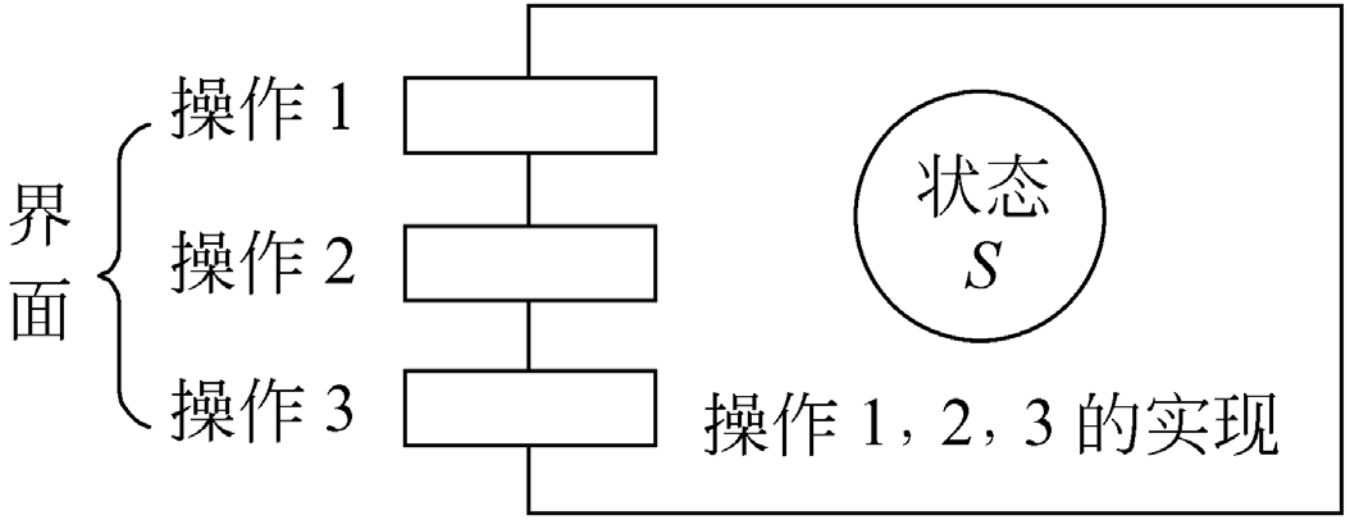
面向对象的基本概念

- 对象(object)
- 类(class)
- 消息(message)
- 封装(encapsulation)
- 继承(inheritance)
- 多态(polymorphism)

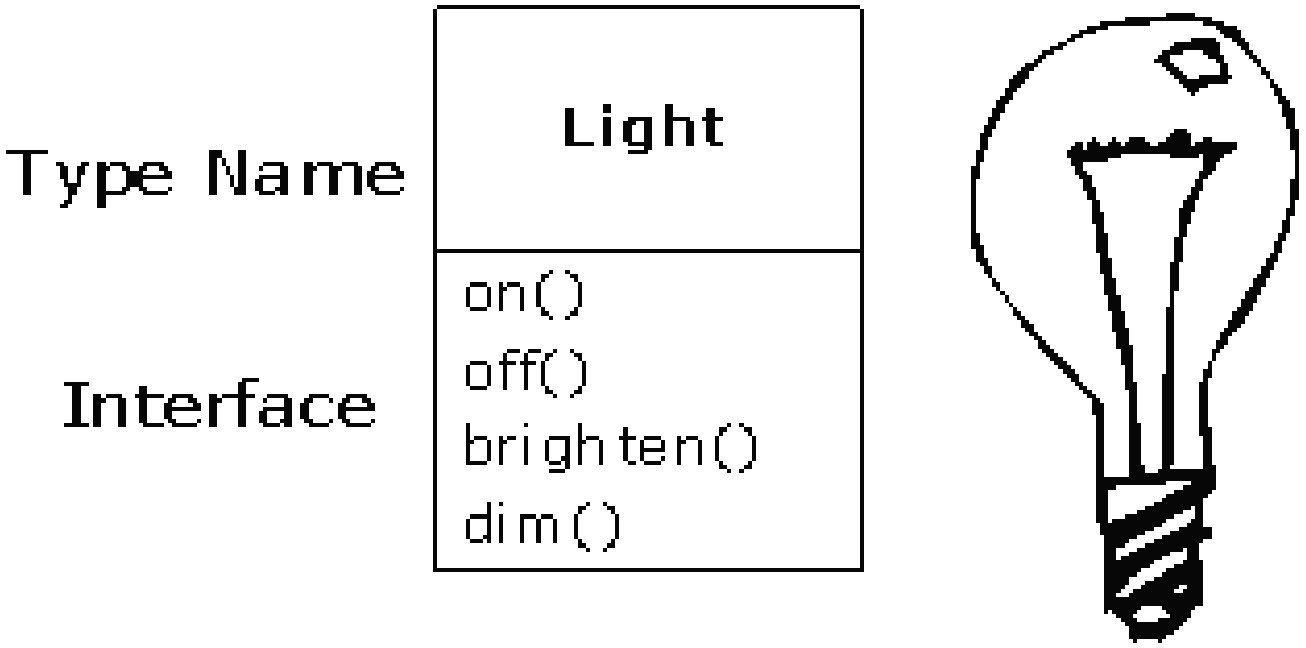


面向对象基本概念1—对象

- 对象是系统中用来描述客观事物的一个实体，它是用来构成系统的一个基本单位。
- 对象由一组属性和一组行为（方法）构成。
  - 属性: 用来描述对象静态特征的数据项。
  - 行为: 用来描述对象动态特征的操作序列。



An object has an interface




```
Light lt = new Light();
lt.on();
```

面向对象基本概念2—类

- 具有相同属性和服务的一组对象的集合，物以类聚。
- 为属于该类的全部对象提供了抽象的描述，包括属性和行为两个主要部分。
- 类与对象的关系
  抽象和具体的关系，类是对象抽象的结果。一个属于某类的对象称为该类的一个实例。

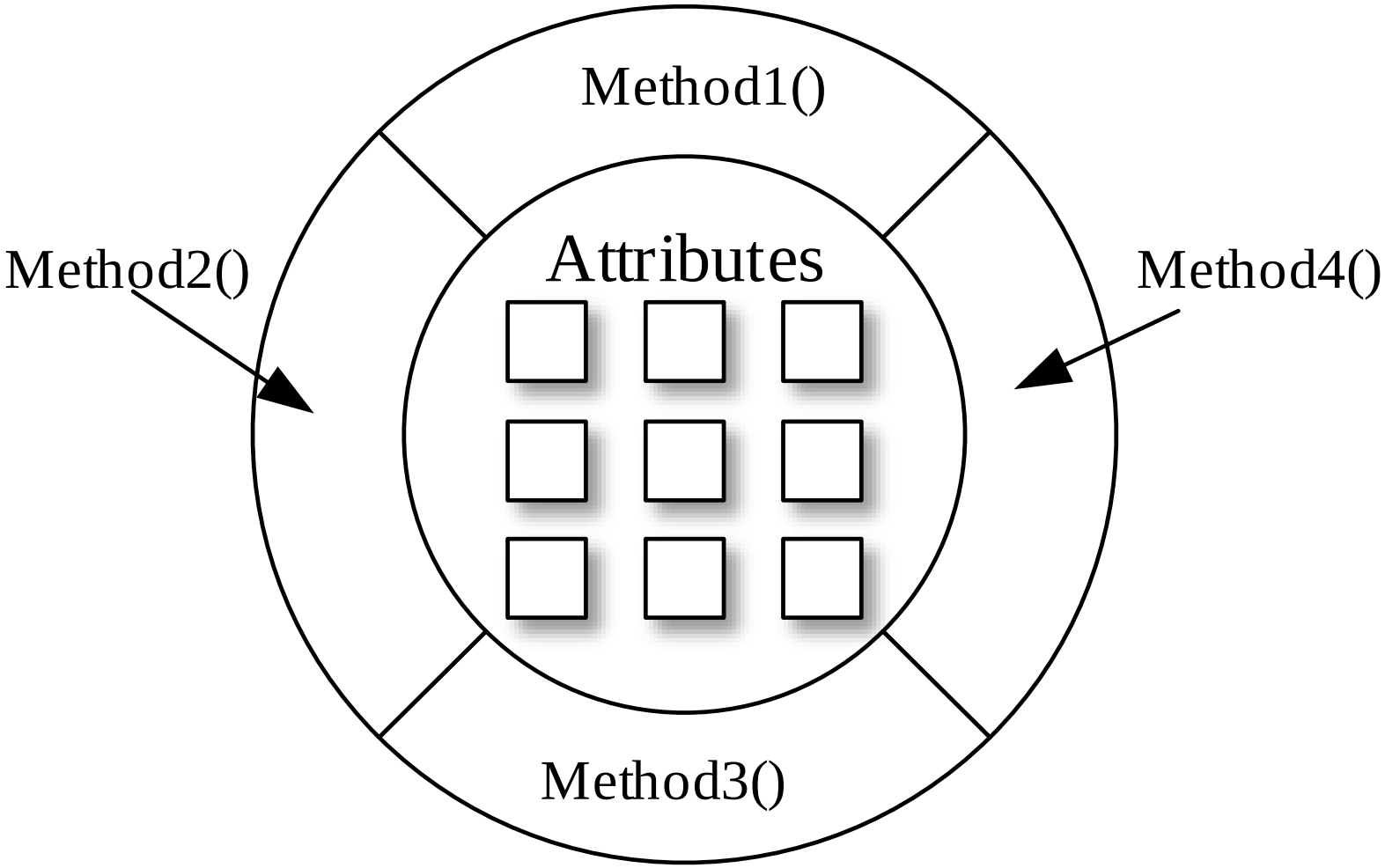
面向对象基本概念3—消息

- 消息：合作之道（message）
  - 消息是发送对象1向目标对象2发送请求的载体，申请对象2的一个方法
  - 消息传递是对象与外部世界关联的唯一途径
- 一个消息由下述3部分组成：
  - 接收消息的对象；
  - 消息名；
  - 零个或多个变元。

```
例: class   PostOffice  {
        private :
              Location       location ;
              Employee     employee ;
                   ……
        public :
             void  send (Request  request, Money  payment);
             void  sell (int  goods, Money  payment) ;
                   ……
    } ;


     main ( )
     {  PostOffice         PO ;
        Request            request ;
        Money              payment ;
           ……
        PO.send (request, payment) ;
           ……
      }
```

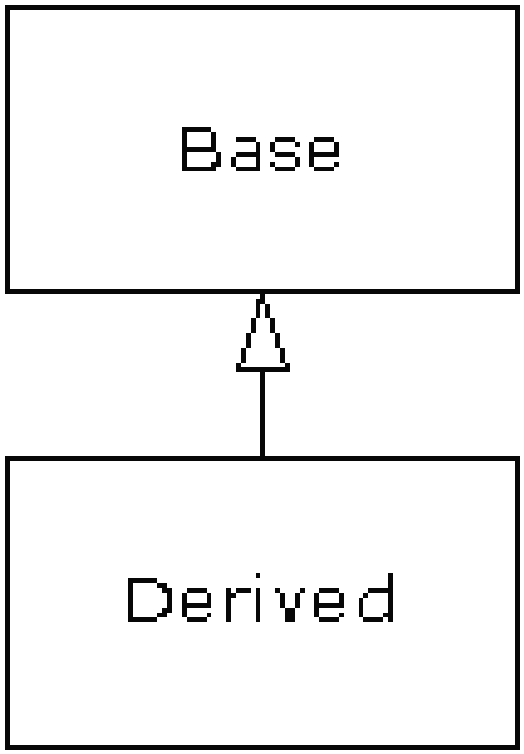
面向对象基本概念4—封装



面向对象基本概念4—封装

- 封装encapsulation是指将对象的状态信息（属性）和行为（方法）捆绑为一个逻辑单元，并尽可能隐藏对象的内部细节，使得对状态的访问或修改只能通过封装提供的接口进行
  - 把对象的全部属性和全部操作结合在一起，形成一个不可分割的独立对象
  - “信息隐藏”：尽可能隐藏对象的内部细节，对外形成一个边界，只保留有限的对外接口使之与外部发生联系

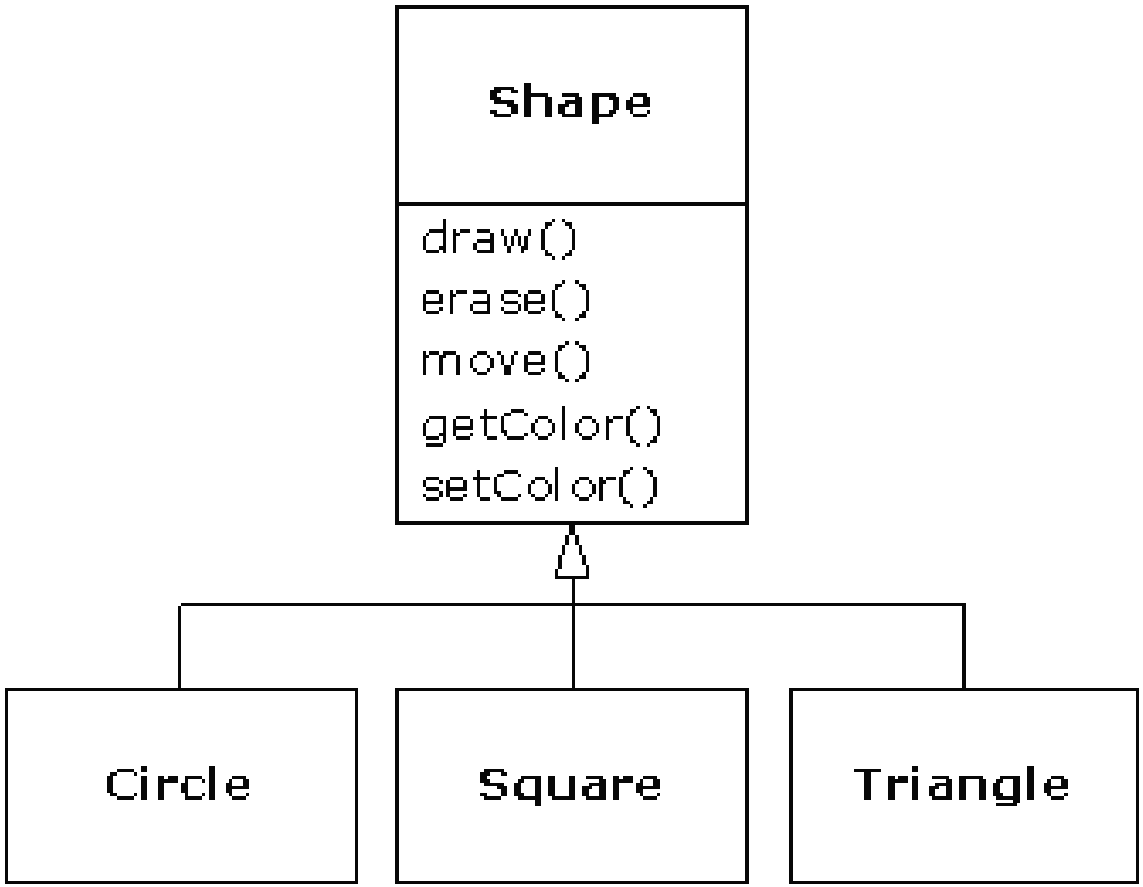
面向对象基本概念5—继承

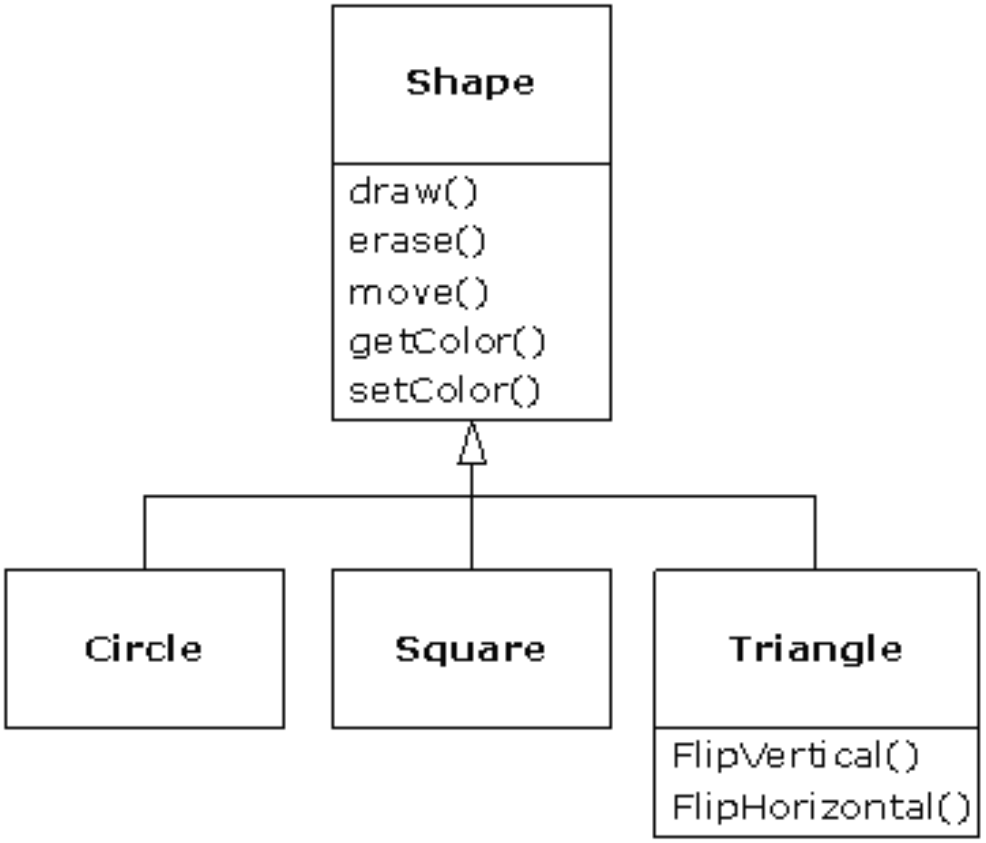


Inheritance

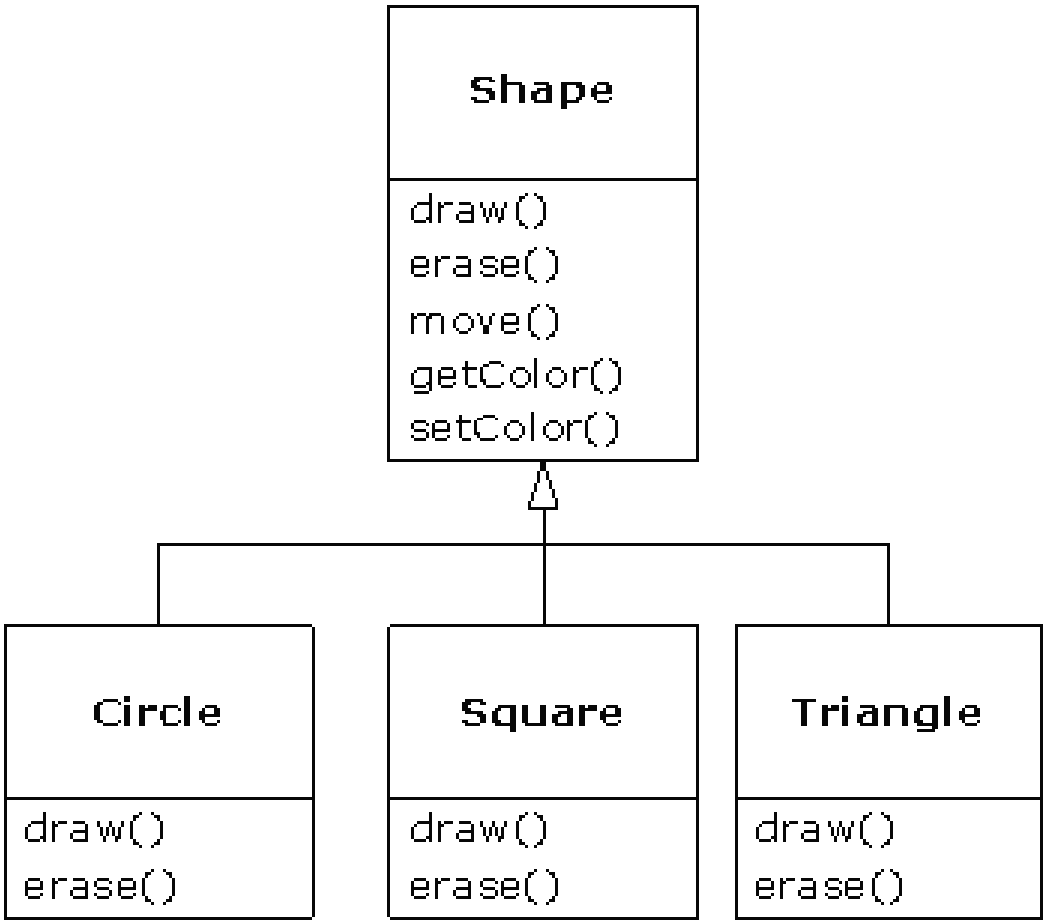
面向对象基本概念5—继承

- 继承（inheritance）对于软件复用有着重要意义，是面向对象技术能够提高软件开发效率的重要原因之一。
- 被继承的类称为超类（父类）、继承的类称为子类。
- 通过继承，子类继承了父类中的属性和操作定义，这些属性和操作就好象在子类本身中定义的一样。
- 子类在继承了父类的属性和操作的基础上还可以添加一些专属于自己的属性和操作。


Shape
draw()
erase()
move()
getColor()
setColor()
Circle
Square
Triangle



Adding new methods

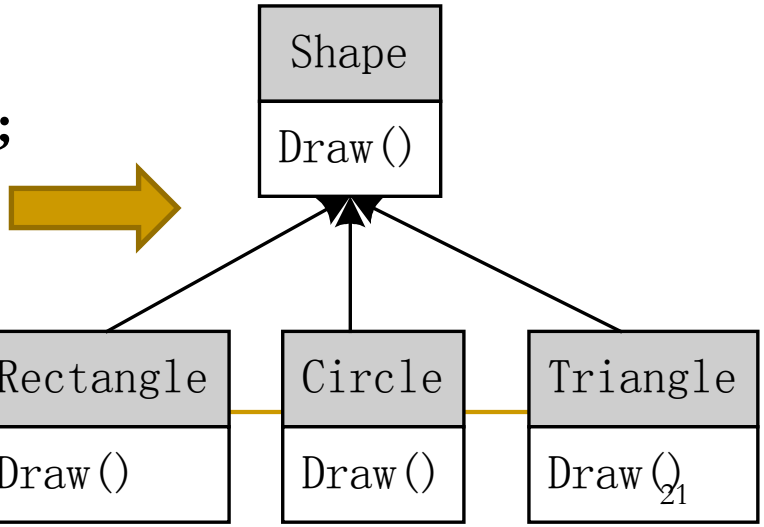


overriding methods

面向对象基本概念6—多态

- 多态性polymorphism是指用相同的操作名在一个类层次的不同类中实现不同的功能。在类等级的不同层次中可以公用一个方法的名字，然而不同层次中的每个类却各自按自己的需要来实现这个行为。
- 当对象接收到发送给它的消息时，根据该对象所属于的类动态选用在该类中定义的实现算法。

```
case of shape:
  if shape = rectangle then DrawRectangle(data);
  if shape = circle then DrawCircle(data);
  if shape = triangle then DrawTriangle(data);
end case;
```

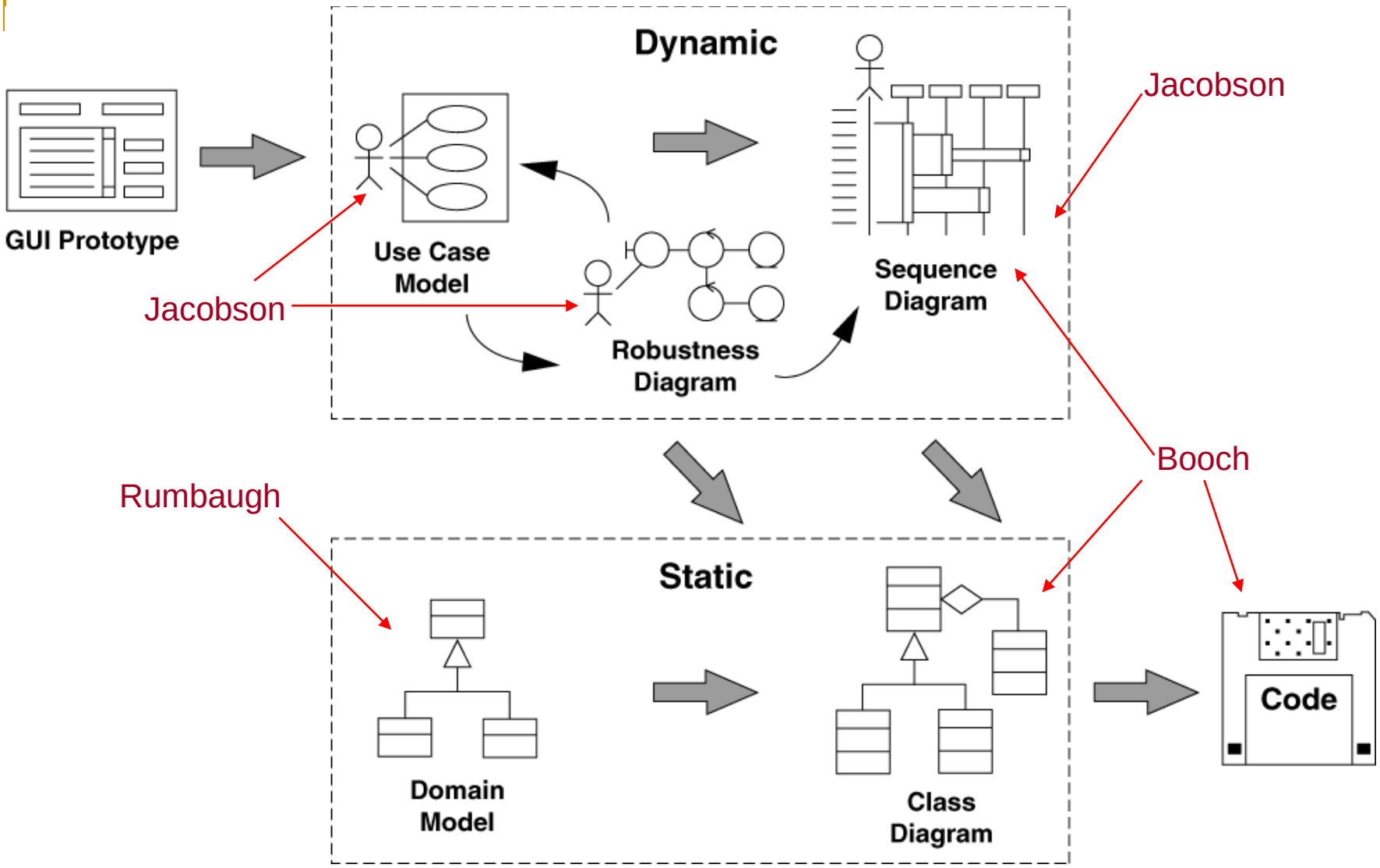


面向对象分析建模

- 回顾3种模型
  - 基于场景的模型：用例图、顺序图
  - 类模型：类图、协作图
  - 行为模型：状态图、顺序图
- 这三种模型从不同侧面描述了对系统的需求。在面向对象的分析（OOA）阶段，这三种模型是必不可少的。

流行的几种建模方法

- Booch方法
- Jacobson方法（简称OOSE）
- James Rumbaugh方法 (Object Modeling Technology, OMT)
- Coad-Yourdon方法
- ESA的HOOD方法
- Wirfs-Brook的RDD方法


Dynamic
GUI Prototype
Use Case Model
Robustness Diagram
Sequence Diagram
Jacobson
Jacobson
Booth
Rumbaugh
Static
Domain Model
Class Diagram
Code

统一建模语言UML

- UML(Unified Modeling Language)产生于90年代中期
- 统一了Booch、OMT和OOSE方法中的概念和表示法，并对其作了进一步扩展
- UML是一个通用的标准建模语言，可以对任何具有静态结构和动态行为的系统进行可视化建模
- UML不是一个开发过程，也不是一个方法，但允许任何一种开发过程和面向对象方法使用它。

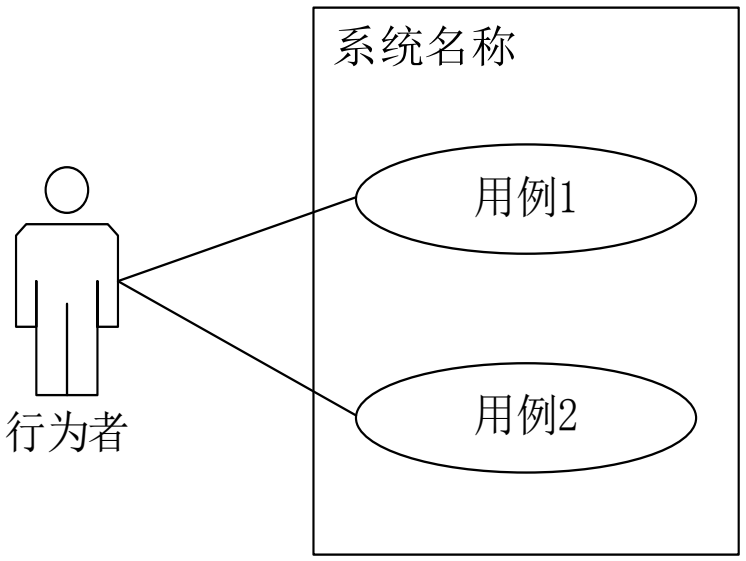
UML的历史

- 80年代末期到90年代，各种OO软件开发方法纷纷涌现：Booch，OMT，OOSE…
- 1994年10月，Booch和Rumbaugh统一Booch93和OMT2，发布UML0.8
- 1995年秋，Jacobson加盟Rational，1996年6月发布UML0.9，1996年10月发布UML0.91
- 1997年UML获众多著名软件公司的支持，11月被OMG接纳为标准
- 1998年UML发展到UML1.4，相应的软件开发环境Rational Rose正式推出
- 2003年UML发展到UML2.0
- 2010年 UML2.3

用例图

- **用例图**是进行需求分析和建立功能模型的强有力工具
- 从用户的角度描述系统的功能
  - 外部行为者（actor）所理解的系统功能
  - 用例模型的建立是系统开发者和用户反复讨论的结果

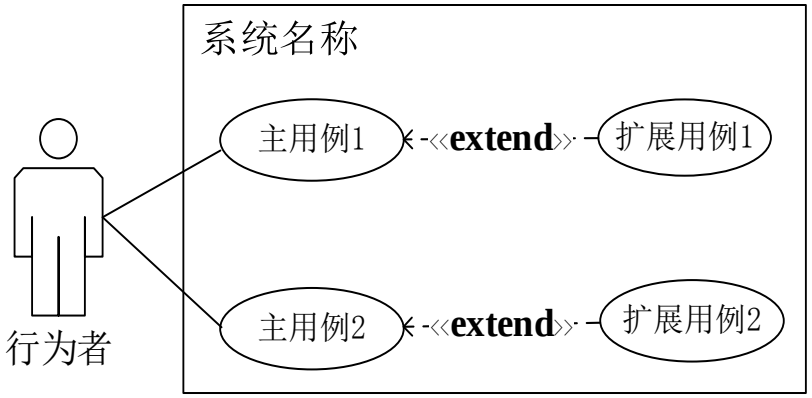
用例图

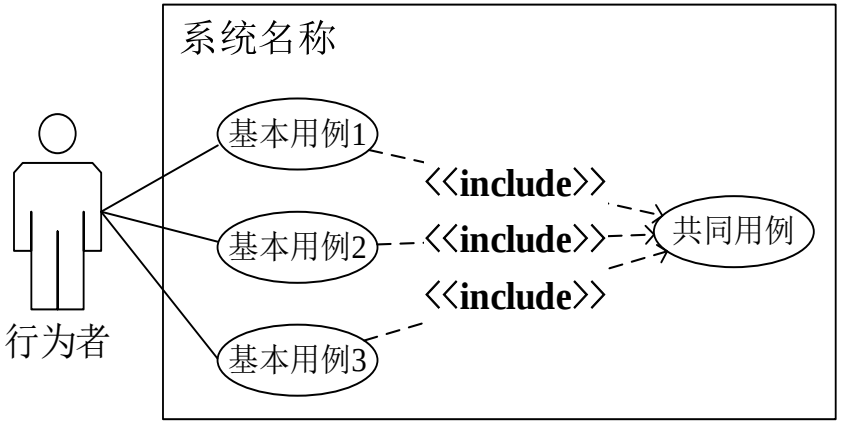


- 用例图的模型元素有**系统**、**行为者**、**用例**及**用例之间的关系**
  - 系统：一个提供用例的黑盒子
  - 行为者：与系统交互的人或其他系统，它代表外部实体
  - 用例：可以被行为者感受到的、系统的一个完整的功能

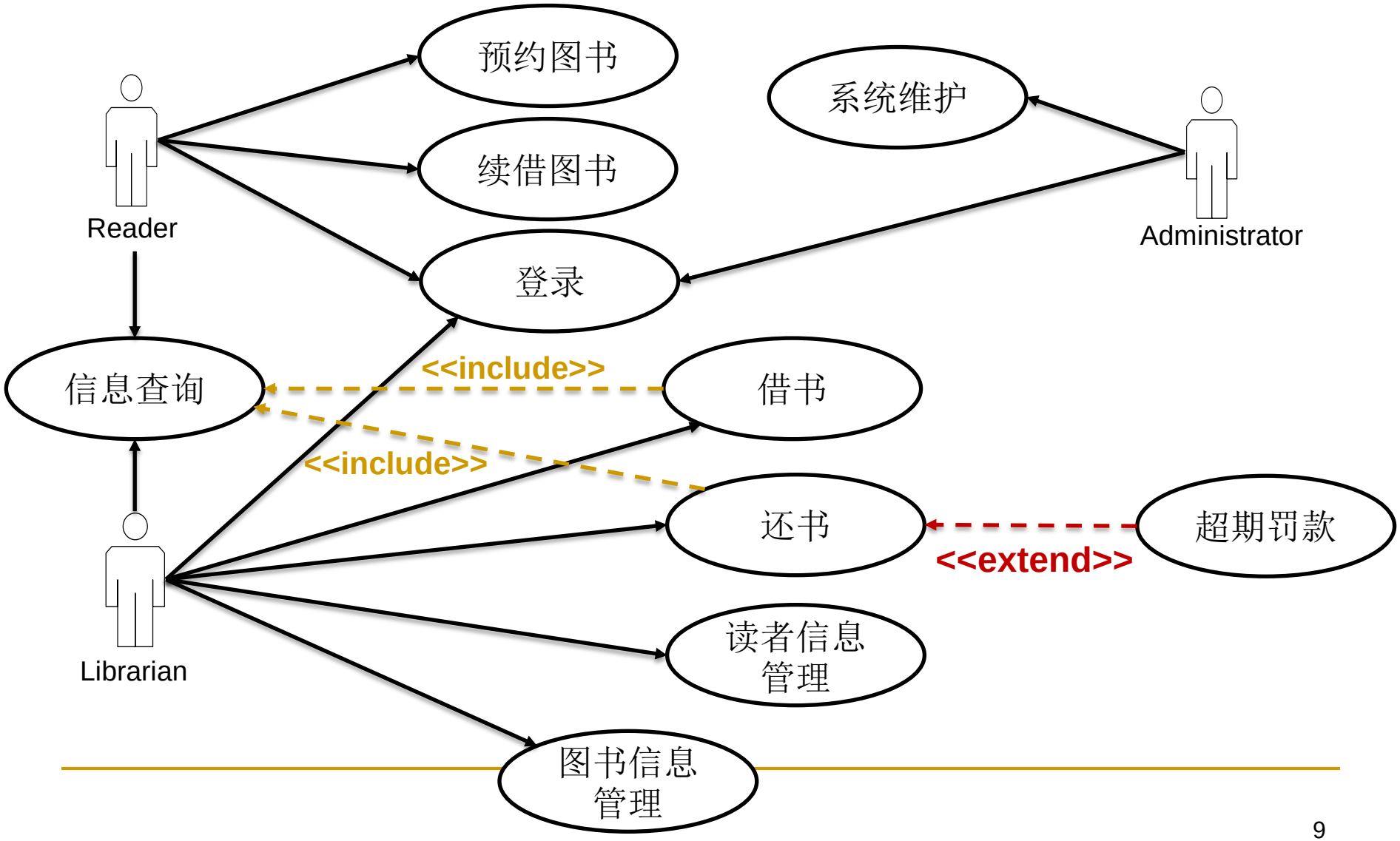
用例图

- 用例之间的关系有**扩展**和**包含**两种
  - extend：将主用例的异常行为或可选分支抽象为扩展用例
  - include：将不同用例的共同行为提取为一个用例



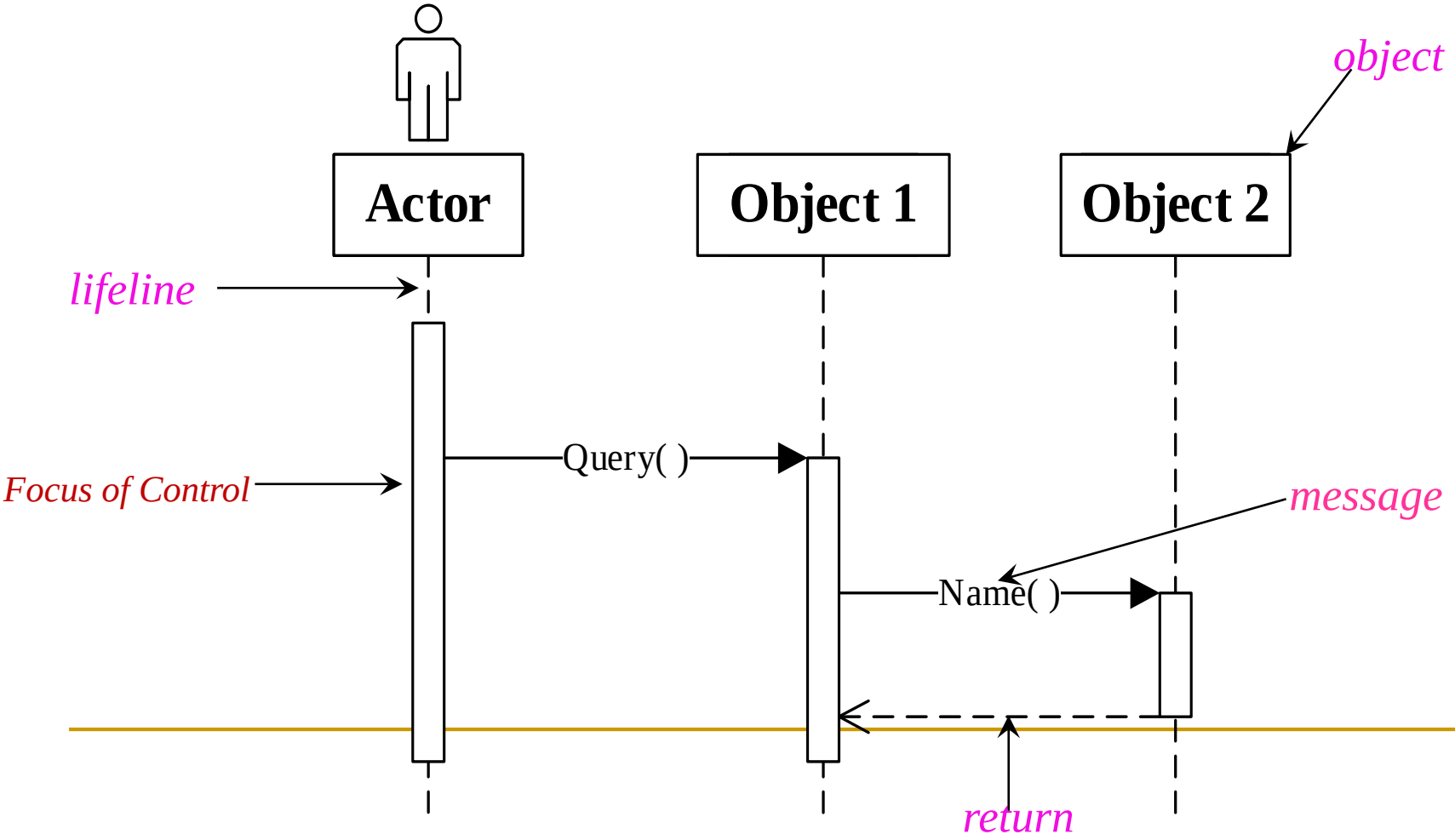


图书管理系统用例图

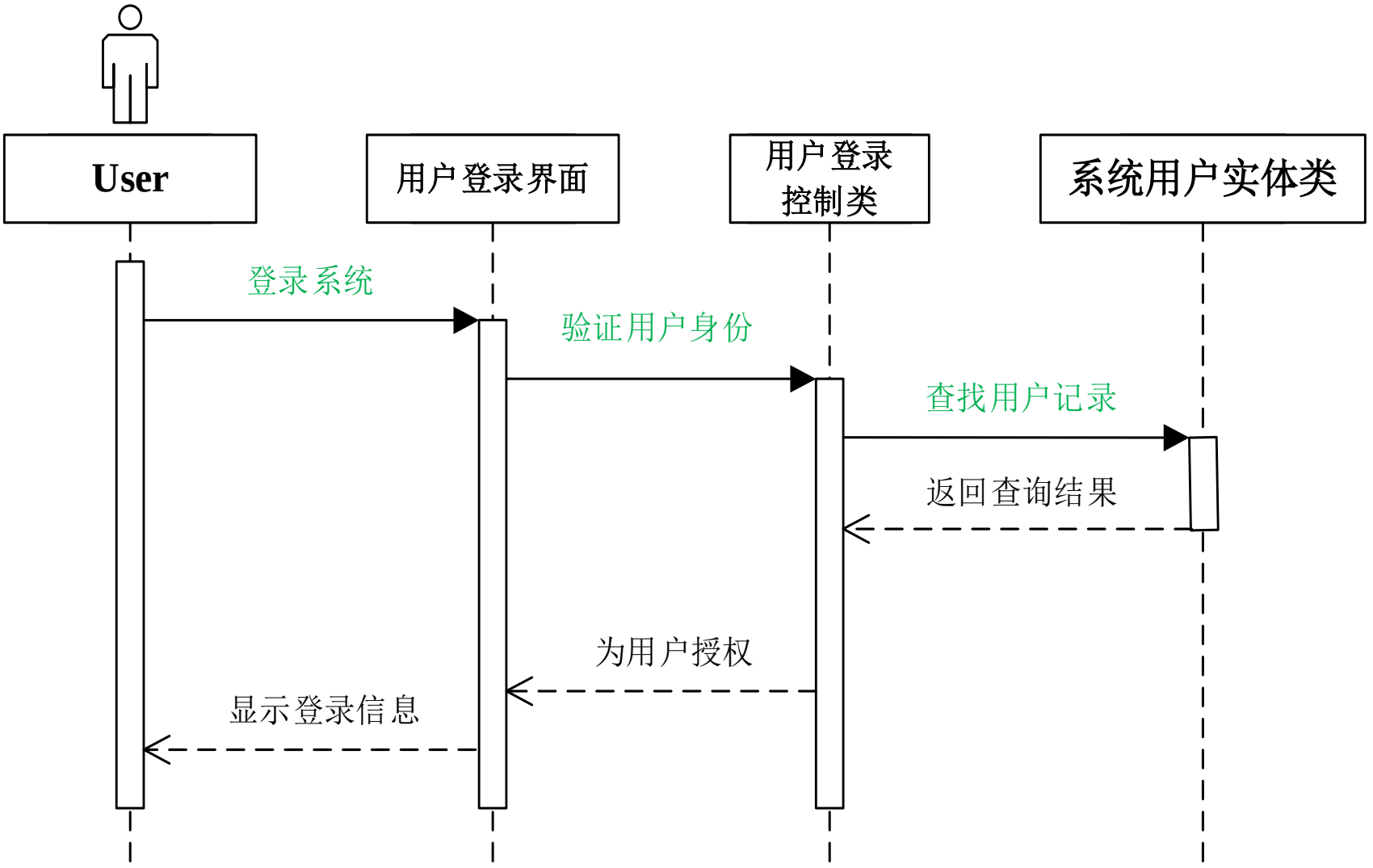

预约图书
续借图书
登录
系统维护
Reader
Administrator
信息查询
借书
<<include>>
<<include>>
还书
超期罚款
<<extend>>
Librarian
读者信息
管理
图书信息
管理

顺序图（Sequence Diagram）

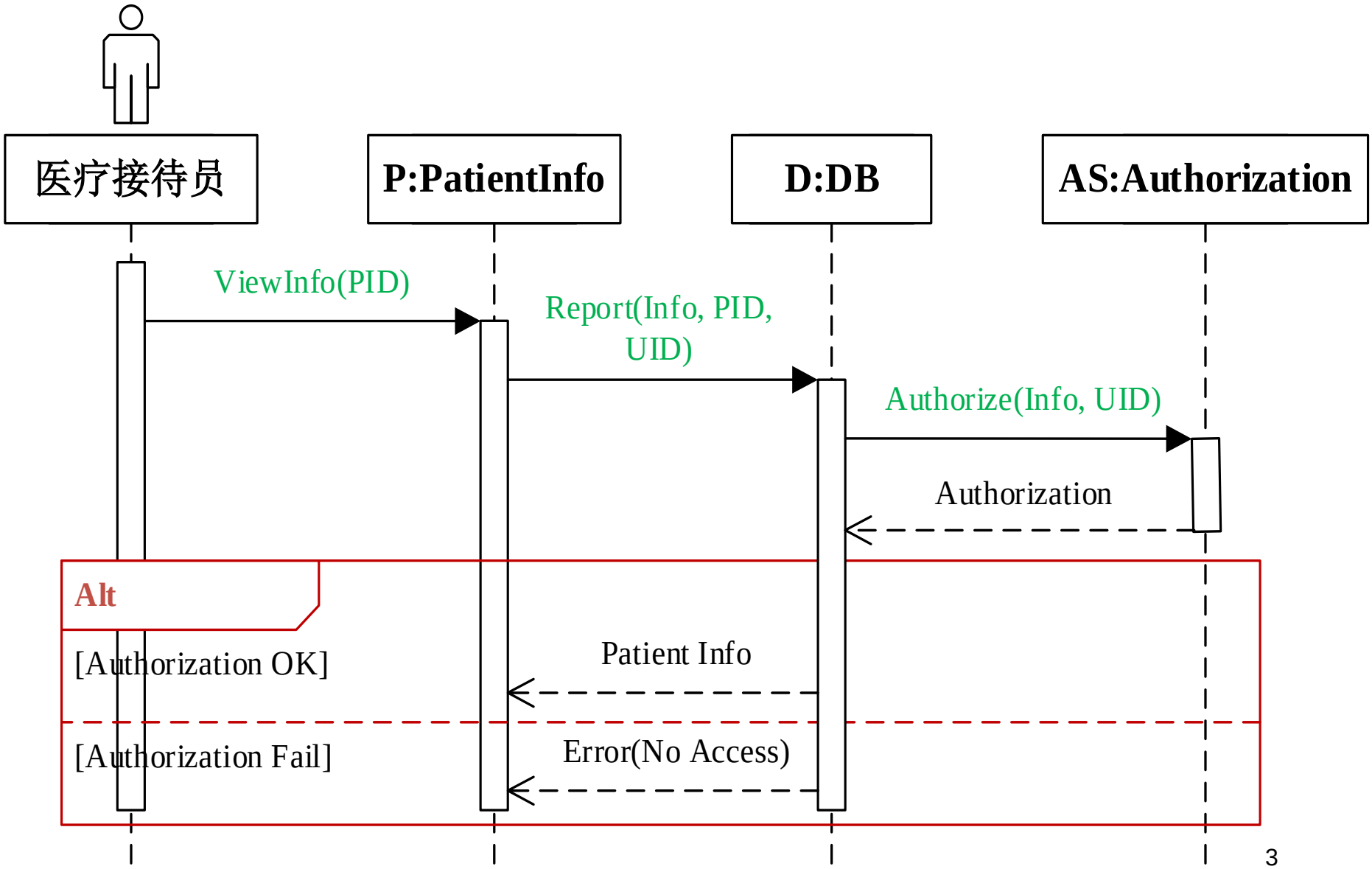
- 对系统中对象间的**交互**建模
- 对用例图的补充说明，描述用例中交互的顺序



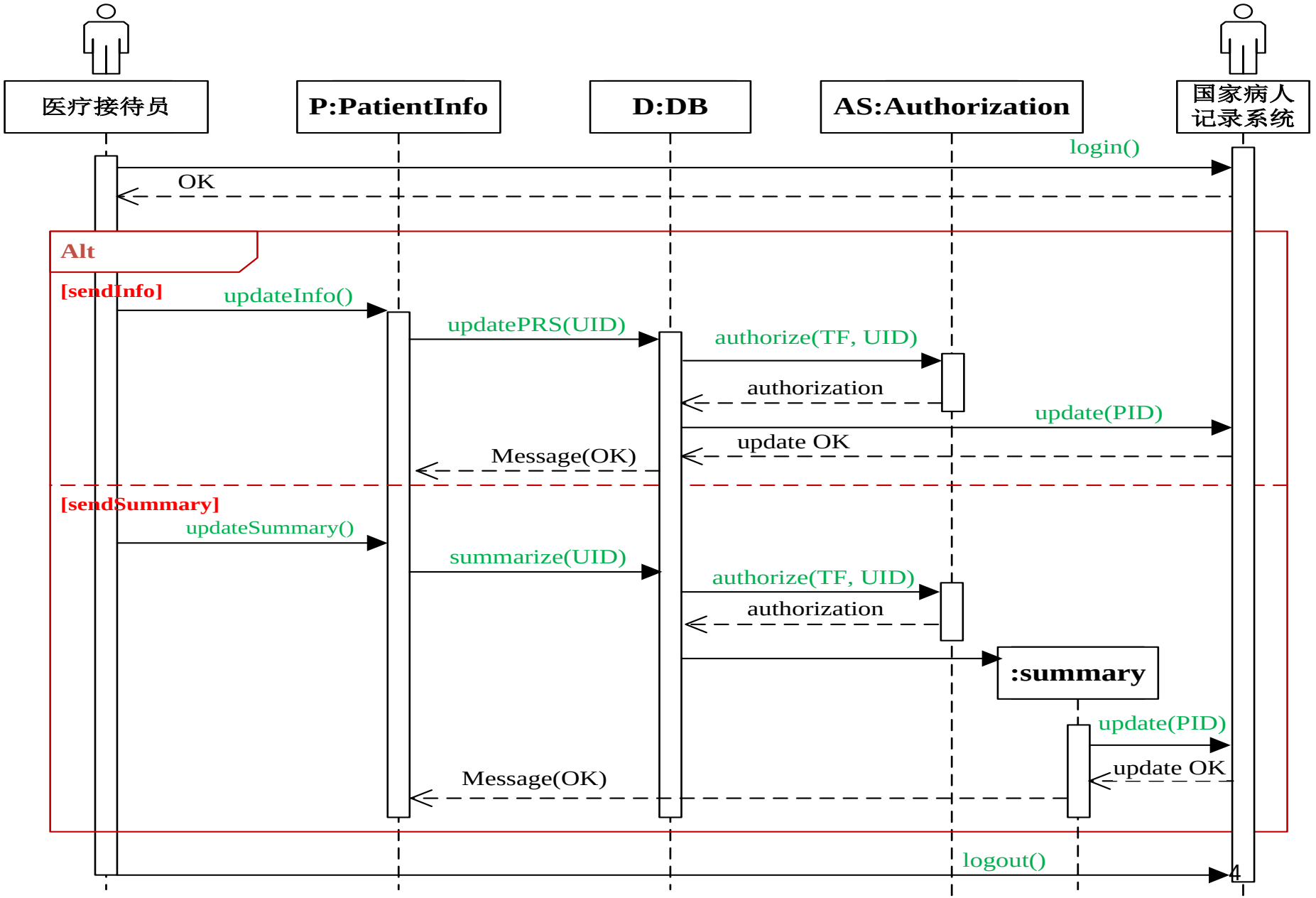
例1：用户登录


User
用户登录界面
用户登录
控制类
系统用户实体类
登录系统
验证用户身份
查找用户记录
返回查询结果
为用户授权
显示登录信息

例2：查看病人信息


医疗接待员
P:PatientInfo
D:DB
AS:Authorization
ViewInfo(PID)
Report(Info, PID, UID)
Authorize(Info, UID)
Authorization
Alt
[Authorization OK]
Patient Info
[Authorization Fail]
Error(No Access)

例3：上传数据


医疗接待员
P:PatientInfo
D:DB
AS:Authorization
国家病人记录系统
login()
OK
Alt
[sendInfo]
updateInfo()
updatePRS(UID)
authorize(TF, UID)
authorization
update(PID)
update OK
Message(OK)
[sendSummary]
updateSummary()
summarize(UID)
authorize(TF, UID)
authorization
:summary
update(PID)
update OK
Message(OK)
logout()

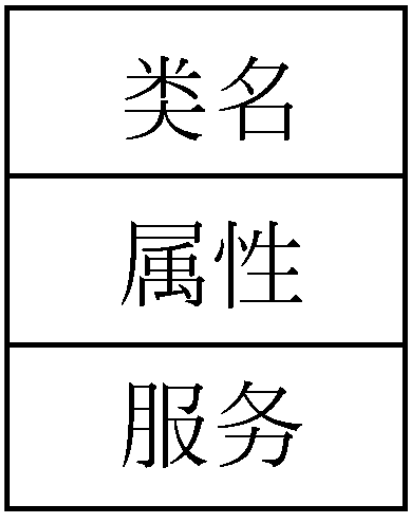
例4

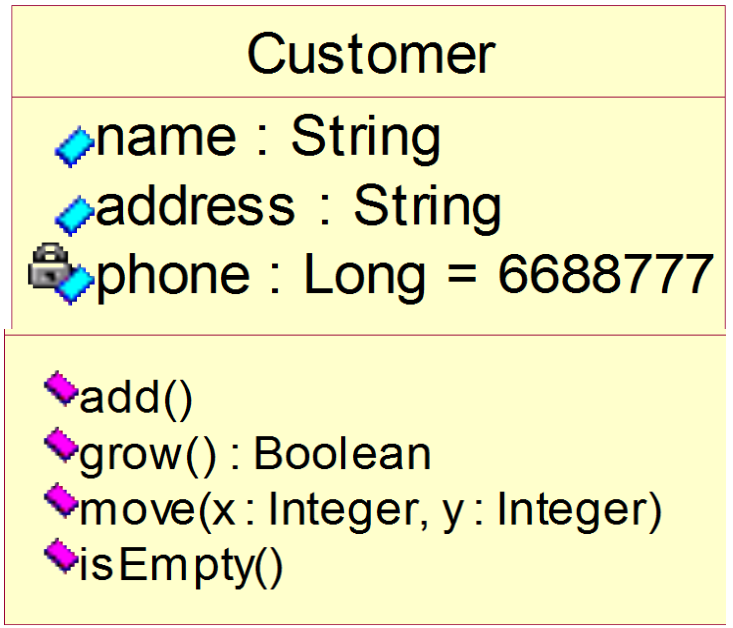
【基幹系】詳細設計書（売掛金）照会
-Web（売上内容照会）

类图

1. 定义类

UML中类的图形符号为长方形，用两条横线把长方形分成上、中、下3个区域，3个区域分别放类的类名、属性和服务





类图

2. 定义属性

属性用来描述类的特征，表示需要处理的数据。

visibility attribute-name : type = initial-value {property-string}
可见性 属性名 ：类型 = 缺省值 {性质串}

可见性(visibility)表示该属性对类外的元素是否可见。

- public（+）公有的，模型中的任何类都可以访问该属性。
- private（-）私有的，表示不能被别的类访问。
- protected（#）受保护的，表示该属性只能被该类及其子类访问。

类图

3. 定义操作

- 对数据的具体处理方法的描述
- 操作说明了该类能做些什么工作。

visibility operating-name(parameter-list): return-type {property- string}
可见性　　操作名　　（参数表）：　返回类型　　{性质串}

- 操作可见性的定义方法与属性相同。
- 参数表是用逗号分隔的形式参数的序列。描述一个参数的语法如下：

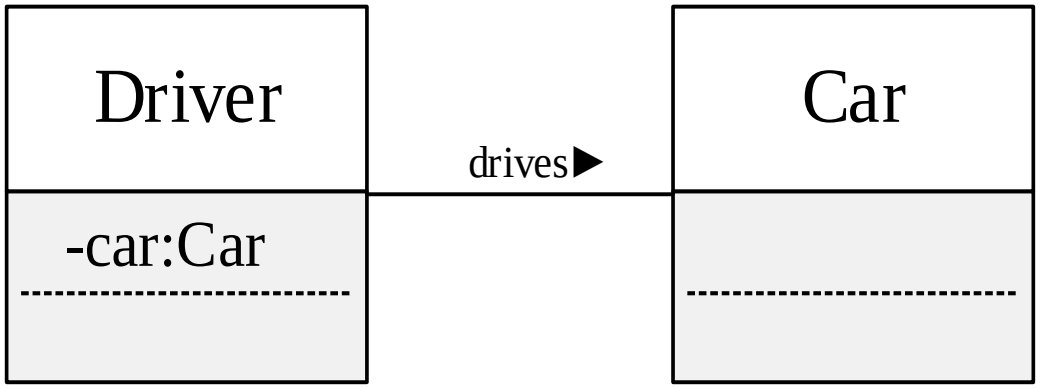
  参数名：类型名=默认值

类之间的关系

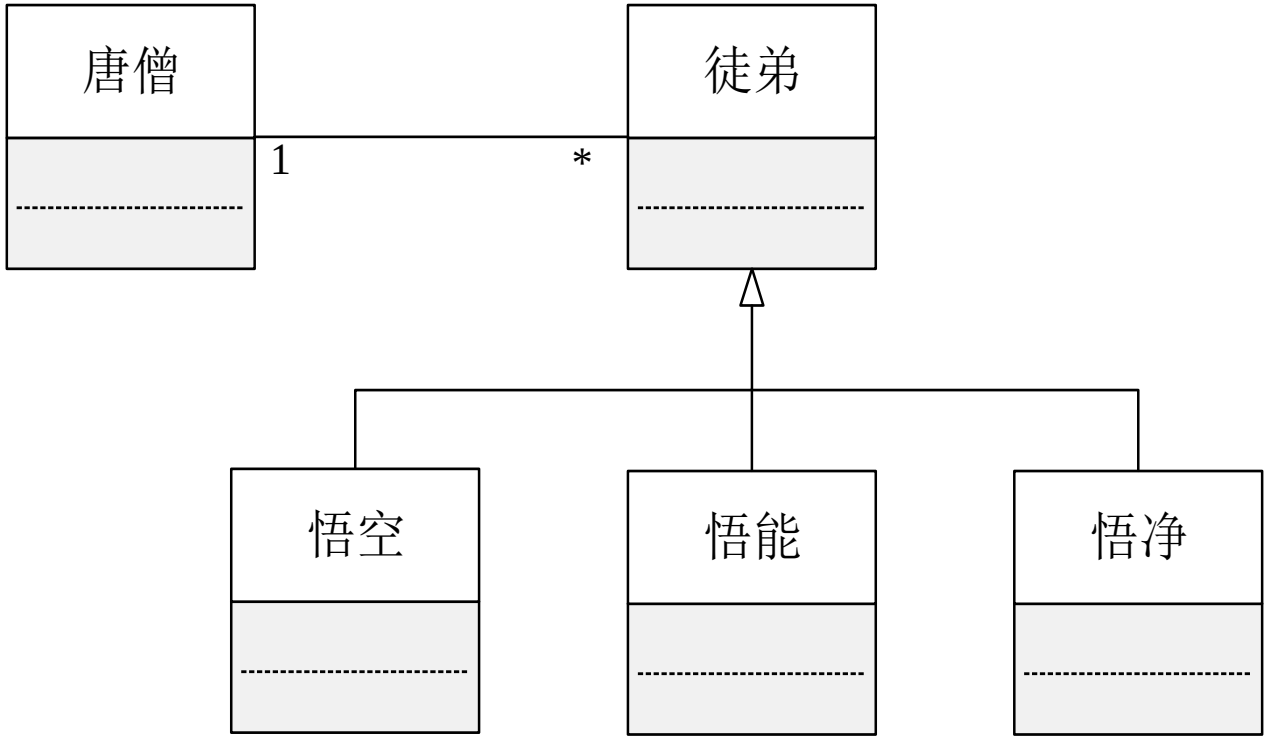
- **关联（Association）**
  - **聚合（Aggregation）**
  - **合成（Composition）**
- **泛化（Generalization）**
- **依赖（Dependency）**

关联关系

- 关联关系：类与类之间的连接，它使一个类知道另一个类的属性和方法。
- 在java中，关联关系是用实例变量实现的。
- 关联中的两个类是同一层次上的。
- 关联有方向和基数
  - 方向：单向关联和双向关联
  - 基数：实例的个数



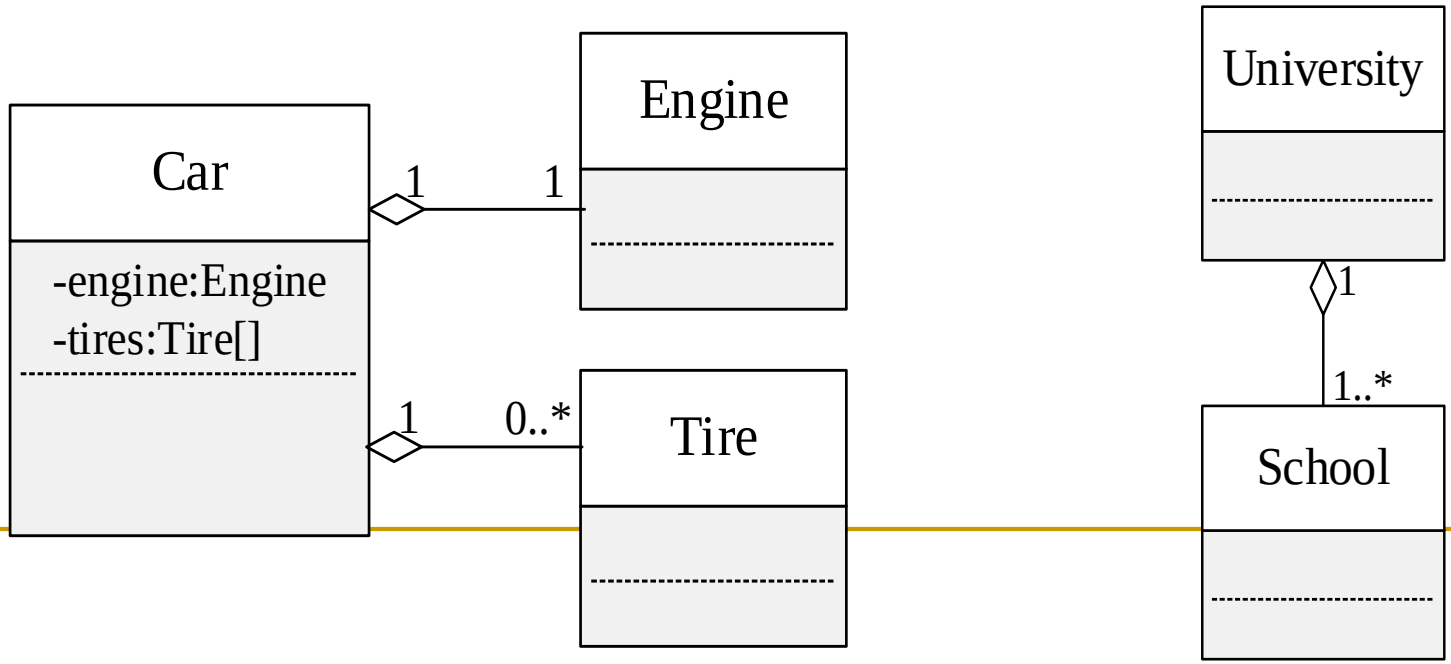
关联关系



基数	含义
0..1	零个或者一个实例
0..*或者*	对实例的数目没有限制（可以是0）
1	只有一个实例
1..*	至少有一个实例

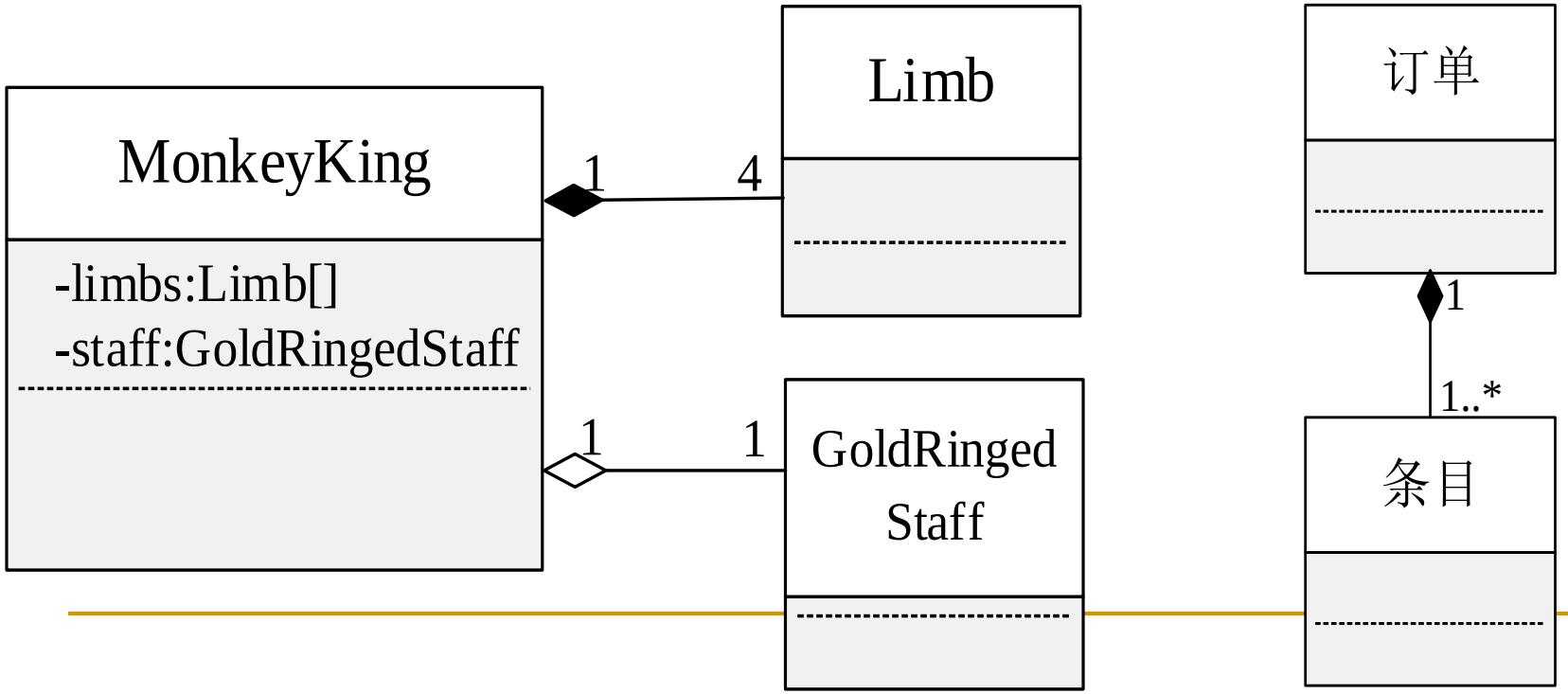
聚合关系：“has-a”

- 聚合关系：关联关系的一种
  - 强的关联关系
  - 整体和部分之间的关系
- 聚合关系是用实例变量实现的
- 聚合中的两个类处在不平等的层次上，一个代表整体，另一个代表部分。



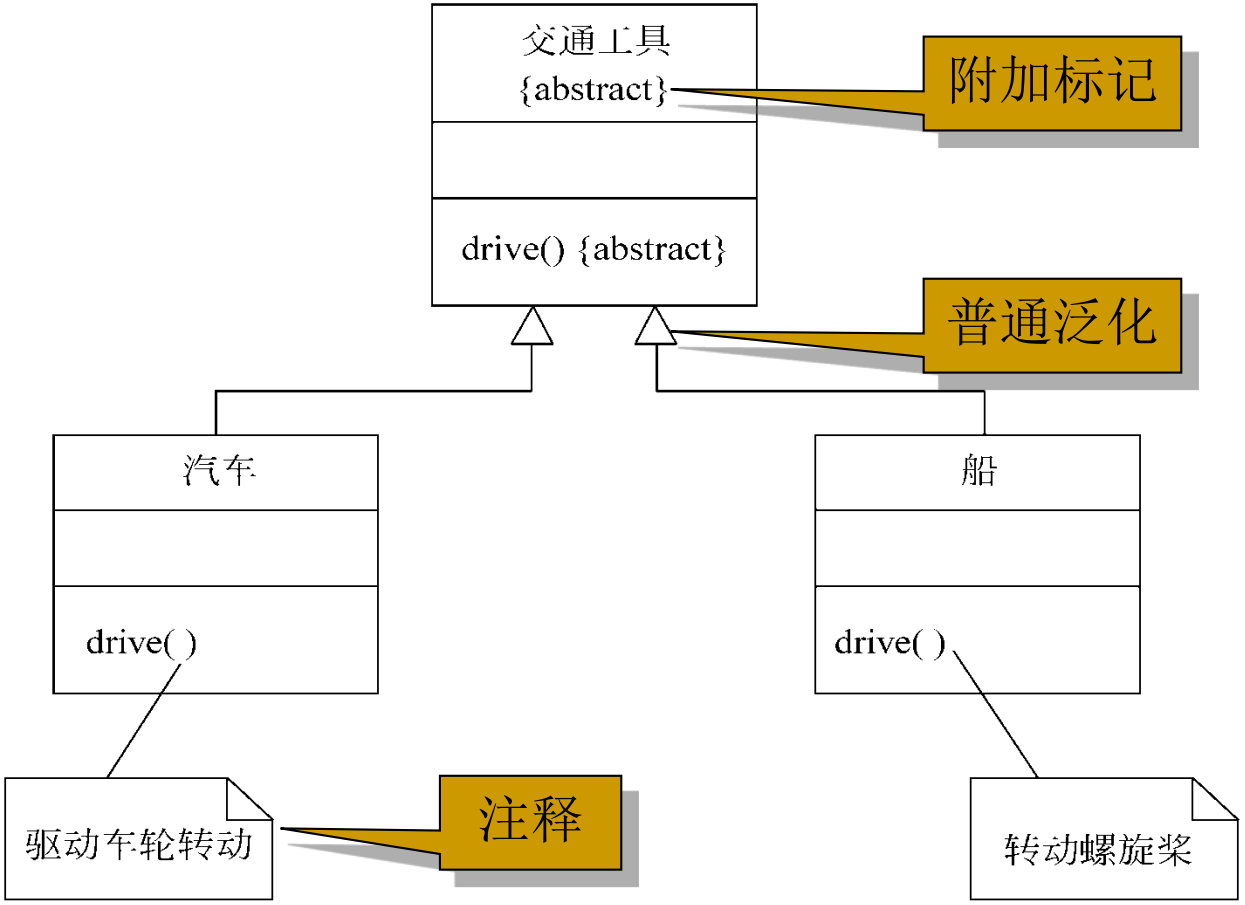
合成关系

- 合成关系：关联关系的一种，比聚合关系强
- 合成关系中代表整体的对象负责代表部分的对象的生命周期
- 当整体类的生命周期结束以后,部分类的生命周期也要结束

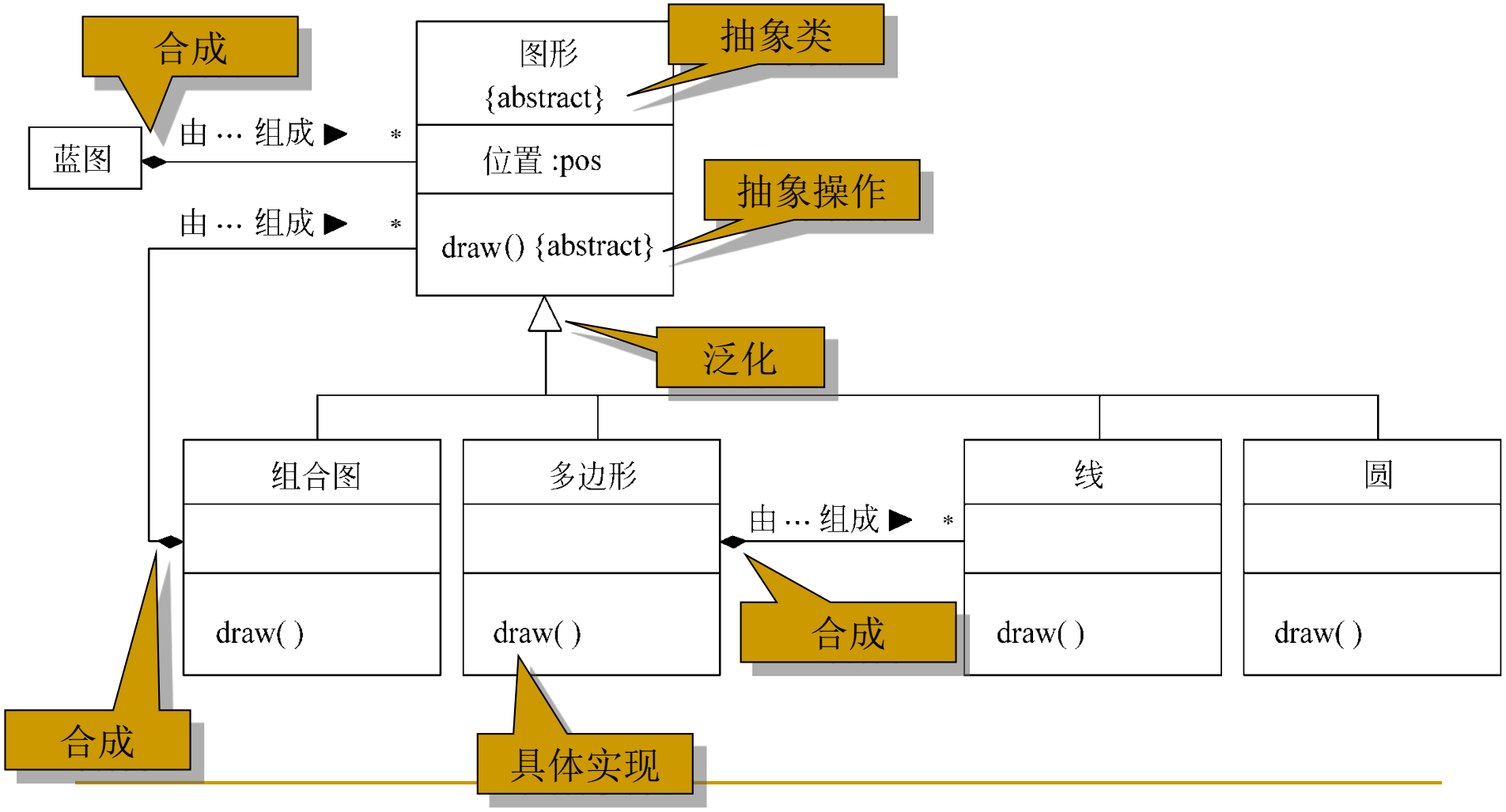


泛化：“is-a”

UML中的泛化关系就是通常所说的继承关系。

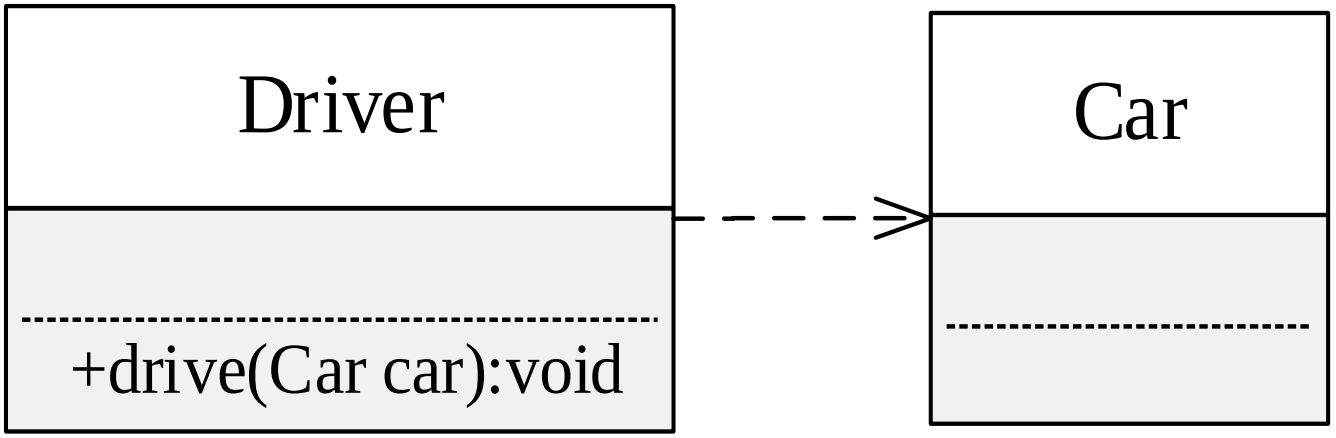


类图示例

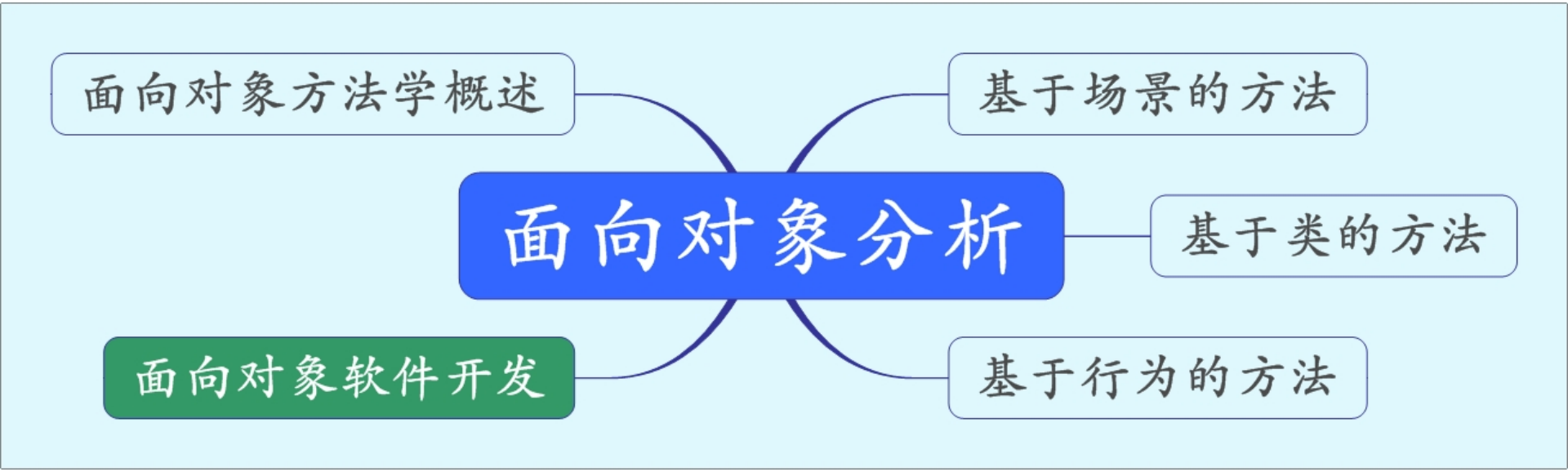

合成
图形
{abstract}
抽象类
由 … 组成 ▶
*
蓝图
位置 :pos
抽象操作
由 … 组成 ▶
*
draw() {abstract}
泛化
组合图
多边形
线
圆
由 … 组成 ▶
*
draw( )
draw( )
draw( )
draw( )
合成
合成
具体实现

依赖

- 一个类依赖于另一个类的定义。
- 在java中，依赖关系是用局部变量、方法的形参等实现的。



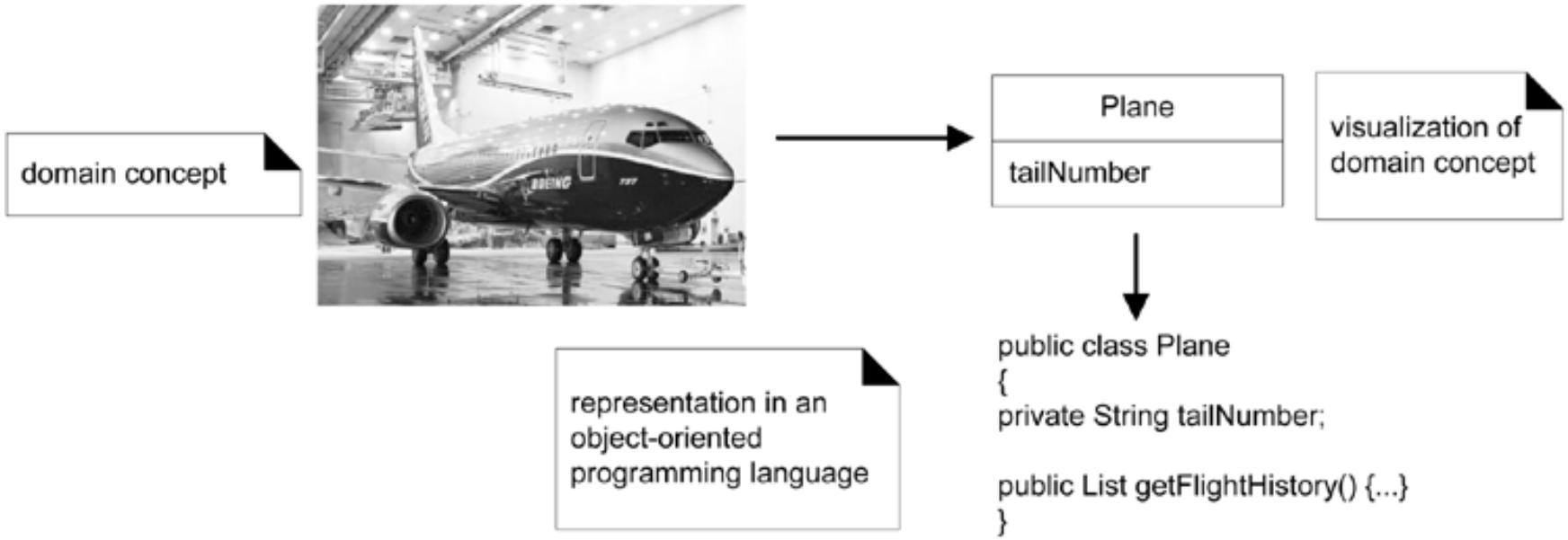
outline



面向对象软件开发

- Useful analysis and design have been summarized in the phrase do the right thing (analysis), and do the thing right (design).

面向对象软件开发



Object-orientation emphasizes representation of objects

面向对象软件开发

- During object-oriented analysis there is an emphasis on finding and describing the objects in the **problem domain**. For example, in the case of the flight information system, some of the concepts include *Plane*, *Flight*, and *Pilot*.
- During object-oriented design there is an emphasis on defining software objects and how they **collaborate** to fulfill the requirements. For example, a Plane software object may have a *tailNumber* attribute and a *getFlightHistory* method.

实例：掷骰子游戏

- 目的：通过这个简单例子，使大家对面向对象的分析和设计有一个简单认识。
- 问题描述：游戏者掷出两个骰子，如果总点数是7，他就赢了，否则就输了。

定义用例 | 定义领域模型 | 定义交互图 | 定义设计类图

实例：掷骰子游戏

第一步：定义用例

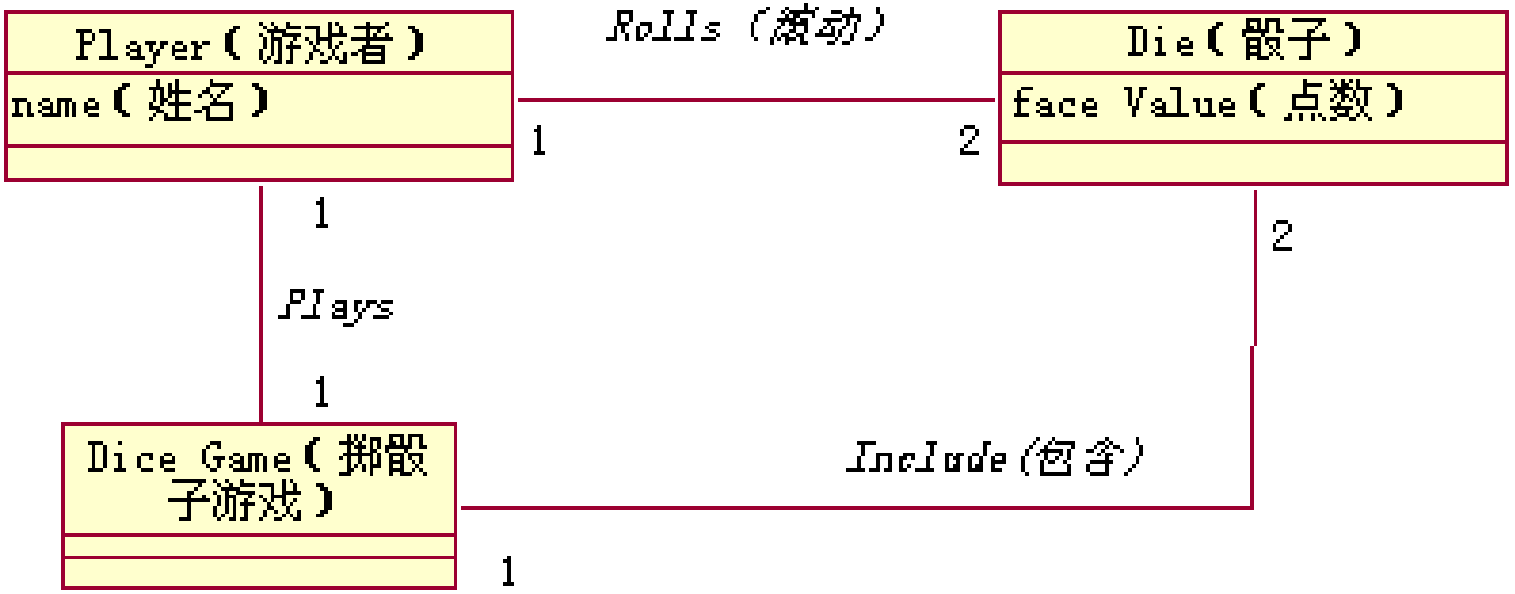
用结构化的文本形式对领域过程进行描述。
并不是面向对象开发特有。

用例：**Play a Game**
参与者：**Player**
描述：这个用例始于游戏者拾起骰子并投掷骰子。如果骰子点数是7，游戏者赢，否则输。

实例：掷骰子游戏

第二步：定义领域模型

对问题域中重要的概念、属性和关联的识别

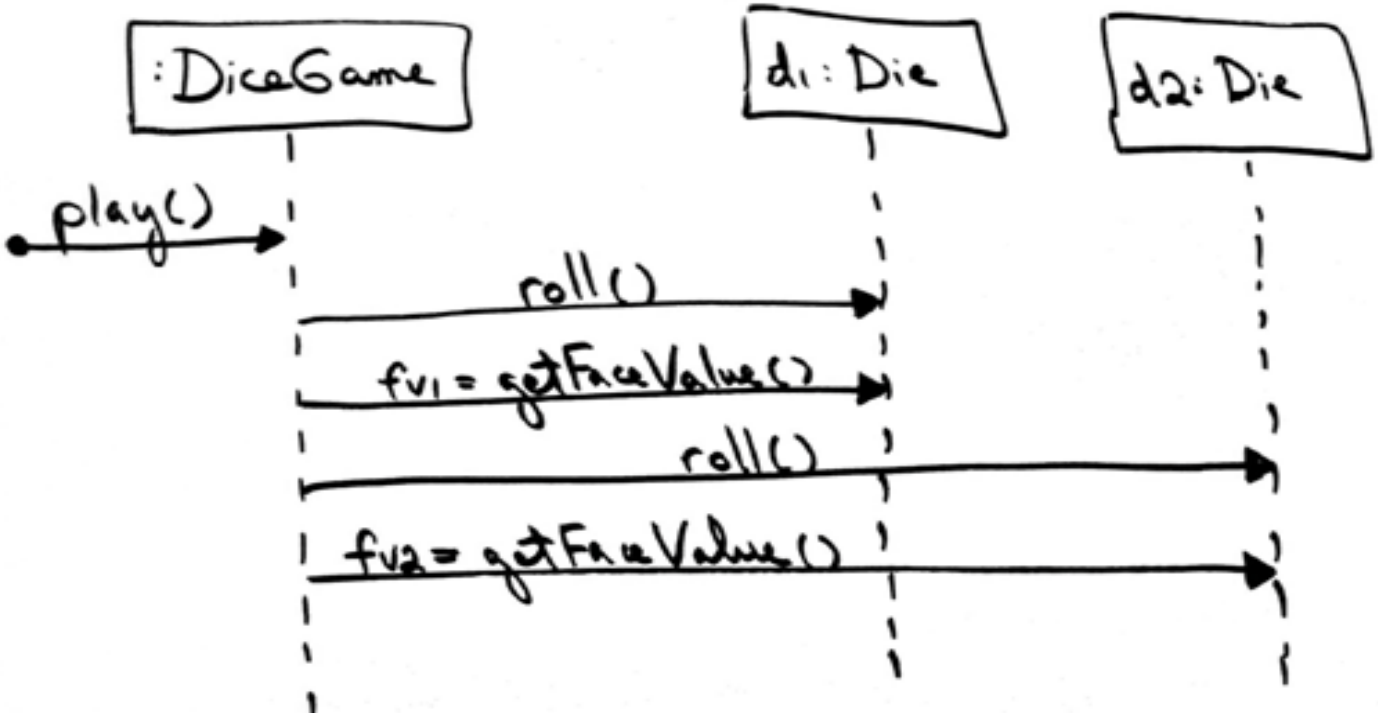


实例：掷骰子游戏

第三步：定义交互图

为对象分配职责、以及展示对象之间如何通过消息交互（用顺序图表示）

. Sequence diagram illustrating messages between software objects.

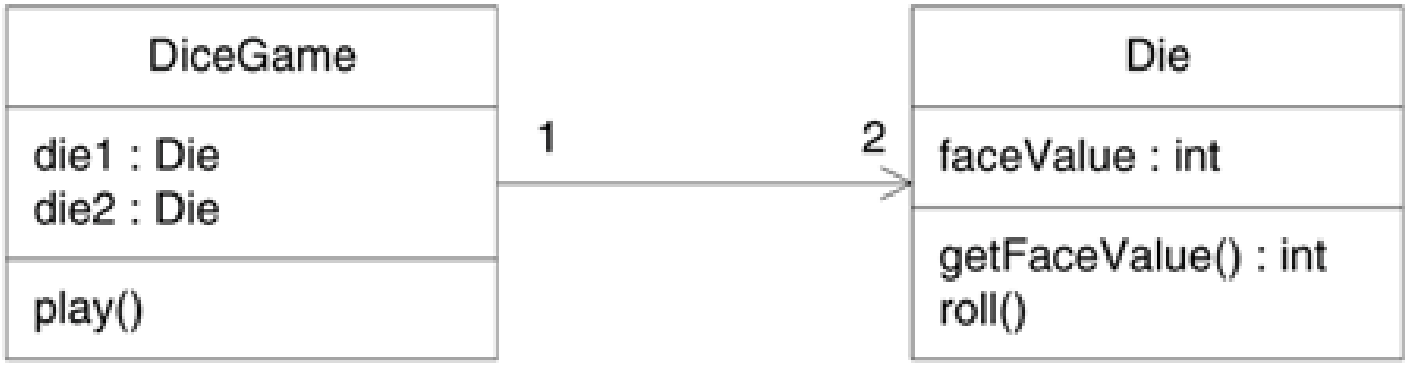


实例：掷骰子游戏

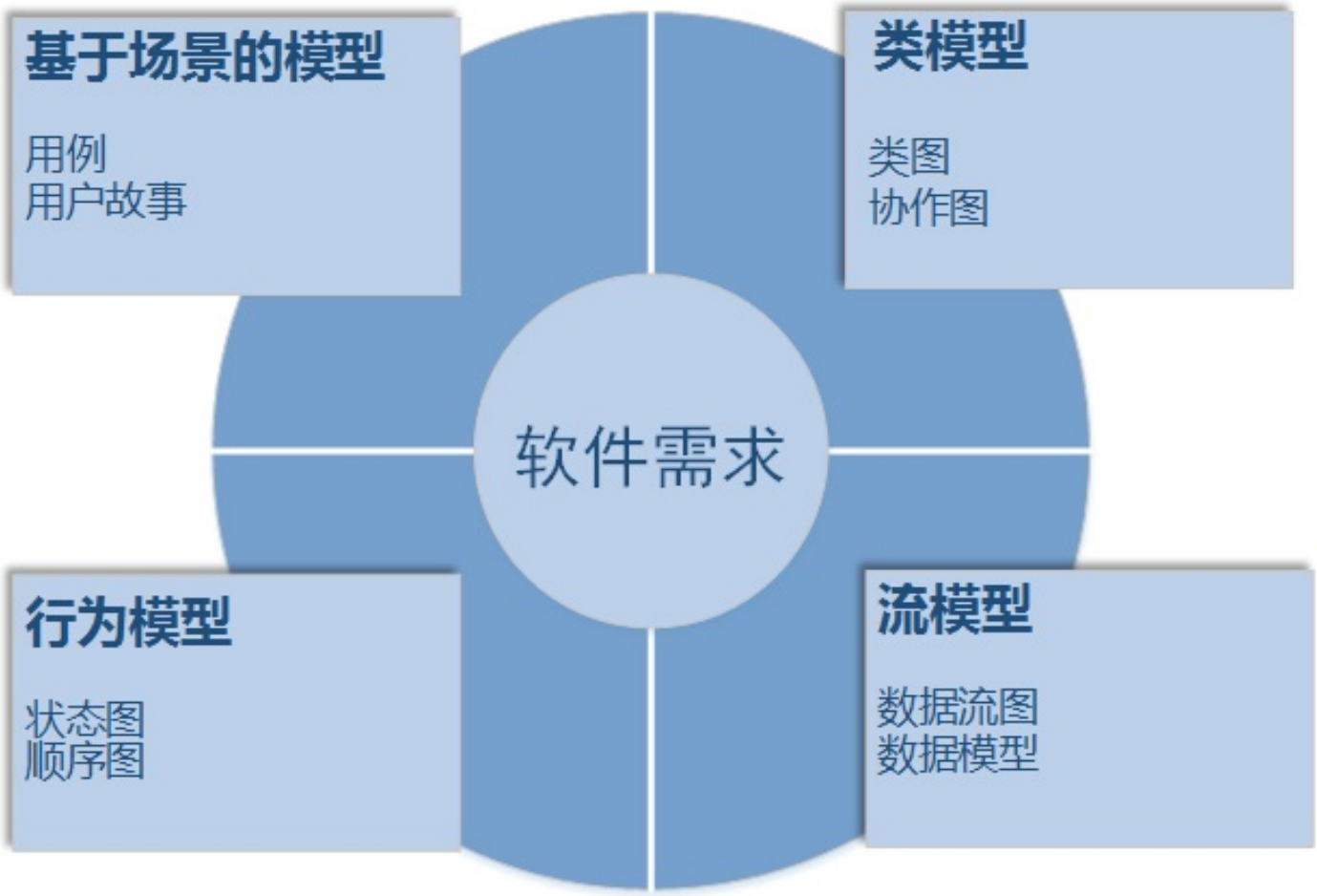
第四步：定义设计类图

确定类中的方法是什么？
确定对象如何和其他对象连接？

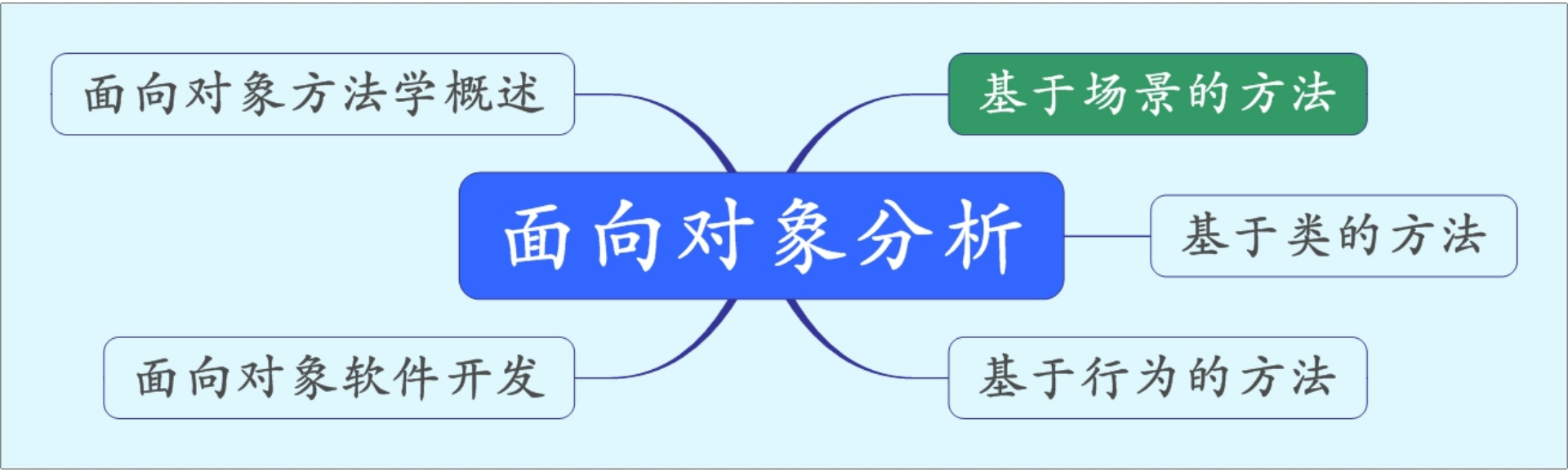
通过检查协作图来回答以上问题



回顾



outline



基于场景建模

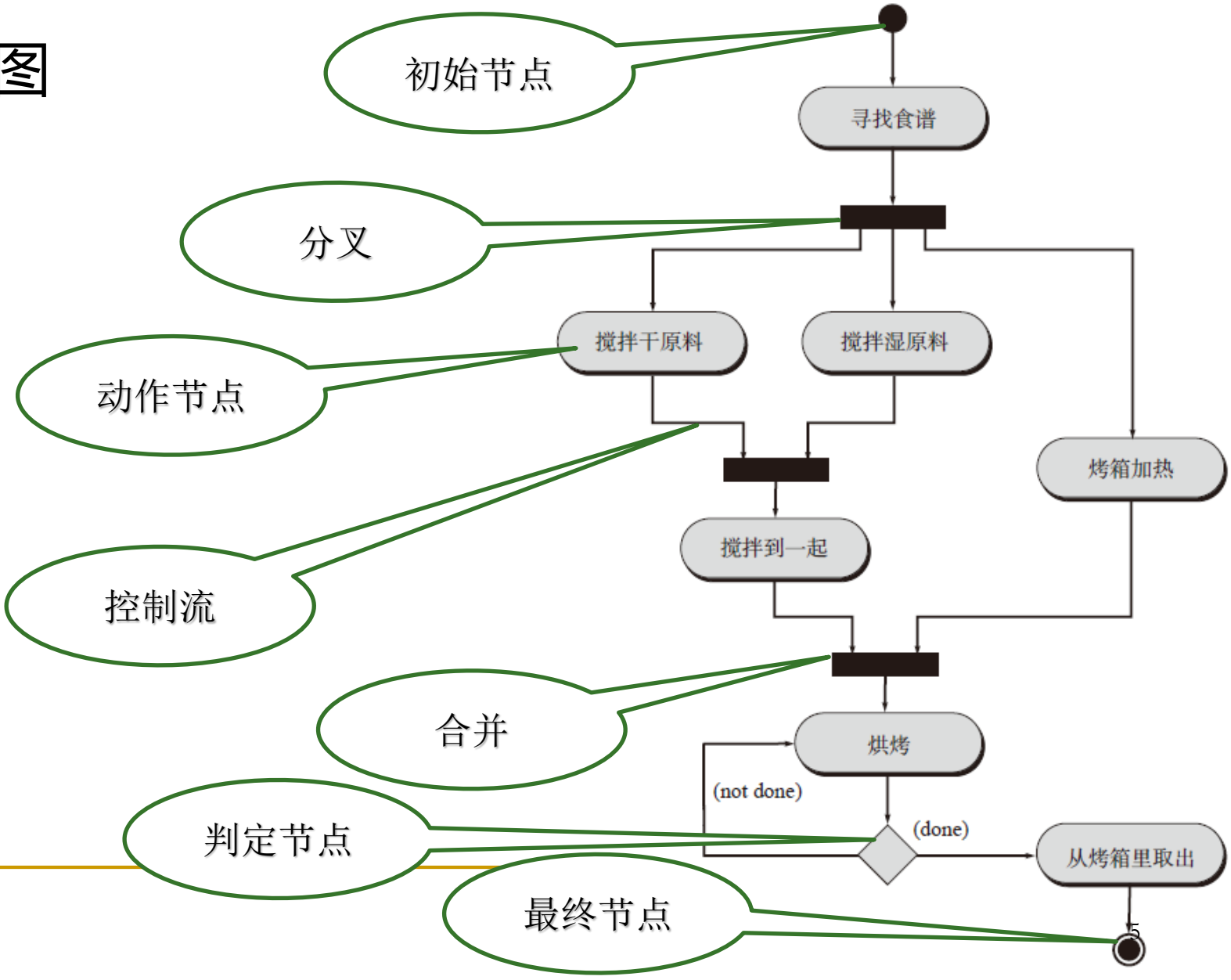
- 创建初始用例
  - 开始开发用例时，应列出特定参与者执行的功能或活动
  - 随着和利益相关者更深入地交谈，需求收集团队为每个标记的功能开发用例。
  - 用例：ACS-DCV
    - 描述性用例：P101
    - 用户活动的顺序序列：P102

基于场景建模

- 细化初始用例
  - 在这一步，参与者能做一些其它动作吗？
  - 在这一步，参与者有没有可能遇到一些错误的条件？如果有可能，这些错误会是什么？
  - 在这一步，参与者有没有可能遇到一些其它的行为（如由一些参与者控制之外的事件调用）？如果有，这些行为是什么？
- 编写正式用例
  - 用例模板：P103
  - 用例图：P105

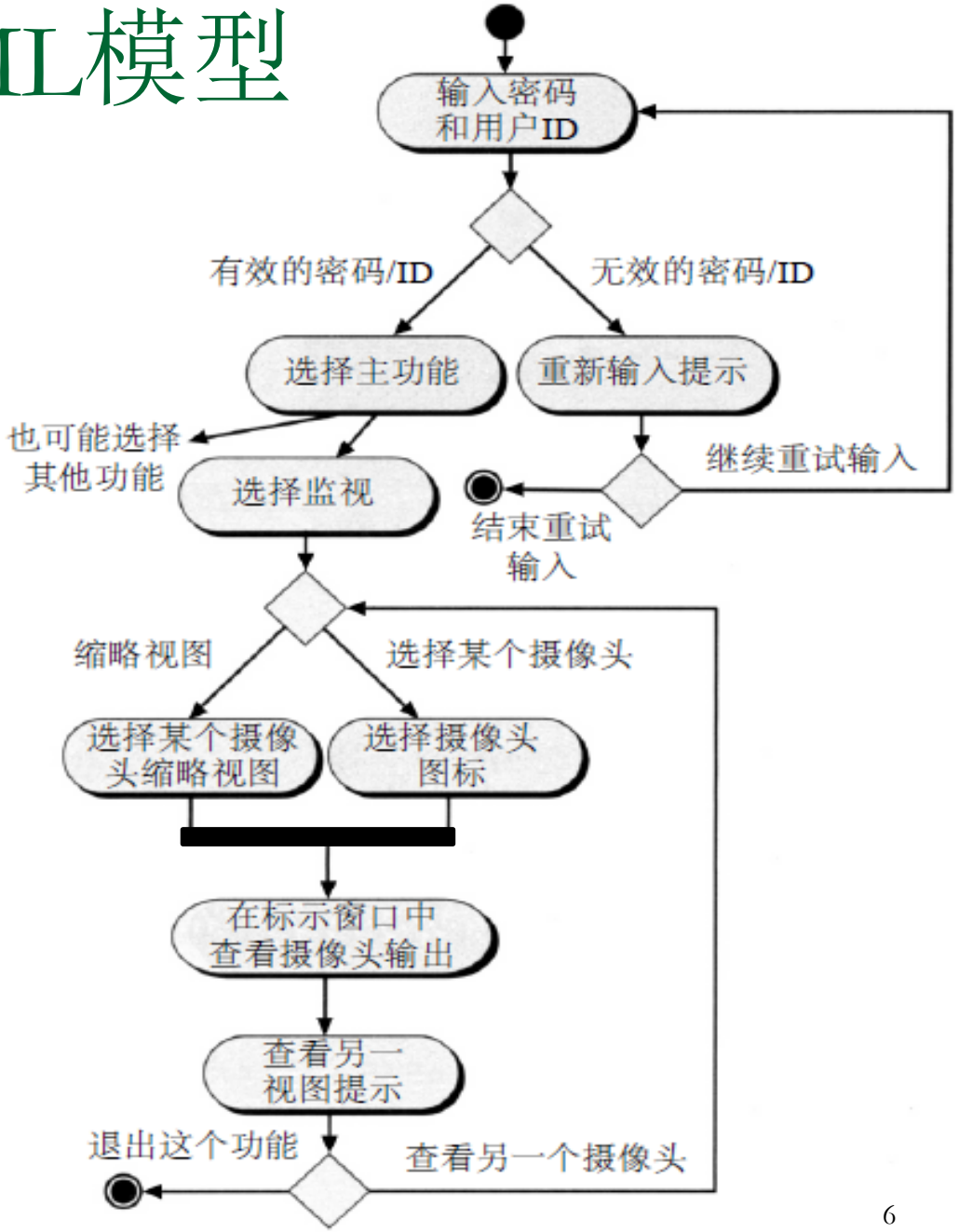
补充用例的UML模型

- 活动图



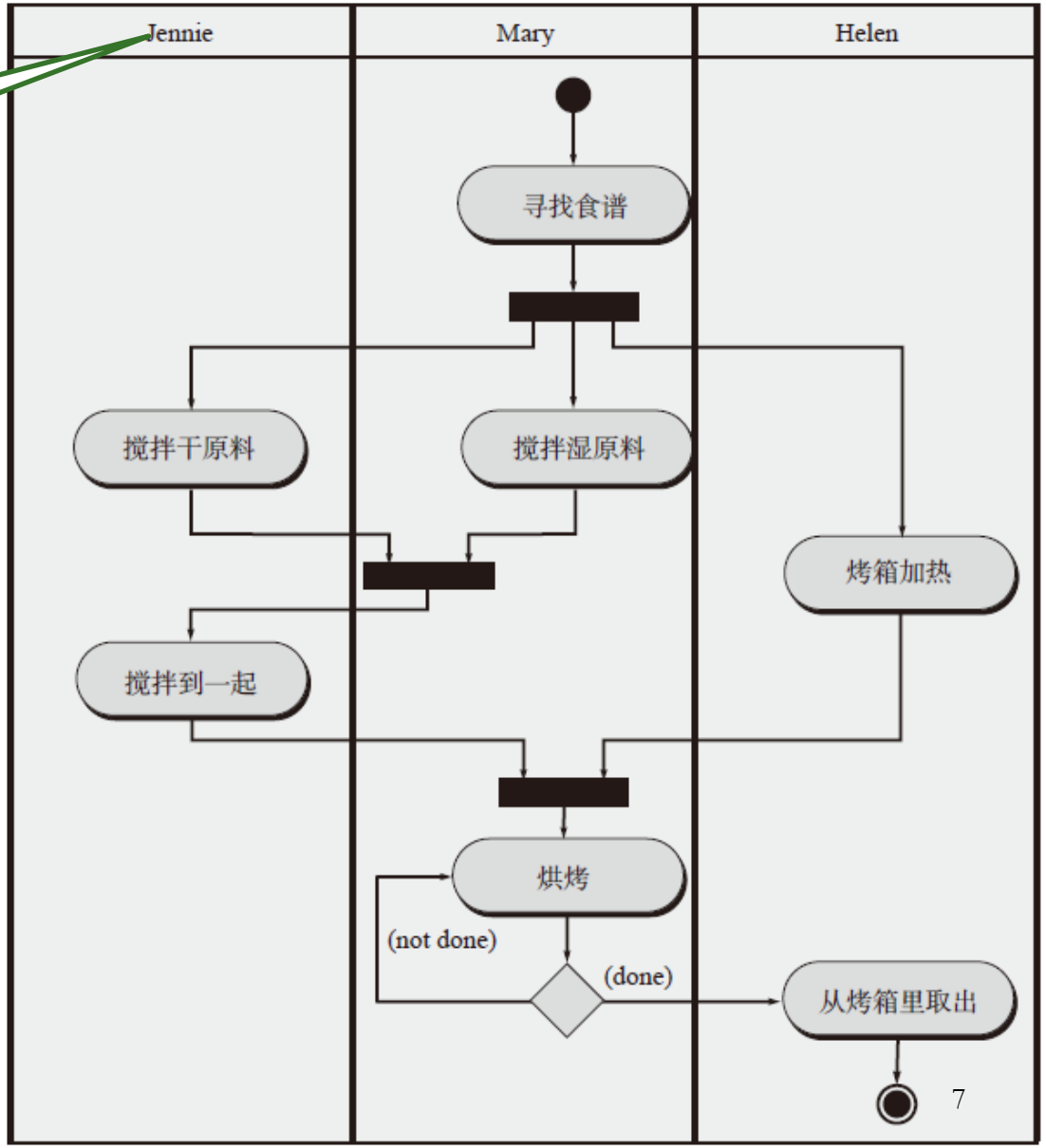
补充用例的UML模型

- ACS-DCV活动图



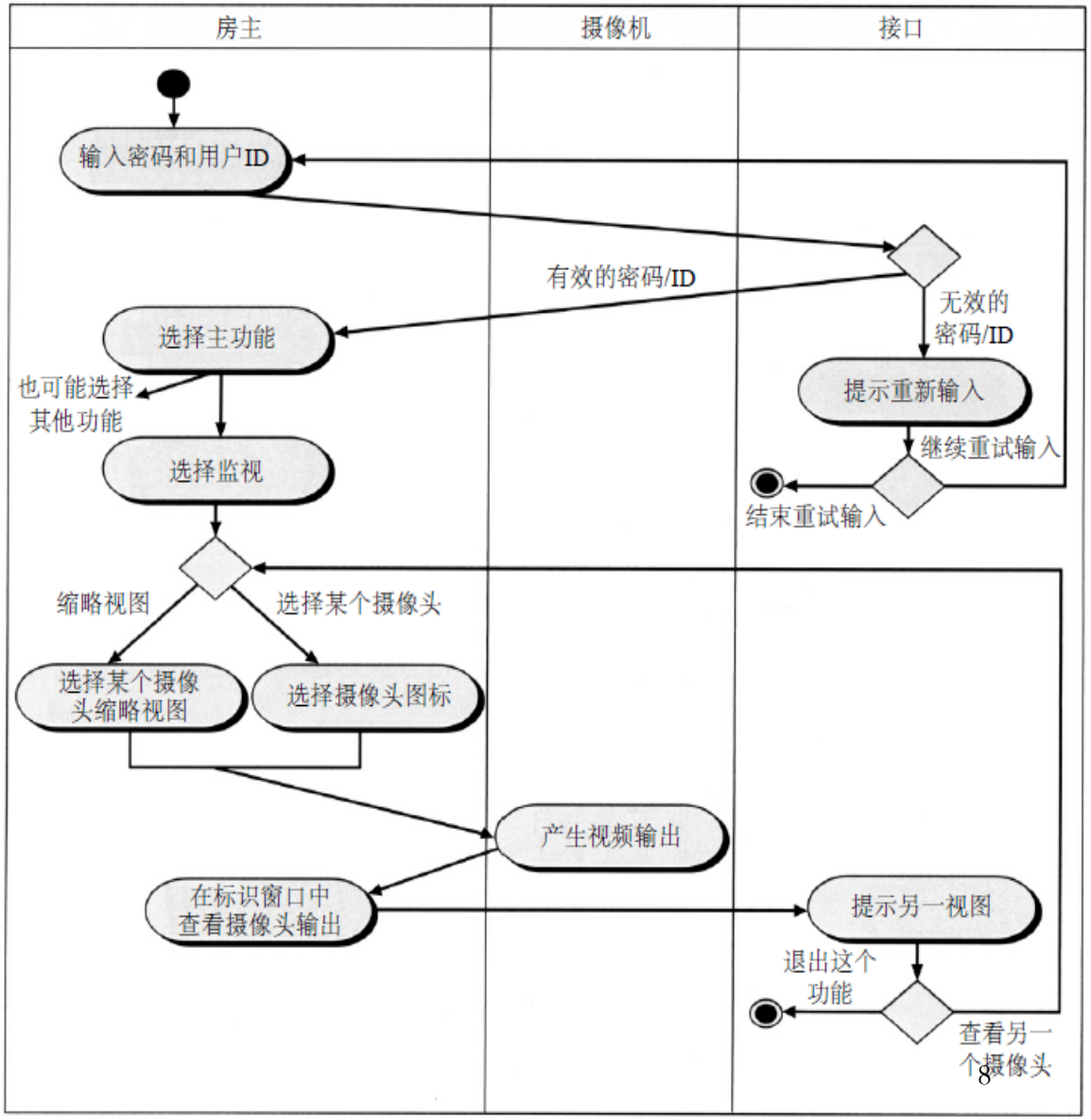
补充用例的UML模型

- 泳道（活动）图
  - 一个泳道内的所有动作都由该泳道对应的参与者完成

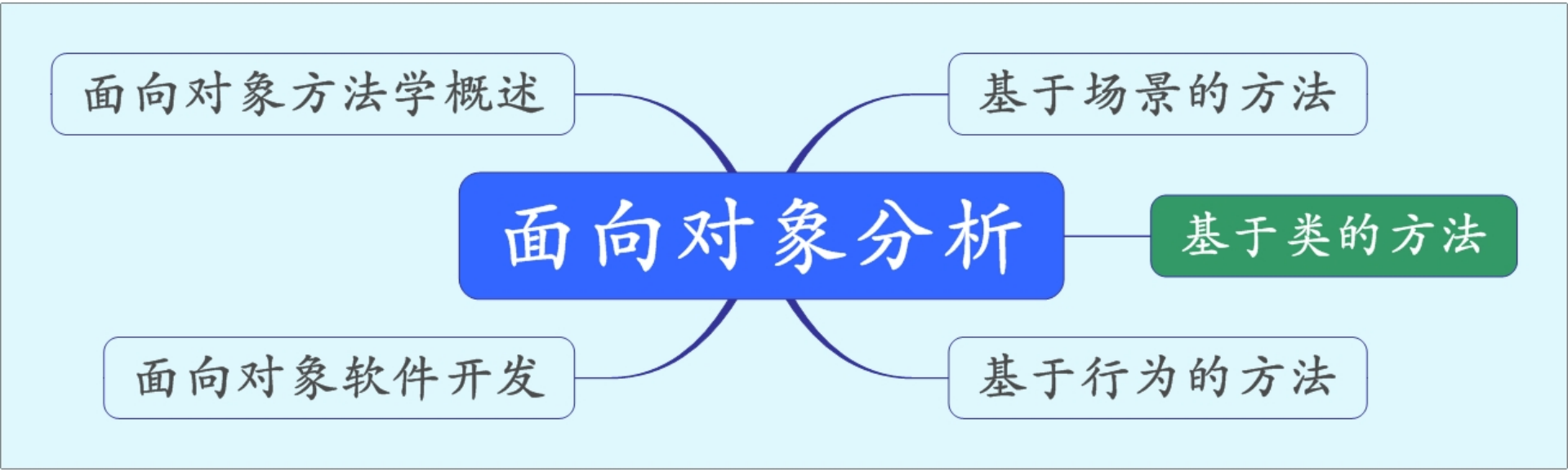


补充用例的UML模型

- ACS-DCV泳道图



outline



基于类建模

- 基于类建模表示了
  - 系统操作的对象
  - 应用于对象间能有效控制的操作（方法或服务）
  - 对象间的关系
  - 已定义类之间的协作
- 基于类的分析模型的元素
  - 类和对象
  - 属性
  - 操作
  - 类-职责-协作者模型(CRC)
  - 协作图
  - 包

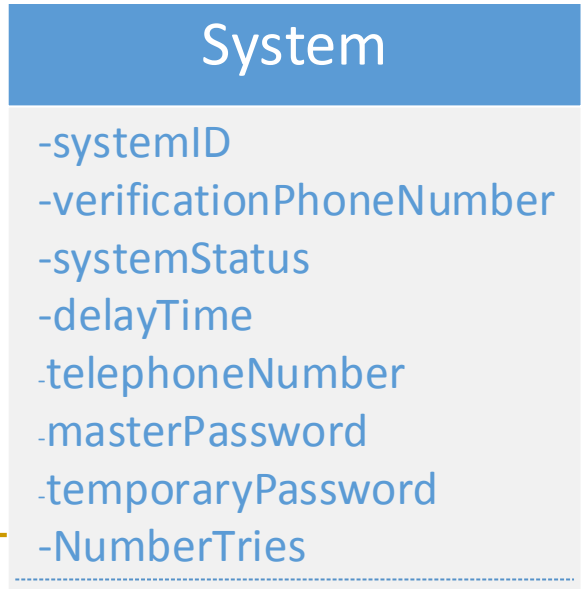
识别分析类

- 对需求模型中的场景进行“语法解析”（P109）
  - 带有下划线的每个名词或名词词组可以确定为类，并将这些名词输入到一个简单的表中。
  - 标注出同义词。
- 一旦分离出所有的名词，我们该寻找什么？
  - 外部实体（设备、人员）
  - 事物（报告、信号）
  - 事件（演出、访问）
  - 角色（工程师、教师）
  - 组织单元（部门、团队）
  - 场地（制造车间、码头）
  - 结构（传感器、计算机）

分析类
（P110）

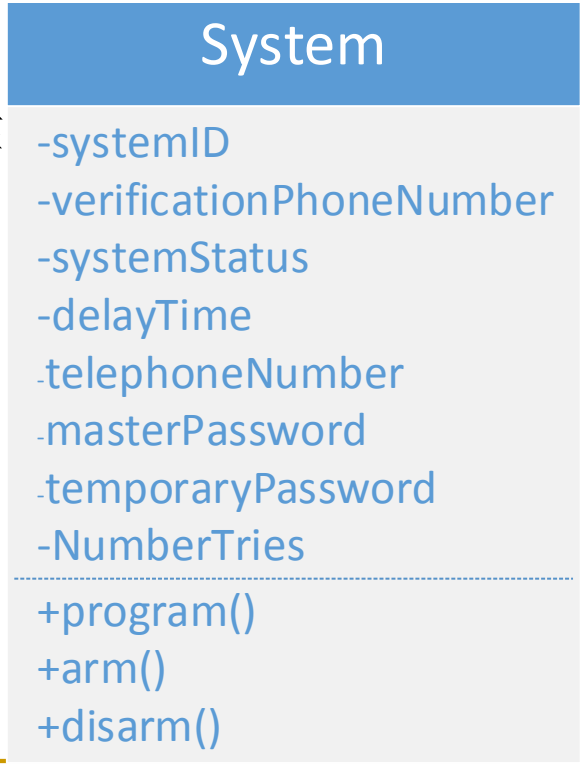
描述属性

- 对需求模型中的类进行描述
- 为职业棒球手建立两种不同的类
  - 数据统计系统：属性是与名字、位置、平均击球次数、担任防守百分比，从业年限、比赛次数等相关的
  - 养老系统：属性是平均工资、充分享受优惠权后的信用、所选的养老计划、邮件地址等相关的

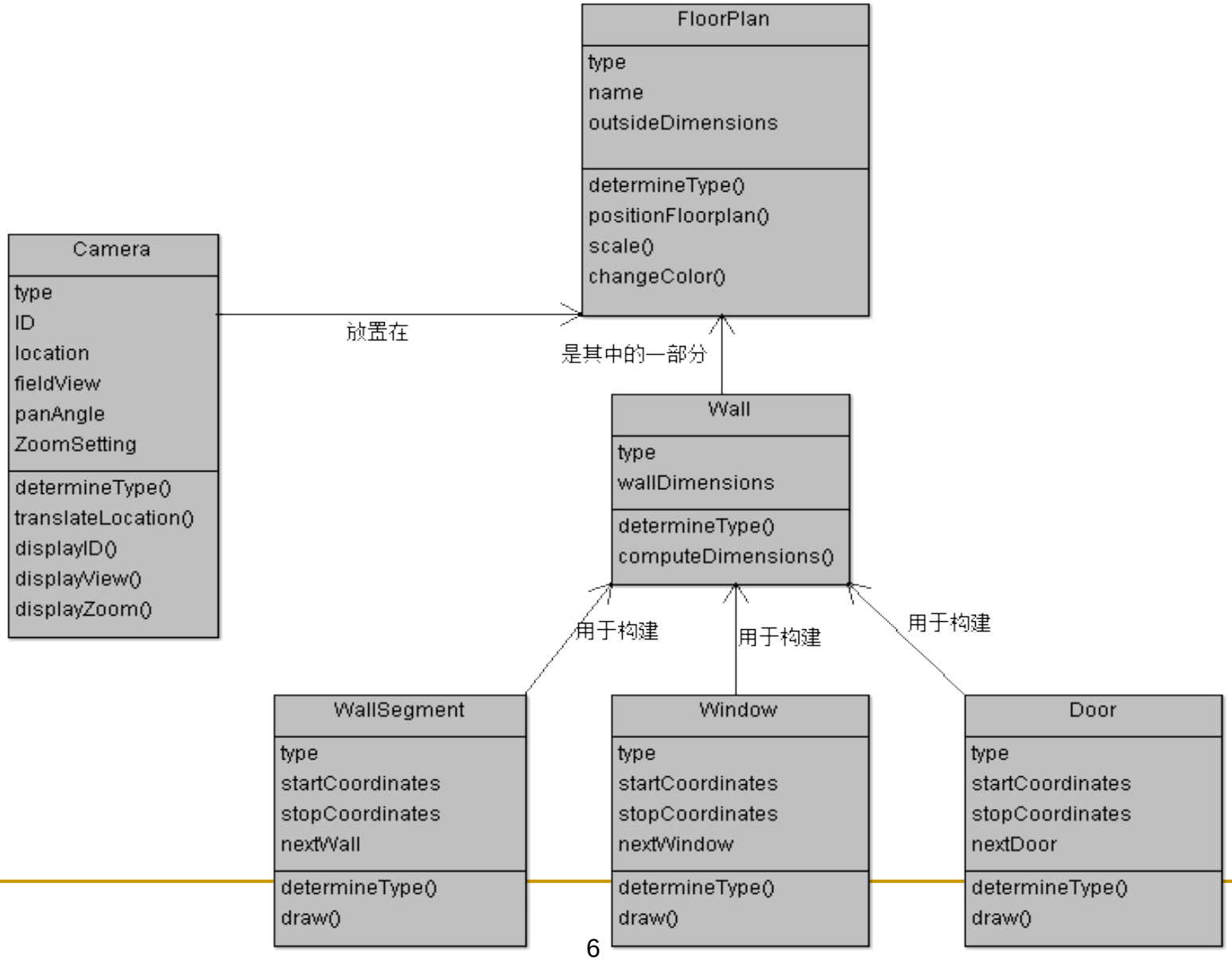


定义操作

- 操作定义了某个对象的行为
  - 以某种方式操作数据（添加、删除、选择）；
  - 执行计算的操作；
  - 请求某个对象的状态的操作；
  - 监视某个对象发生某个控制事件的操作
- 语法解析：动词
  - “*配置*系统”
  - “主密码用于*激活*和*解除*系统”



ACS-DCV平面设计图类



CRC模型

- 类-职责-协作者（Class-Responsibility-Collaborator，CRC）
  - 识别和组织与系统或产品需求相关的类
- 表示类的标准索引卡片的集合
  - 顶部：类名
  - 左侧：类的职责
  - 右侧：类的协作者

平面设计图CRC

Class: FloorPlan	
说明	
职责：	协作者：
定义住宅平面图的名称/类型	
管理住宅平面图的布局	
缩放显示住宅平面图	
合并墙、门和窗	Wall
显示摄像头的位置	Camera

类的分类

- **实体类**（模型或业务类），从问题说明中直接提取（FloorPlan和Sensor）
- **边界类**用于创建用户可见的和在使用软件时交互的接口（交互屏幕或打印的报表）
- **控制类**自始至终管理“工作单元”
  - 实体类的创建或更新
  - 边界类从实体对象获取信息后的实例化
  - 对象集合间的复杂通信
  - 对象间或用户和应用系统间交换数据的确认

职责

- 职责应分布在所有类中以求最大程度地满足问题的需求。
- 每个职责的说明应尽可能具有普遍性。
- 信息和与之相关的行为应放在同一个类中。
- 某个事物的信息应局限于一个类中而不要分布在多个类中。
- 适合时，职责应由相关类共享。

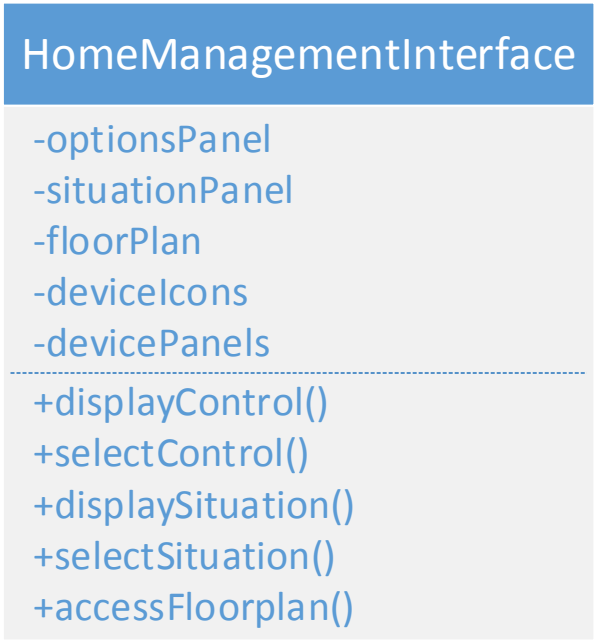
协作

- 实现职责的方法
  - 类可以使用其自身的操作控制各自的属性，从而实现特定的职责；
  - 一个类可以和其他类协作。
- 要识别协作可以通过确认类本身是否能够实现自身的每个职责
  - ControlPanel对象的职责之一：确定是否启动所有的传感器
  - determine-sensor-status()
  - 传感器信息要从每个Sensor对象获取

评审CRC模型

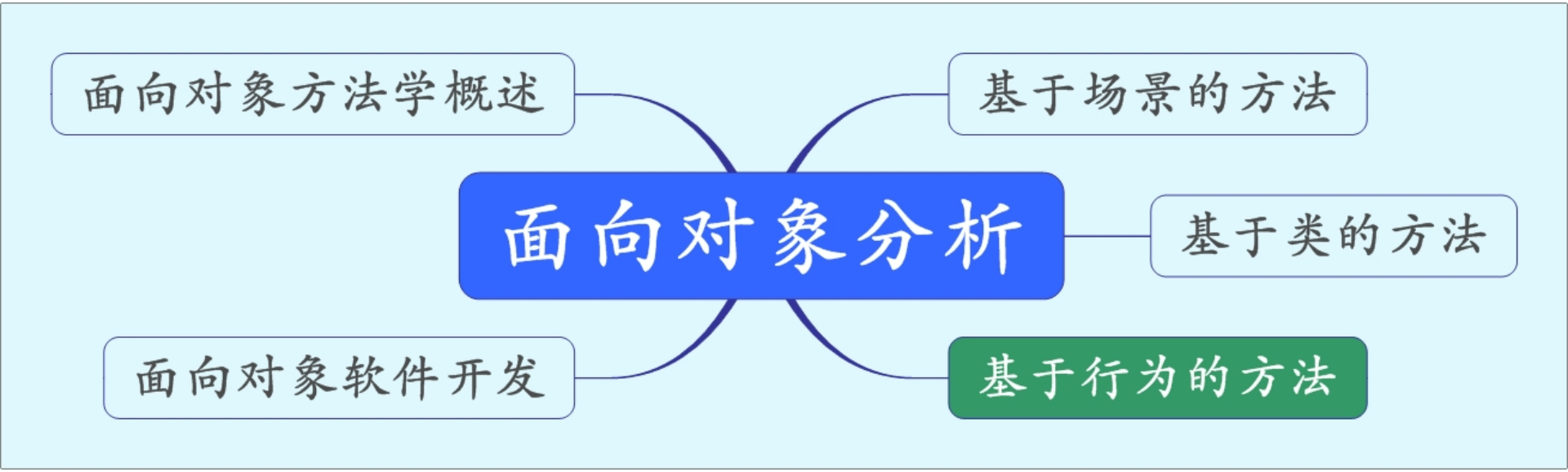
- 所有参加评审的人员拿到一部分CRC模型索引卡
- 分类管理所有的用例场景（以及相关的用例图）
- 评审组长细致地阅读用例
  - 当评审组长看到一个已命名的对象时，给拥有相应类索引卡的人员一个令牌
- 当令牌传递时，该类卡的拥有者需要描述卡上记录的职责
  - 评审组确定（一个或多个）职责是否满足用例需求
- 如果记录在索引卡上的职责和协作不能满足用例，就需要修改卡片
  - 修改可能包括定义新类（和相关的CRC索引卡），或者在已有的卡上说明新的或修改的职责、协作

SafeHome住宅管理功能



类：HomeManagementInterface	
职责：	协作者：
dispalyControl() selectControl() dispalySituation() selectSituation() accessFloorplan()	OptionPanel类 OptionPanel类 SituationPanel类 SituationPanel类 FloorPlan类

outline



生成行为模型

- 行为模型显示了软件如何对外部事件做出响应
- 生成模型的步骤：
  - 评估所有的用例，以保证完全理解系统内的交互顺序
  - 识别驱动交互顺序的事件，并理解这些事件如何与特定的对象相互关联
  - 为每个用例生成序列
  - 创建系统状态图
  - 评审行为模型以验证准确性和一致性