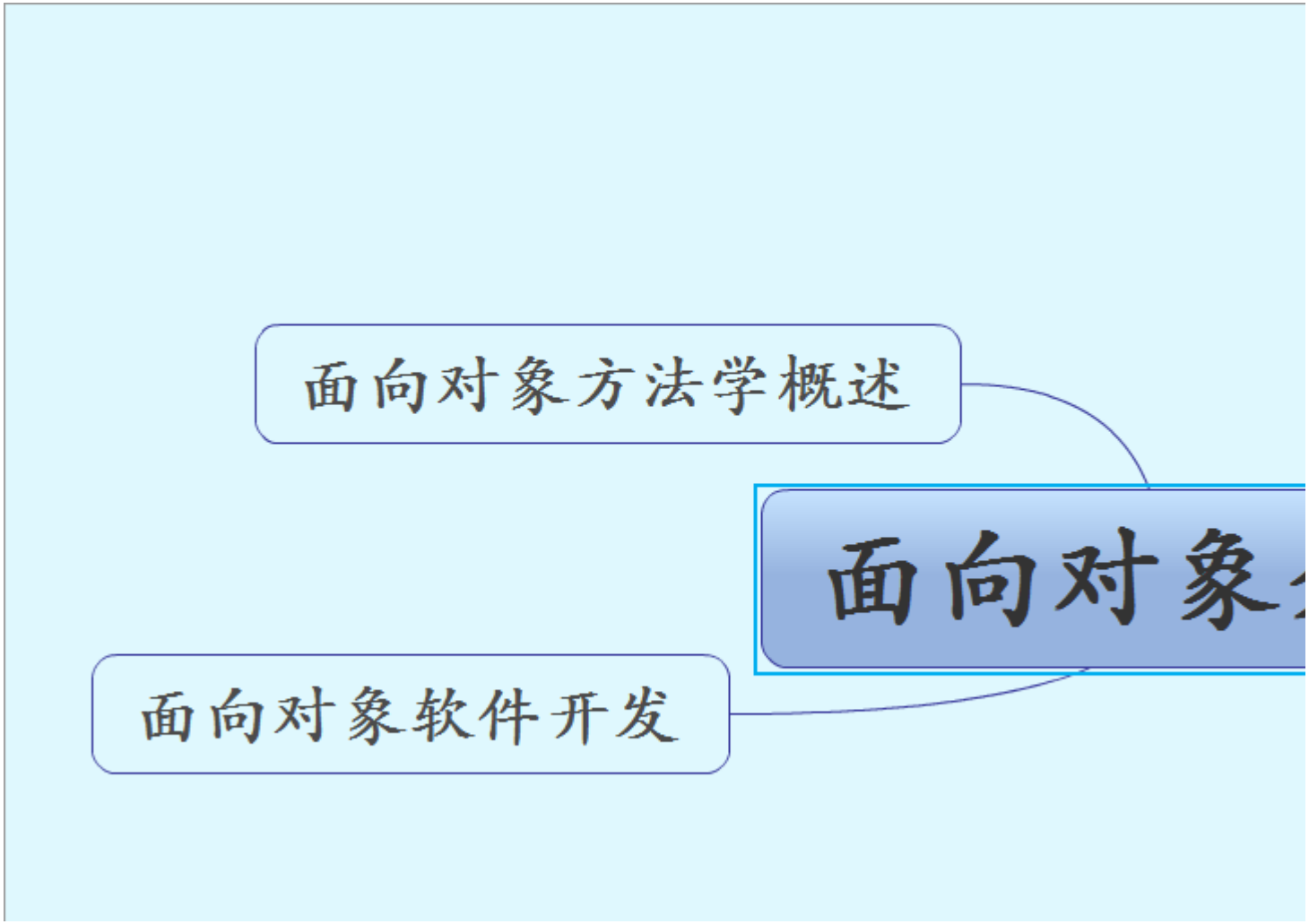
面向对象分析
(Object-Oriented Analysis, OOA)

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，而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，许多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等）, 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

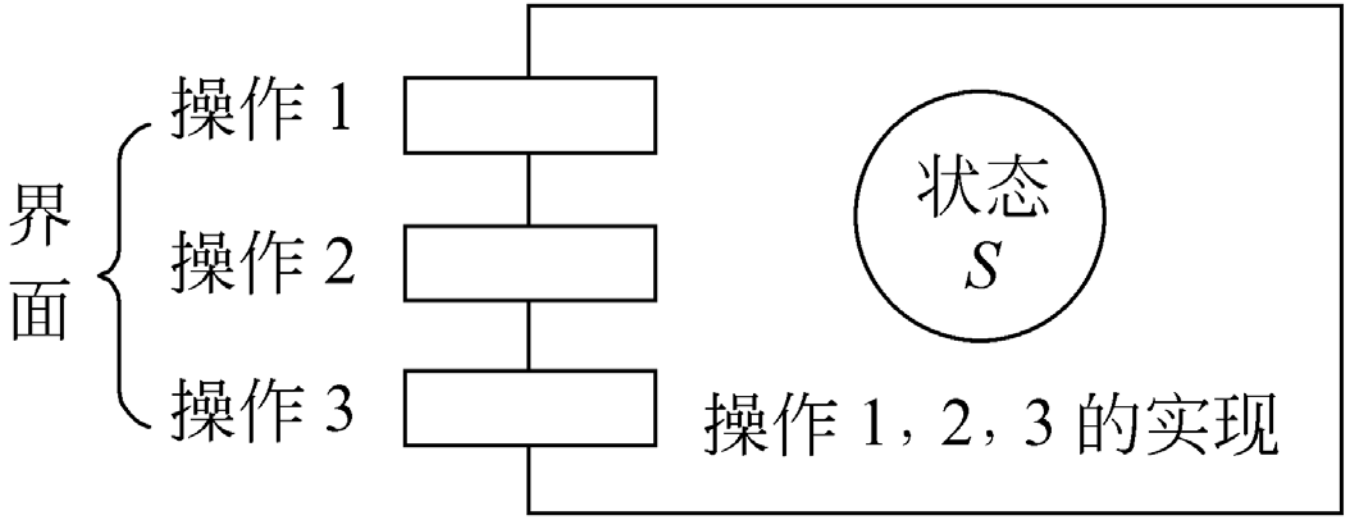
    是面向对象的技术中发展最快的，使用面向对象的程序设计语言，进行编码。

面向对象的基本概念

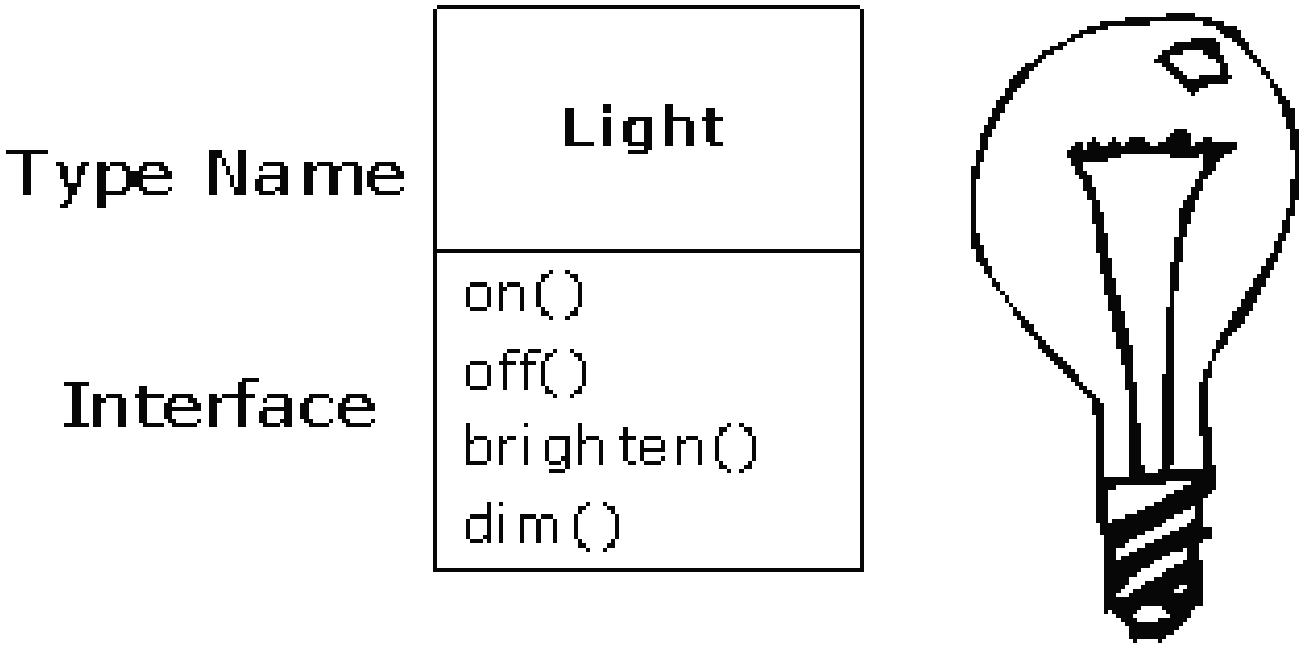
- **对象(object)**
- **类(class)**
- **消息(message)**
- **封装(encapsulation)**
- **继承(inheritance)**
- **多态(polymorphism)**

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。

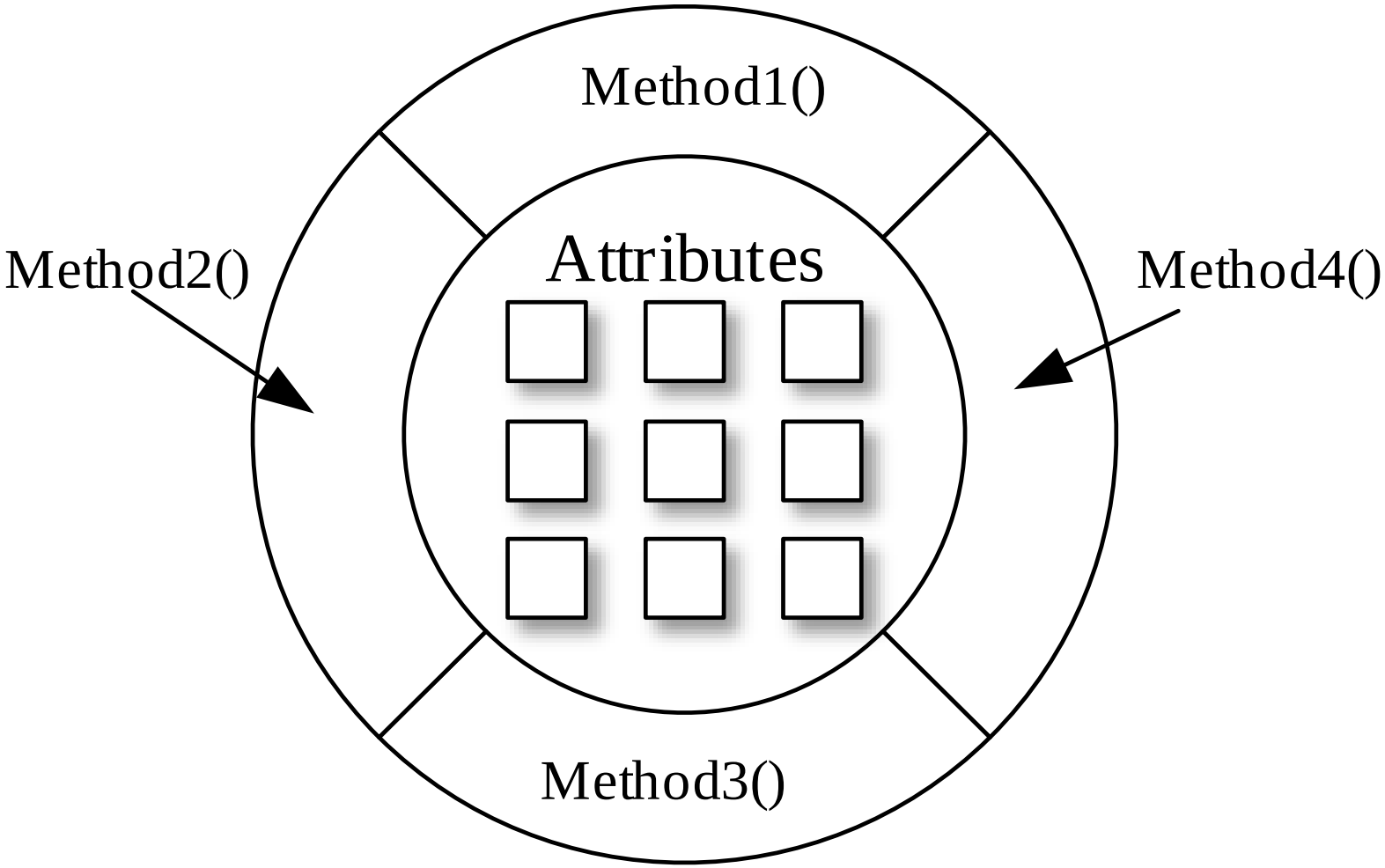
面向对象基本概念3—消息

- 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) ;
           ……
      }
```

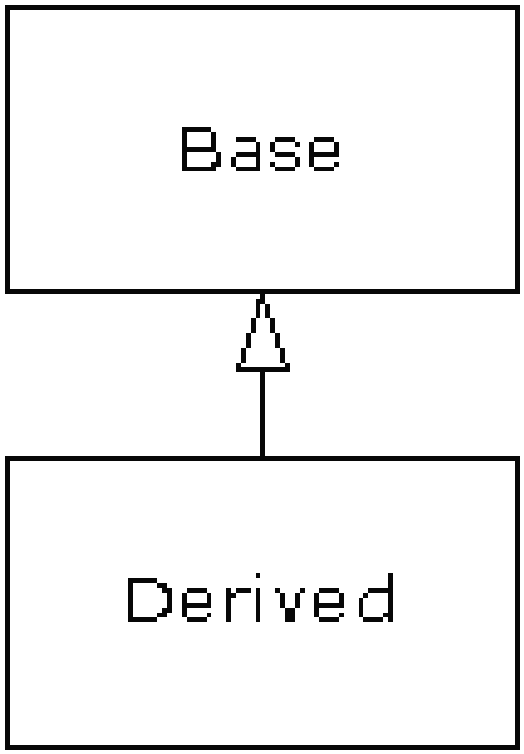
面向对象基本概念4—封装



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。

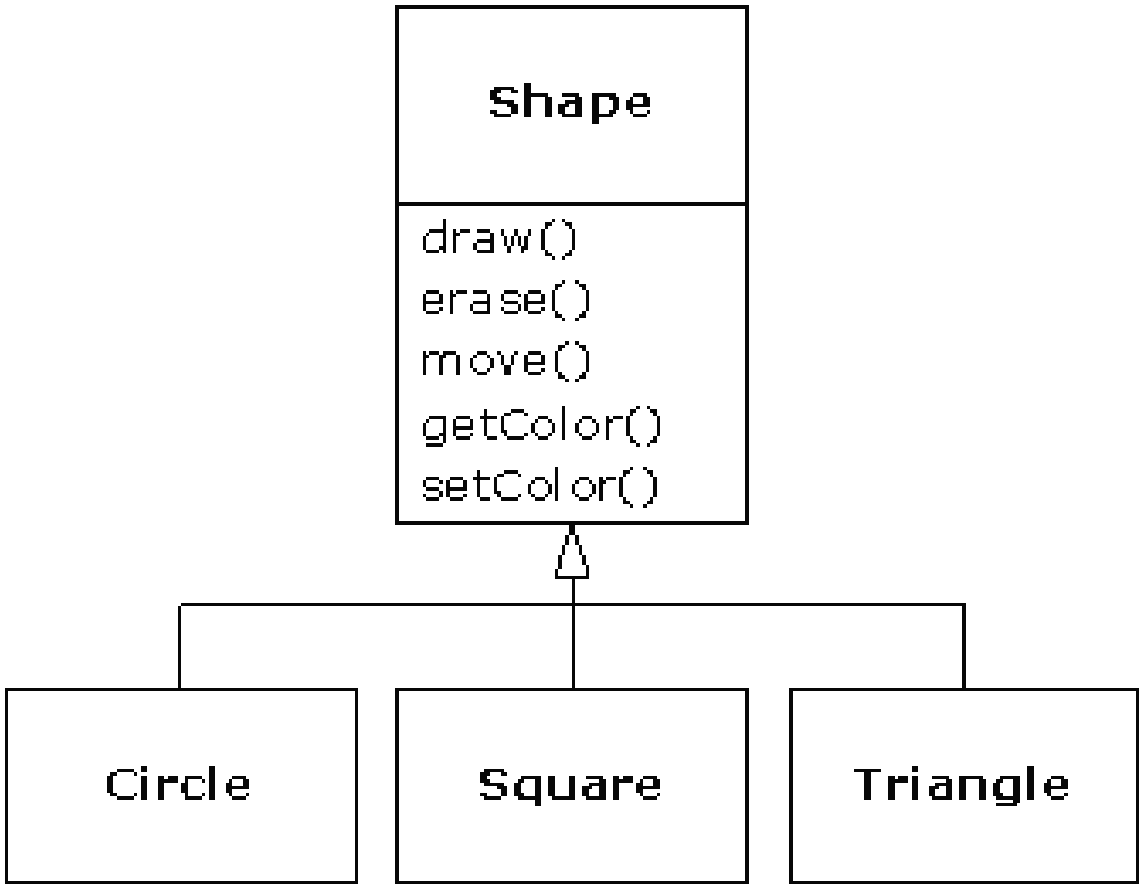
面向对象基本概念5—继承

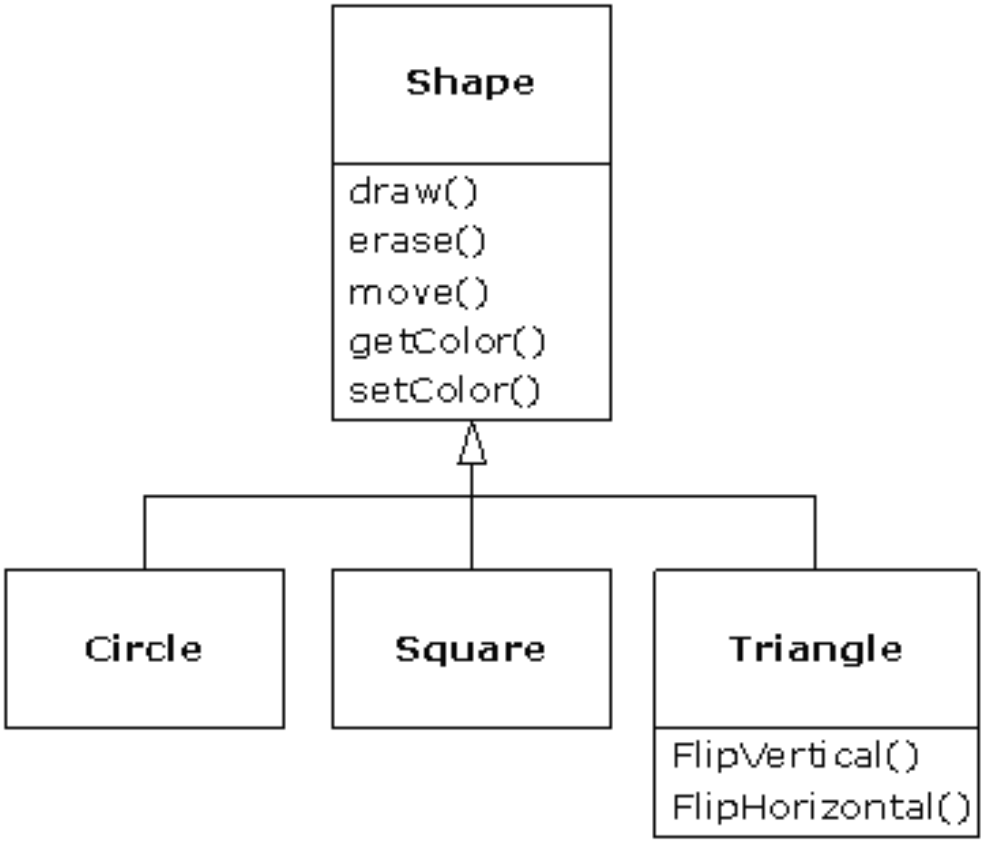


Inheritance

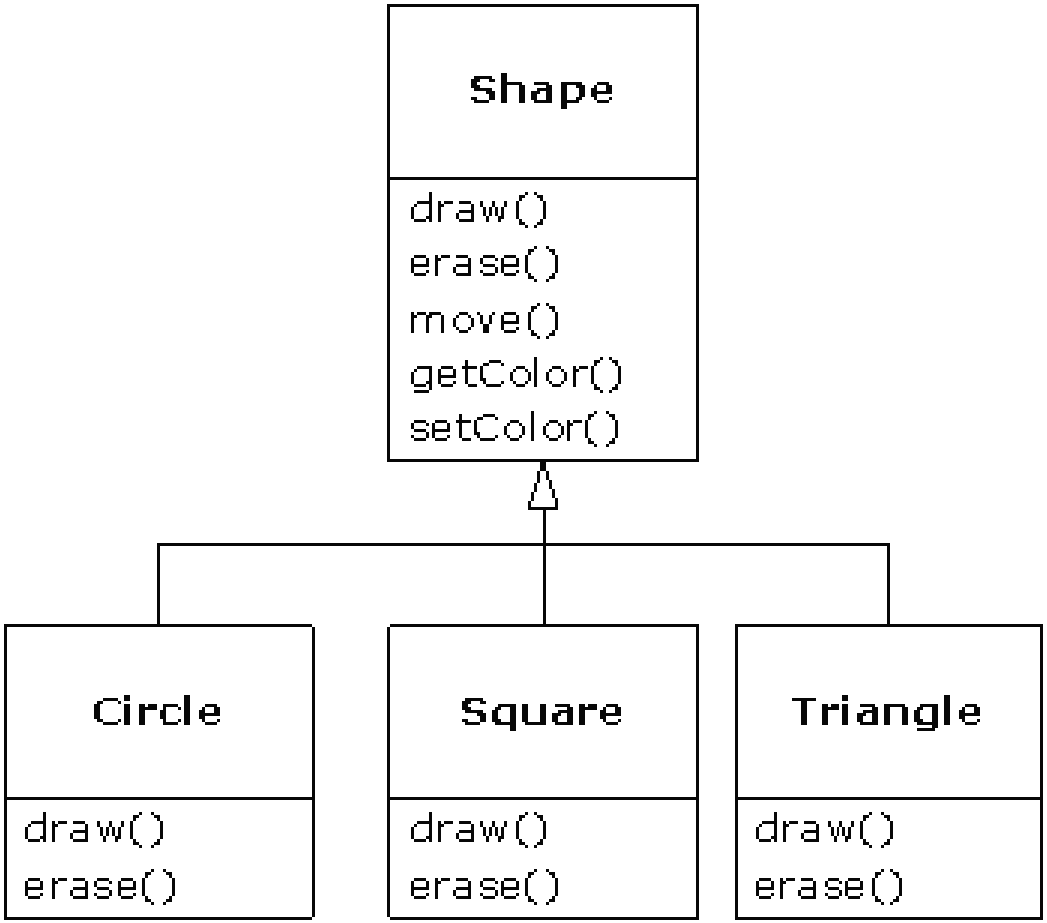
面向对象基本概念5—继承

- 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

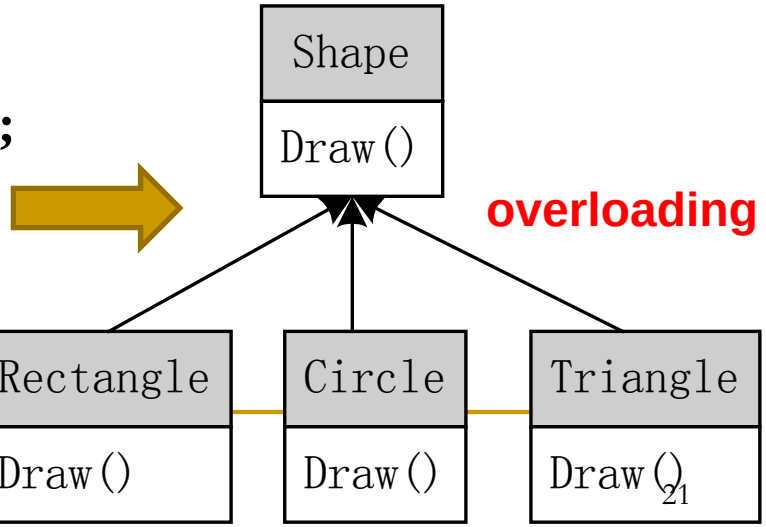


overriding methods

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;
```

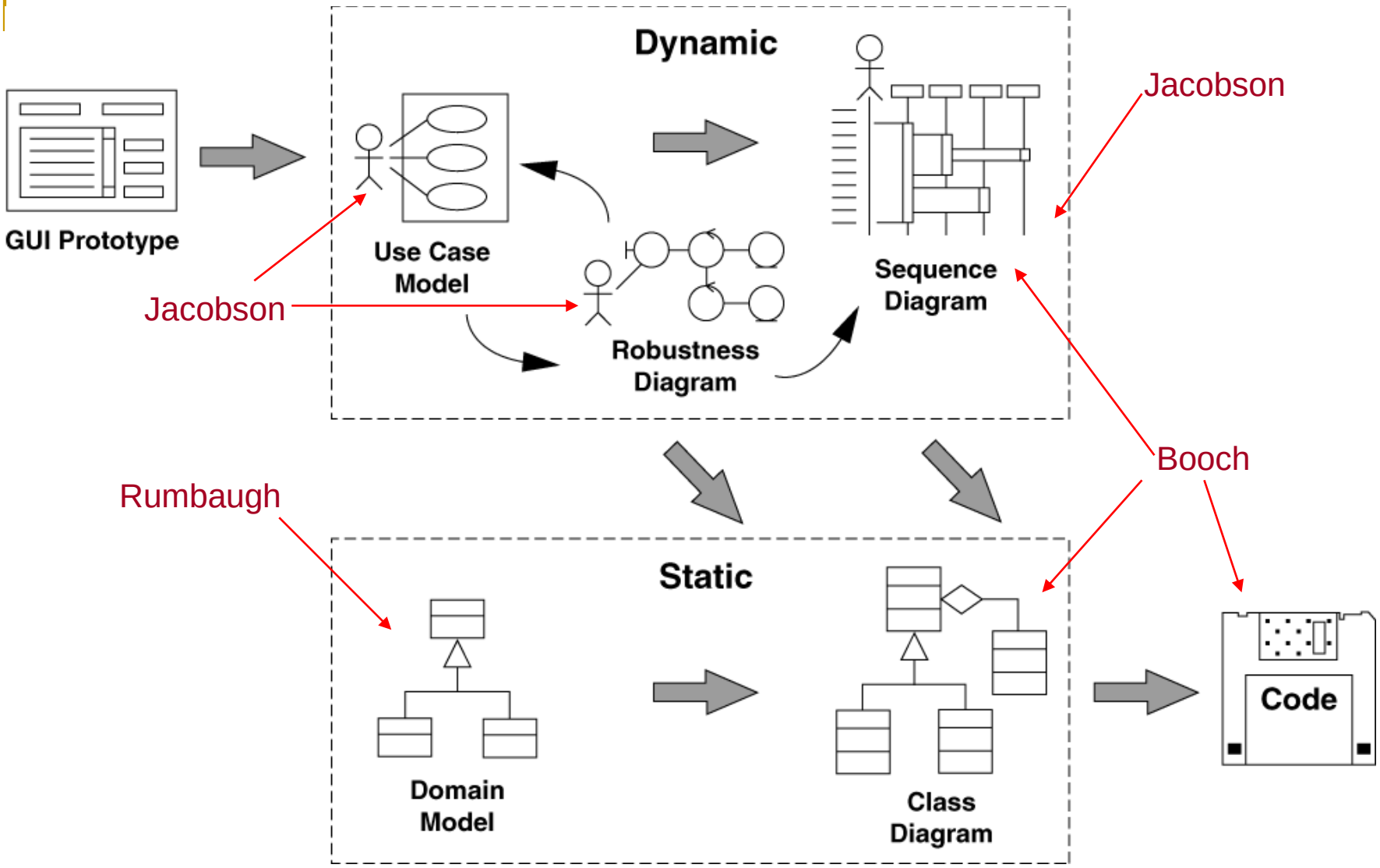


面向对象分析建模

- 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

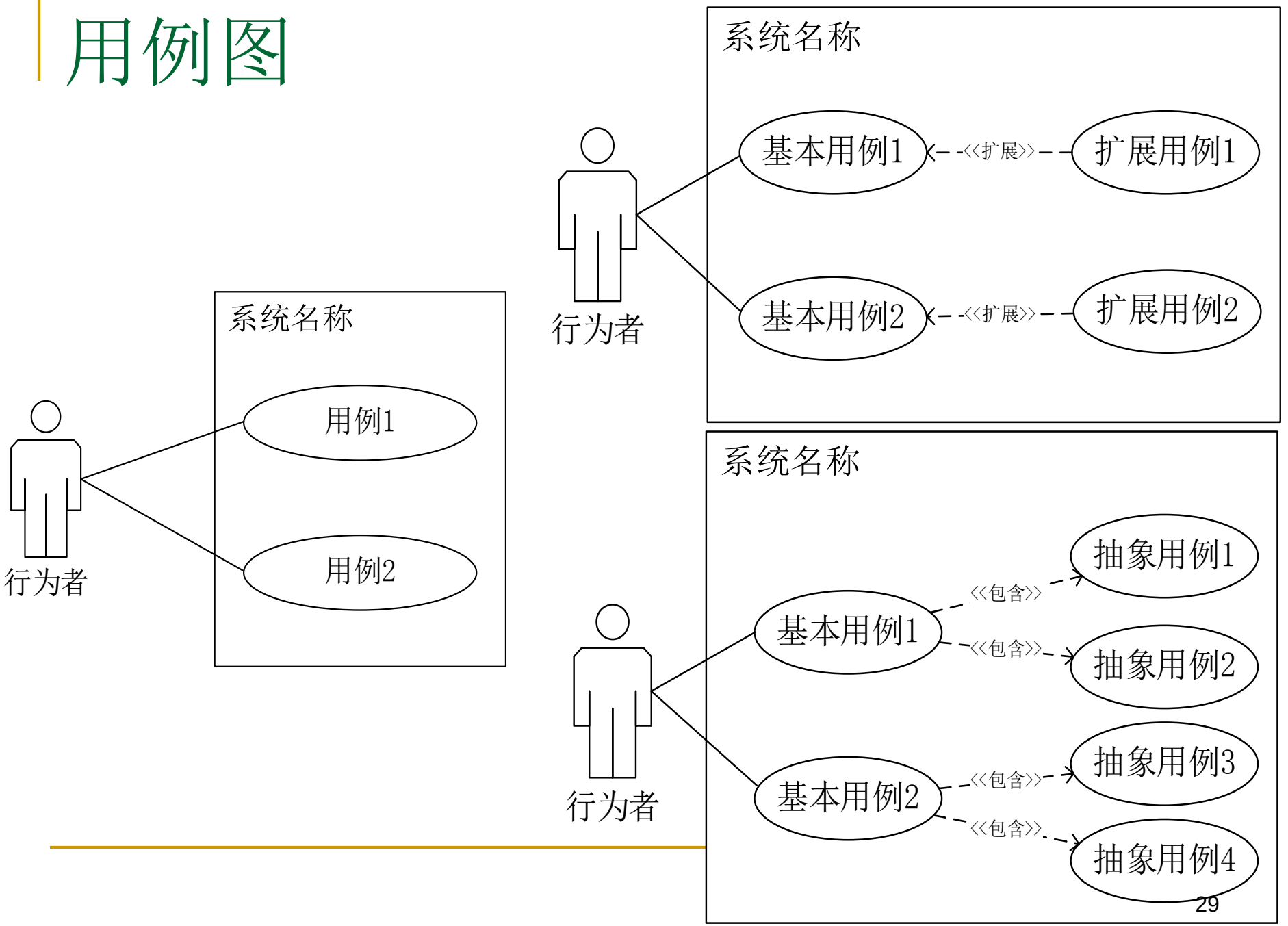
用例图

- UML提供的**用例图**是进行需求分析和建立功能模型的强有力工具。
  - 用例模型描述的是外部行为者（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

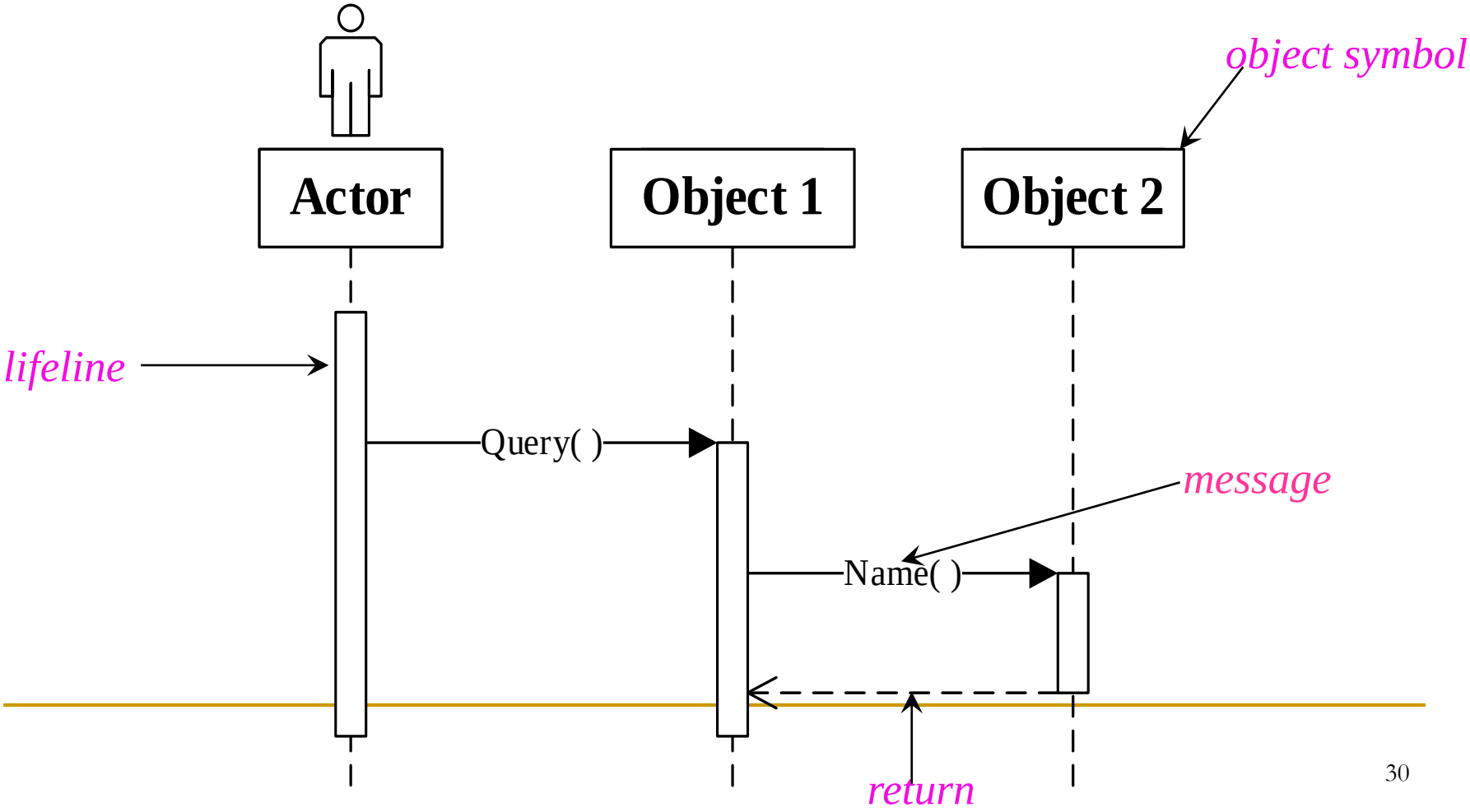
- 一幅用例图包含的模型元素有系统、行为者、用例及用例之间的关系。
  - 系统被看作是一个提供用例的黑盒子。
  - 行为者是指与系统交互的人或其他系统，它代表外部实体；行为者代表一种角色。
  - 一个用例是可以被行为者感受到的、系统的一个完整的功能。
  - UML用例之间主要有扩展和使用两种关系。
    - 扩展：向一个用例中添加一些动作后构成了另一个用例
    - 使用：一个用例使用另一个用例

用例图

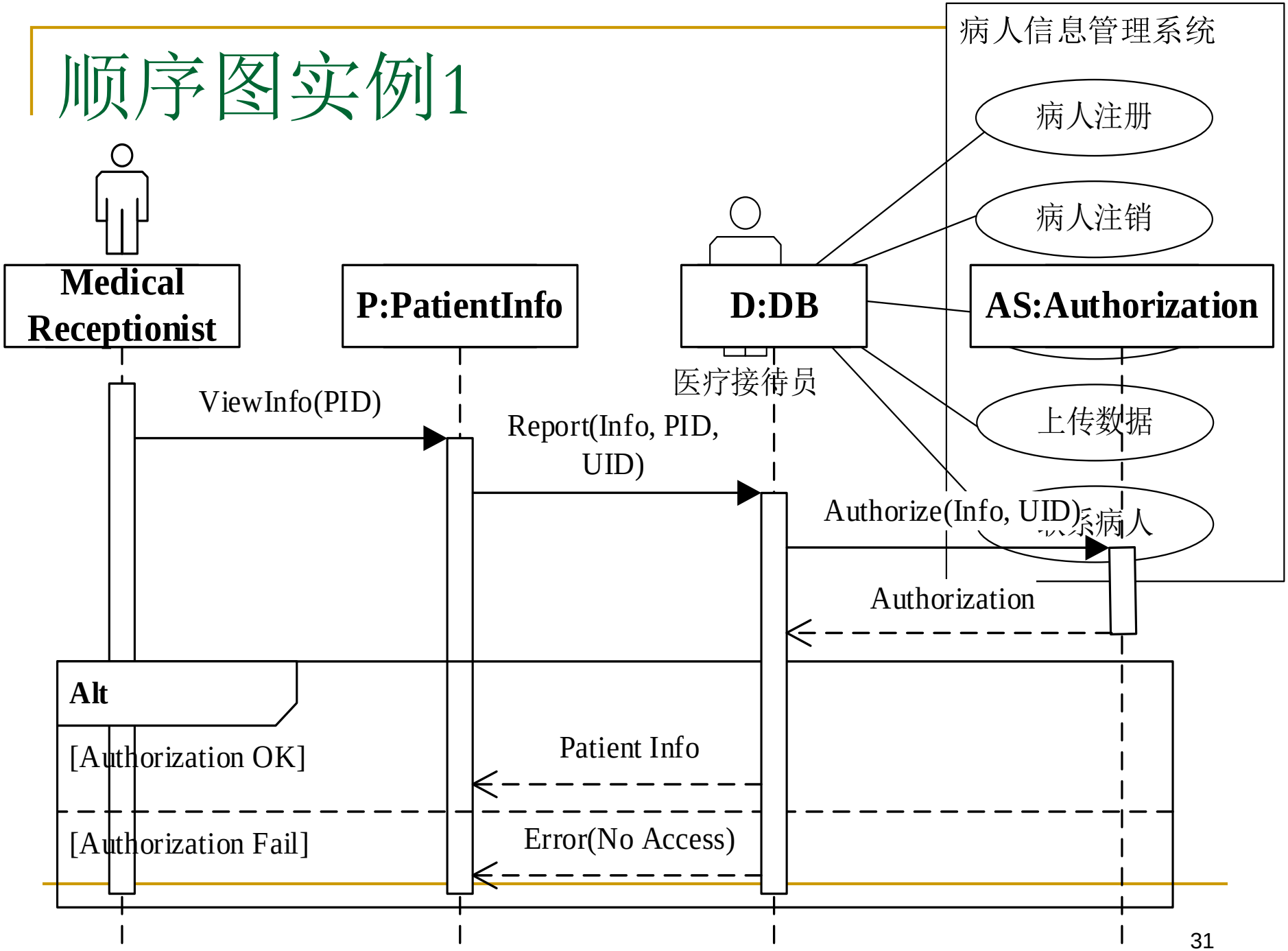

系统名称
基本用例1
<<扩展>>
扩展用例1
基本用例2
<<扩展>>
扩展用例2
行为者
系统名称
用例1
用例2
行为者
系统名称
抽象用例1
<<包含>>
基本用例1
<<包含>>
抽象用例2
抽象用例3
<<包含>>
基本用例2
<<包含>>
抽象用例4
行为者

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



顺序图实例1


病人信息管理系统
病人注册
病人注销
上传数据
联系病人
Medical Receptionist
P:PatientInfo
D:DB
医疗接待员
AS:Authorization
ViewInfo(PID)
Report(Info, PID, UID)
Authorize(Info, UID)
Authorization
Alt
[Authorization OK]
Patient Info
[Authorization Fail]
Error(No Access)

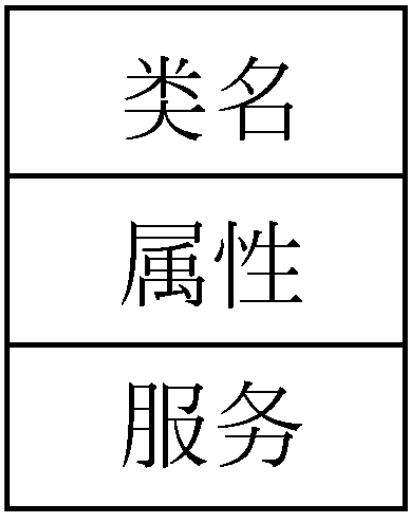
顺序图实例2

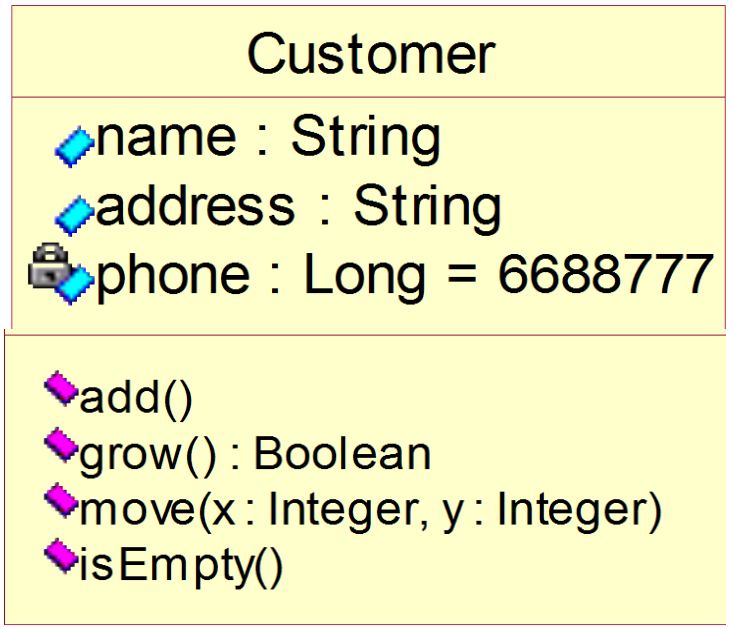
【基幹系】詳細設計書（売掛金）照会
-Web（売上内容照会）

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：

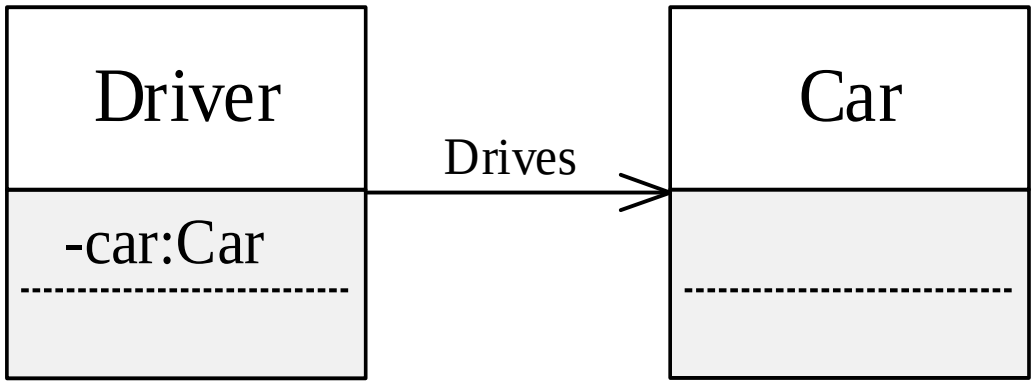
  参数名：类型名=默认值

类之间的关系

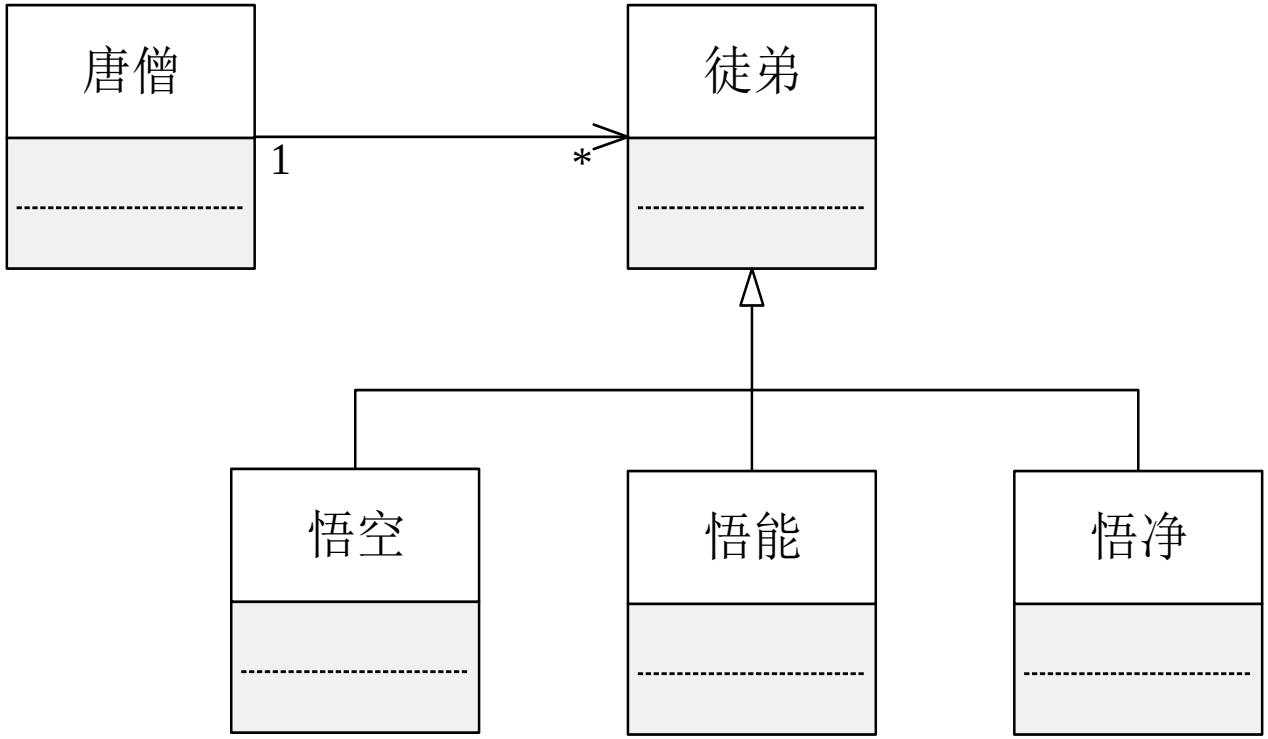
- 关联（**Association**）
  - 聚合（**Aggregation**）
  - 合成（**Composition**）
- 泛化（**Generalization**）
- 依赖（**Dependency**）

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



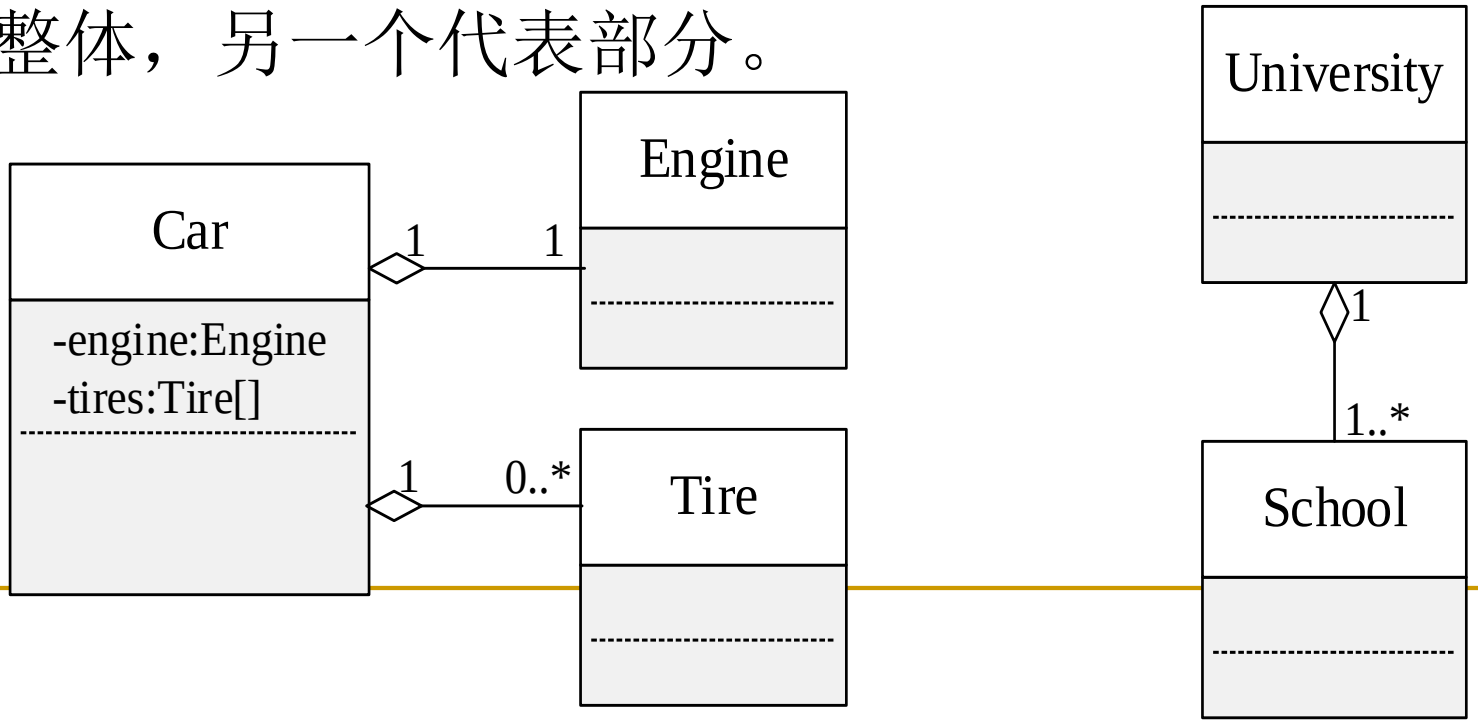
关联关系



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

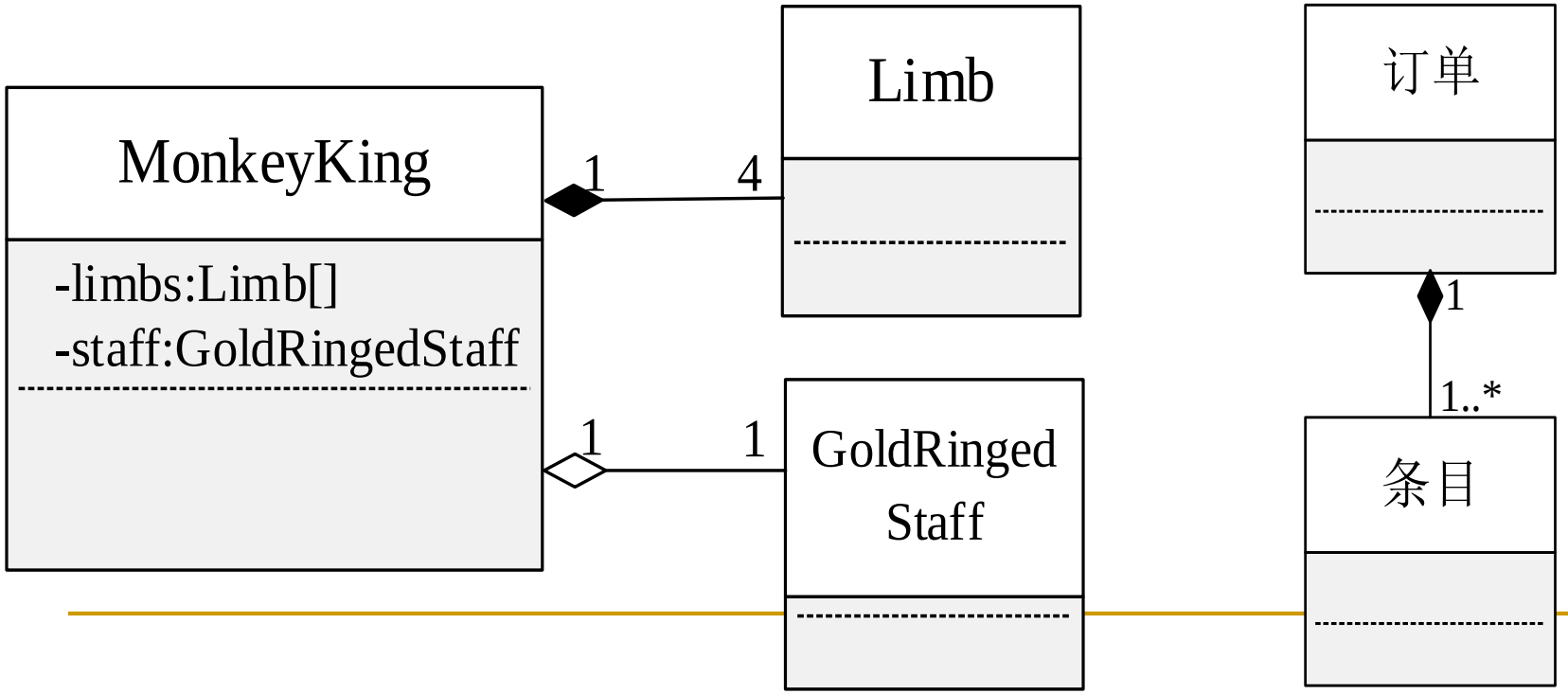
聚合关系

- 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。



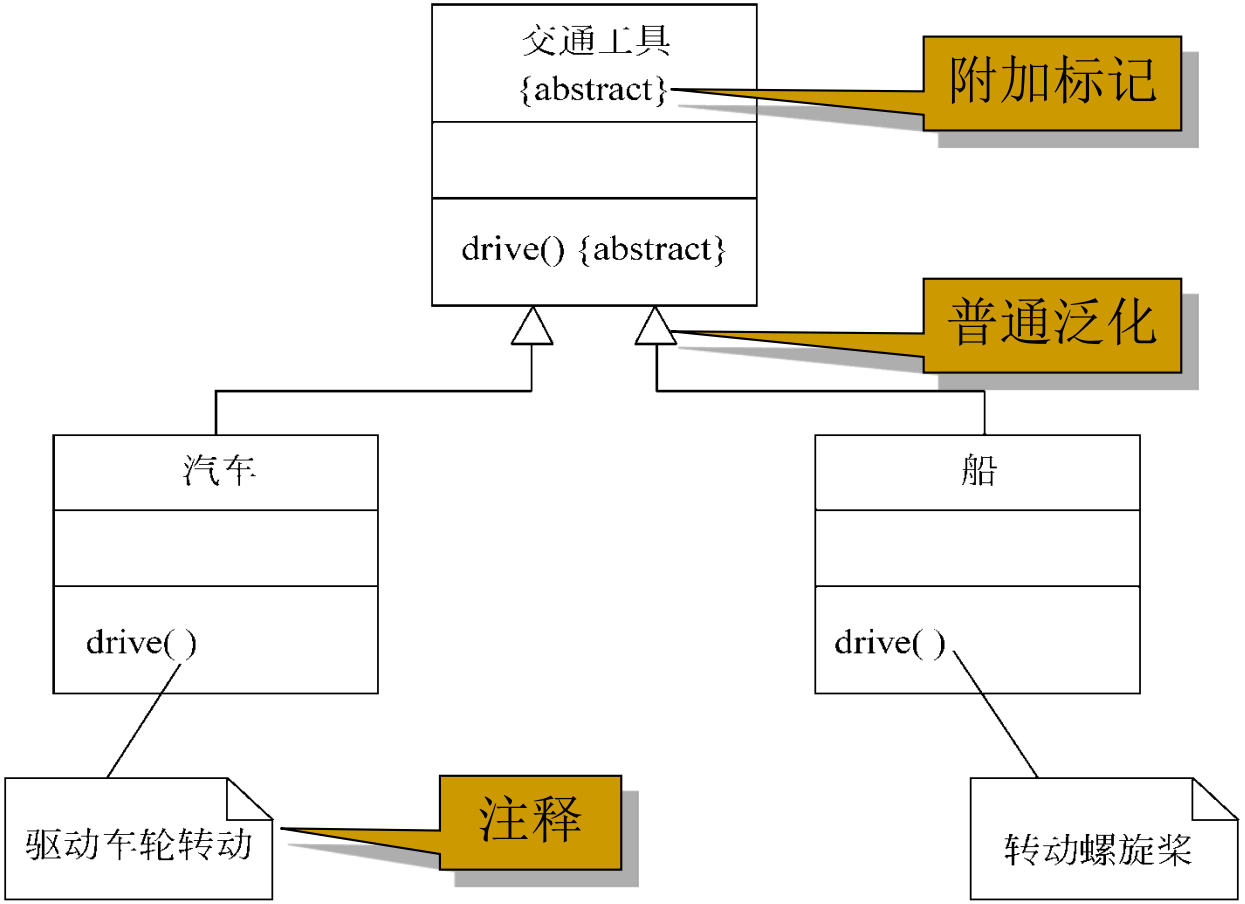
合成关系

- 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

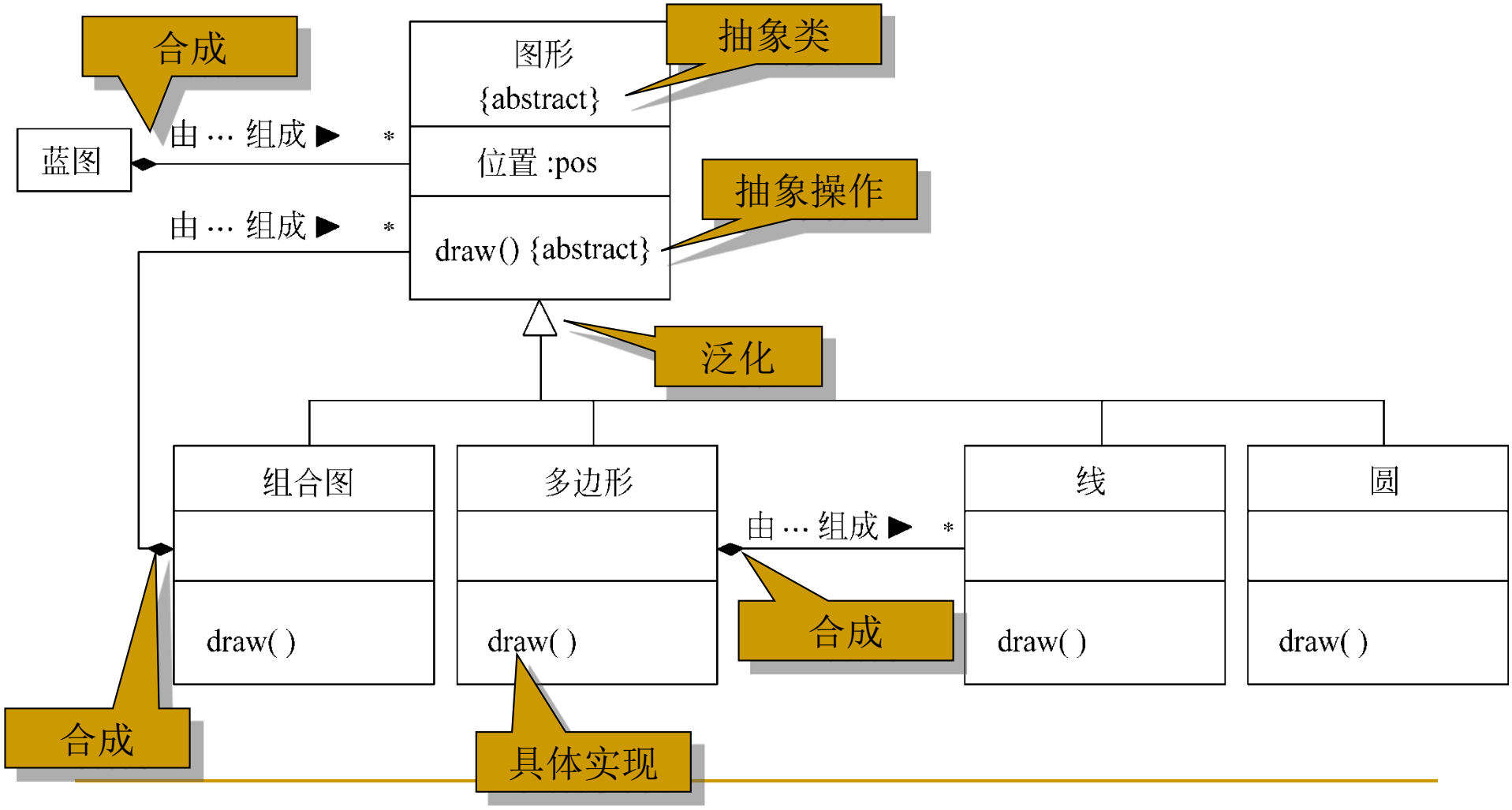


泛化

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。