



6.2.1 结构化地表达客观事物的术语

6.2.1.1 类与对象

—— 体现数据抽象

(1) 定义与表示

类 (Class) : 是一组具有相同属性、操作、关系和语义的对象的描述。

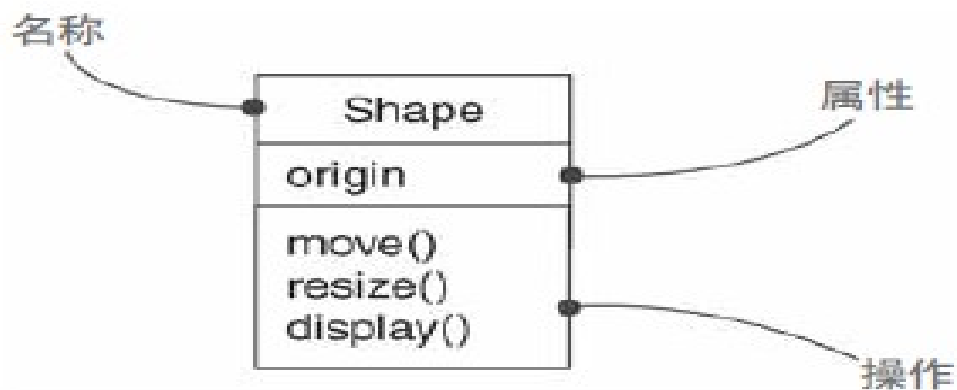
对象 (object) : 对象是类的一个实例。



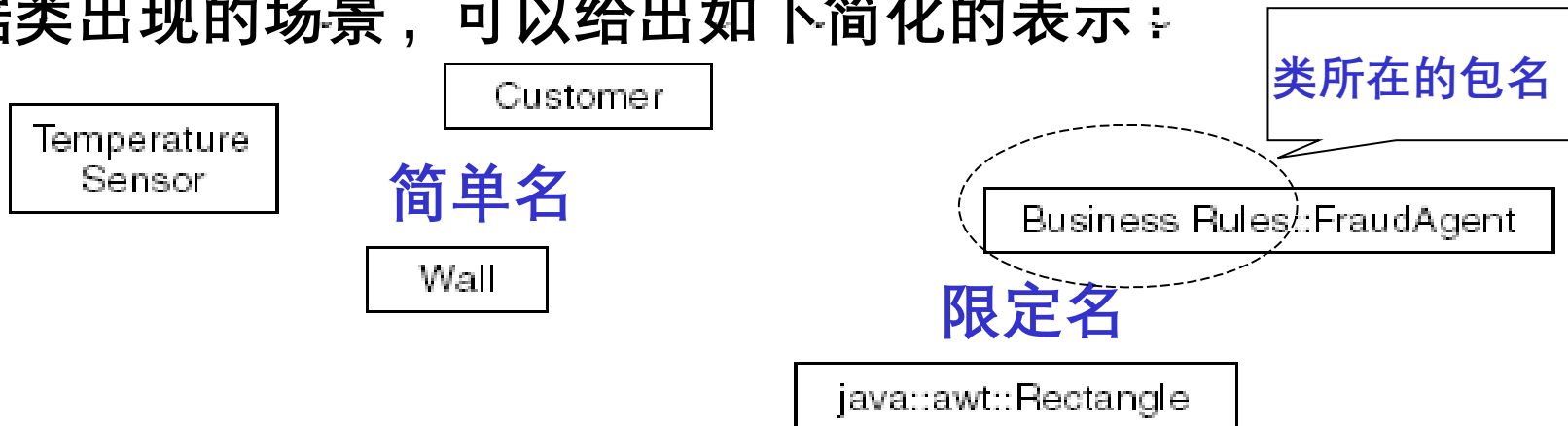
北京大学



类表示为具有三个栏目的矩形，如下所示：



依据类出现的场景，可以给出如下简化的表示：





类可以是抽象类，即没有实例的类，此时类名采用斜体字。
例如：

<i>Window</i>

<i>Window</i>
size:Area
visibility:Boolean
<i>display()</i>
<i>hide()</i>





(2) 类名 (类的标识)

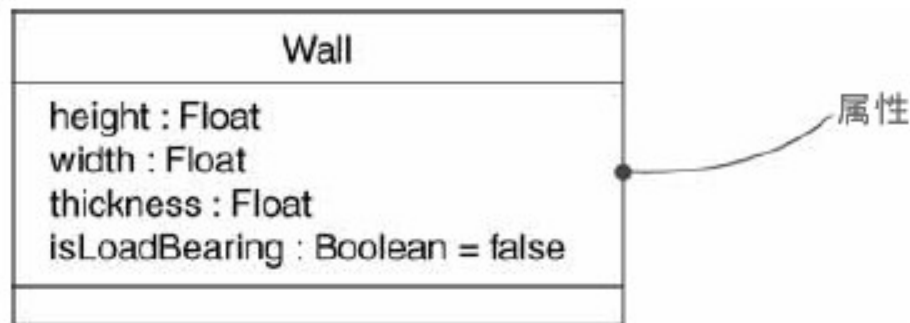
- ① 类名使用黑体字，第一个字母通常要大写，并位于第一栏的中央。
- ② 类名往往是从正被建模系统的词汇表中提取的简单名词或名词短语。



(3) 属性 (attribute)

属性是类的一个命名特性，由该类的所有对象所共享，用于表达对象状态的数据。

表示：



- ① 一个属性往往具有所属的类型，用于描述该特性的实例可以取值的范围。
- ② 类的一个对象对每一个属性应有特定的值。
- ③ 一个类可以有多个属性，也可以没有属性。






属性的作用范围：

- ① **实例范围的属性**：一个类的所有对象具有相同的属性即属性的个数、名称、数据类型相同，但属性值可不同，并随程序的执行而变化。
- ② **类范围的属性**：描述类的所有对象共同特征的一个数据项，对于任何对象实例，它的属性值都是相同的，通常对属性加下划线来表示该属性为类范围的属性。

注：如 C++ 中冠以 static 的成员变量和 smalltalk 中的 class attribute 都是类属性。

Frame	
header:FrameHeader	_____
<u>uniqueID:Long</u>	
	

实例范围的属性
类范围的属性



定义属性的格式为：

[可见性] 属性名 [: 类型] [多重性] [= 初始值] [{ 特性串 }]

注：加了方括号的内容是可选的。

① 可见性

表明该属性是否可以被其它类所使用。

其可见性的值可以为：

- + 公有的：可供其它类使用之；

- # 受保护的：其子类可以使用之；

- 私有的：只有本类的操作才能使用之；

- ~ 包内的：只有在同一包中声名的类才能使用之。

也可以使用关键字 `public`、`protected`、`private` 和 `package`，分别表示公有的、受保护的、私有的和包内的。



北京大学



为什么引入可见性？

引入“可见性”的目的——

为了支持信息隐蔽这一软件设计原则。所谓信息隐蔽是指在每个模块中所包含的信息（包括表达信息的数据以及表达信息处理的过程）不允许其它不需要这些信息的模块访问。信息隐蔽是实现模块低耦合的一种有效途径。



北京大学



② 属性名

属性名是一个表示属性名字的标识串。通常以小写字母开头，左对齐。

③ 类型

类型是对属性实现类型的规约，与具体实现语言有关。

例如：name:String

其中，“name”是属性名，而“String”是该属性的类型。

④ 多重性

多重性用于表达属性值的数目。即该类实例的这一特性可以具有的值的范围。

例如：points:Point[2..*]

多重性是可以省略的。在这种情况下，多重性是 1..1。即属性只含一个值。如果多重性是 0..1，就有可能出现空值。

例如：name:String[0..1]



北京大学



⑤ 初始值

初始值是与语言相关的表达式，用于为新建立的对象赋予初始值。例如：`origin:Point=(0,0)`

初始值是可选的。如果不声明对象这一属性的初始值，那么就要省略语法中的等号。对象的构造函数可以参数化或修改默认的初始值。

⑥ 性质串

如果说“类型”、“多重性”以及“初始值”都是围绕一个属性的可取值而给出的，那么“性质串”是为了表达该属性所具有的性质而给出的。例如：

`a:integer=1{frozen}`

其中，“frozen”是一个性质串，表示属性是不可以改变的。如果没有对一个属性给出这一性质串，那么就认为该属性是可以改变的。





属性的声明举例：

origin 只有属性名

+ origin 可见性和属性名

origin : Point 属性名和类型

name : String[0..1] 属性名、类型和多重性

origin : Point=(0,0) 属性名，类型和初始值

id: Integer{readonly} 属性名，类型和性质串

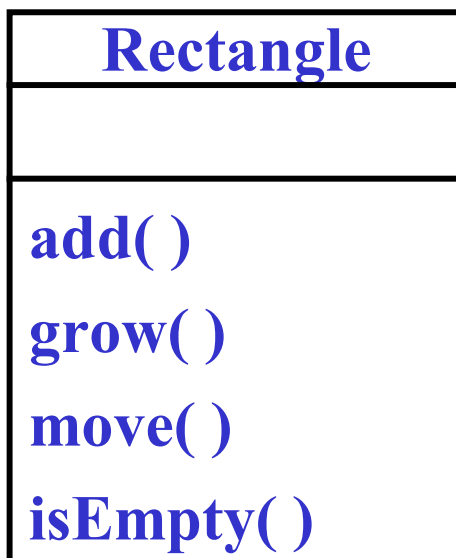




(4) 操作 (operation)

操作是对一个类中所有对象要做的事情的抽象。

表示：



- ① 一个类可以有多个操作，也可以没有操作。
- ② 操作名除第一个词之外，其他每个词的第一个字母要大写



北京大学



- ③ 操作名往往是描述其所在类的行为的动词或动词短语。
- ④ 可以通过给出操作的特征标记进一步描述之，特征标记通常包括参数名、类型和默认值；如果该操作是一个函数，那么其特征标记还包括返回类型。如下所示：



TemperatureSensor
<code>reset()</code> <code>setAlarm(t:temperature)</code> <code>value():Temperature</code>





⑤ 操作可以是抽象操作，即没有给出实现的操作。此时的操作名采用斜体。例如：

<i>Window</i>
<i>size:Area</i>
<i>visibility:Boolean</i>
<i>display()</i> <i>hide()</i>

注：抽象操作映射到 C++ 称为纯虚操作

⑥ 调用一个对象上的操作可能会改变该对象的数据或状态。





操作的作用范围：

类 名
<u>操作 1</u> <u>操作 m</u>

类范围的操作和实例范围的操作的唯一区别是：
类范围的操作名和类型表达式要加下划线

类范围的操作和实例范围的操作的表示



北京大学



表达操作的完整语法格式为

:

(表)][: 返回类型][{ 性质串 }]

其中:

① 可见性 如同属性的可见性一样，其值可以为：

- + 公有的 可供其它类访问之；
- # 受保护的 其子类能访问之；
- 私有的 只有本类的操作才能访问之；
- ~ 包内的 只有在同一包中声名的类才能访问之。

② 操作名

- 操作名一般是一动词或动词短语，通常以小写字母开头，左对齐；
- 如果操作名是动词短语，除第一个词外，其余每个词的第一个字母为大写，例如 isEmpty()；
- 若操作是一个抽象操作，则以斜体字表示之。



北京大学



③ 参数表

给出该操作的参数。一个操作可以有参数表，也可以没有。如果有参数表的话，其语法为：

[方向] 参数名：类型 [= 默认值]

- 方向是对输入 / 输出的规约，其取值可以为：
 - in 输入参数，不能修改之
 - inout 输入参数，为了与调用者进行信息通讯，可能要对之进行修改。
 - out 输出参数，为了与调用者进行信息通讯，可能要对之进行修改。
- 类型是实现类型的（与语言有关）规约；
- 默认值是一个值表达式，用最终的目标语言表示。
该项是可选的。

注释：out 或 inout 参数等价于返回参数和 in 参数。提供 out 和 inout 参数是为了与较老的编程语言相兼容。可用显式的返回参数来代替。





④ 返回类型

返回类型是对操作的实现类型或操作的返回值类型的规约，它与具体的实现语言有关。

- 如果操作没有返回值（例如 C++ 中的 void），就省略冒号和返回类型。
- 当需要表示多个返回值时，可以使用表达式列表。
- 根据实际问题的需要，可以省略全部的参数表和返回类型，但不能只省略其中的一部分。





⑤ 性质串

给出应用于该操作的性质值。该项是可选的，但若省略操作的性质串，就必须省略括号。UML 提供了以下标准的性质值：

- **leaf**：指明该操作是“叶子”操作；
- **abstract**：指明该操作是抽象操作；
- **query**：指明该操作的运行不会改变系统状态，即是完全没有副作用的纯函数。
- **sequential**：指明在该类对象中，一次仅有一个控制流。
- **guarded**：指明执行该操作的条件，实现操作调用的顺序化，即一次只能调用对象的一个操作，以保证在出现多控制流的情况下，对象具有的语义和完整性。
- **concurrent**：指明来自并发控制流的多个调用可以同时作用于一个对象的任何一个并发操作，而所有操作均能以正确的语义并发进行。并发操作必须设计成：在对一个对象同时进行顺序的或监护的操作情况下仍能正确地执行。
- **static**：指明该操作没有关于目标对象的隐式参数，其行为如同传统的全局过程。

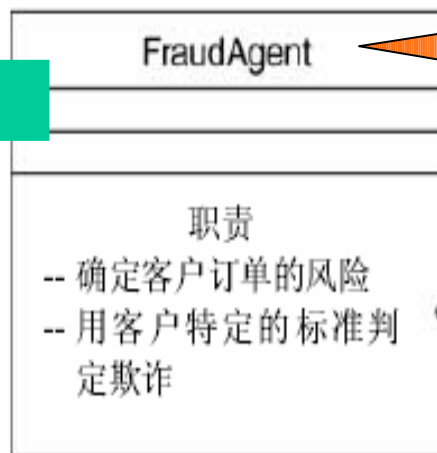




(5) 类的责任 (responsibility)

●在模型化一类事物中，其起始点往往是给出该类的责任，即一个类承诺的任务（contract）或职责（obligation）。

表示：



在信用卡应用系统中，类 FraudAgent 负责处理汇票，并决定汇票是否合法及有欺诈性

图 4-8 职责

注解 职责是自由形式的文本。实际上，可以把单个的职责写成一个短语、一个句子或（最多）一段短文。

- 
- 在实践中，每个结构良好的类至少要有一个职责。

例如：一个“调制解调器”有四个操作：

modem

```
{ public void dial (string pno);  
  public void hangup ();  
  public void send (char c);  
  public void recv ();      }
```

根据责任的定义，可以认为 modem 有两个职责：

- ① 连接处理 （ **public void dial (string pno);
public void hangup ();** ）
- ② 数据通信 （ **public void send (char c);
public void recv ();** ）





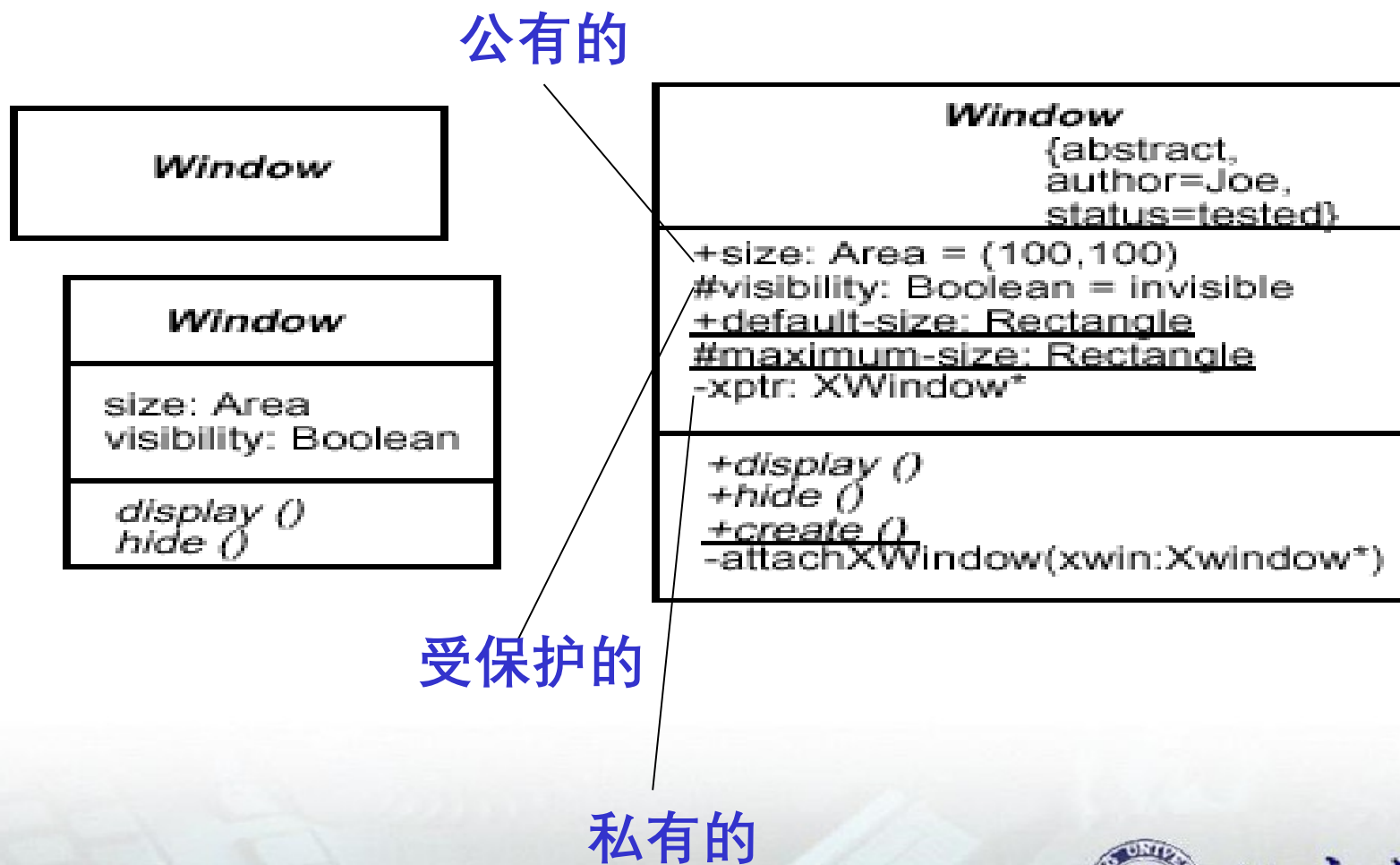
- 类的责任是由类中定义的属性和操作实现的。在精化类当中，需将责任转换为一组能够很好地完成责任的属性和操作。

在规约类时，为了使该类具有更多的语义信息，除了规约以上提到的责任、属性和操作外，有时还需要规约类的其它特征，例如属性和操作的可见性，操作的多态性等。



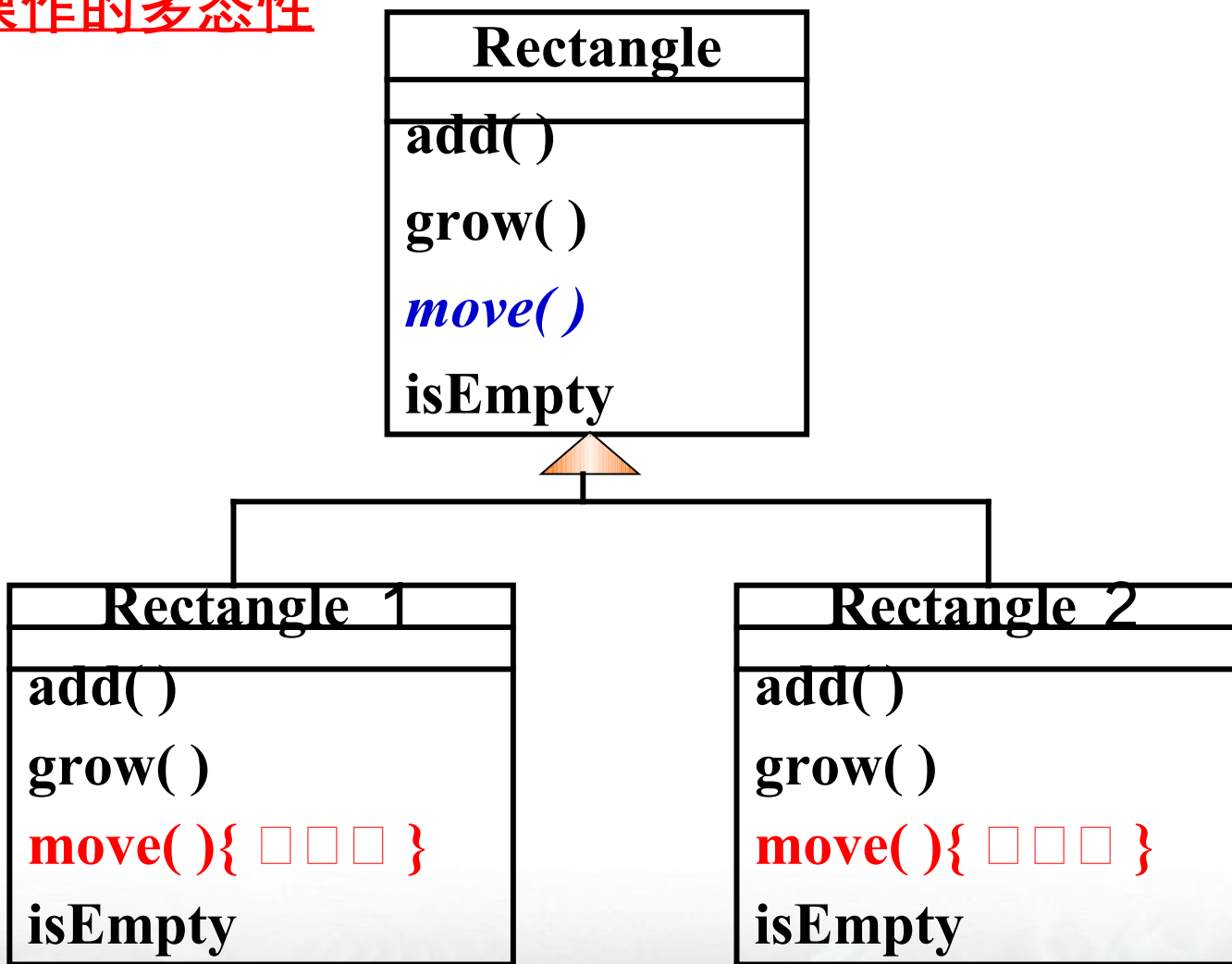


(6) 属性和操作的可见性





(7) 操作的多态性





(8) 类在建模中的主要用途（即常用的建模技术）

① 模型化一个系统中的词汇（即对系统的词汇建模）

要做以下工作：

- 识别用户或实现者用于描述问题或者描述解决方案的那些事物，将其表示为类。

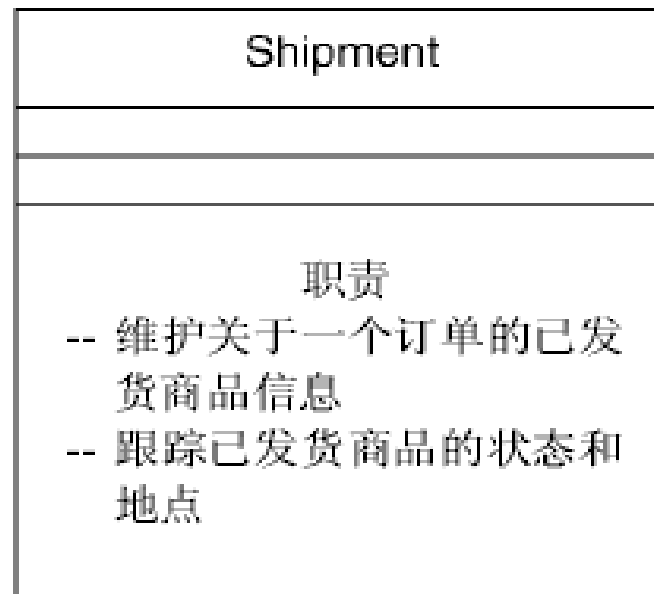
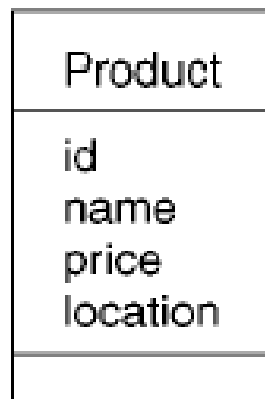
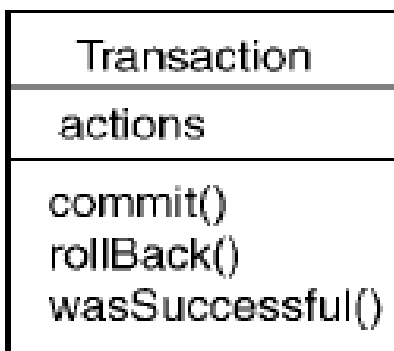
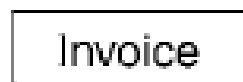
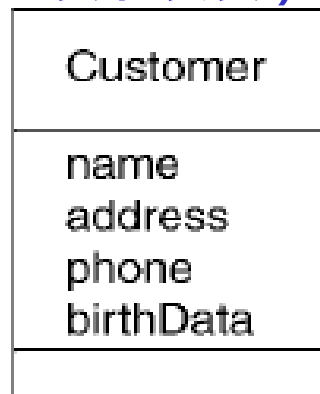
注：其中可使用 USE CASE 分析技术。

- 标识每个类的责任集。确保每一个类通过这一责任集予以清晰地定义，并使这些责任在所有类之间得到很好的均衡。
- 为每个类的责任的实现，提供了所需的属性和操作。





例如：从一个零售系统抽取的一组类，其中包括 Customer、Order 和 Product。这个图也包含了一些来自问题的词汇表中的其他抽象，如 Shipment（用于跟踪订单）、Invoice（用于按订单开发票）和 Warehouse（在发货之前储存货物），还有一个与解决方案相关的抽象 Transaction（用于订货和发货）。





② 模型化系统中的责任分布（即对系统中的责任分布建模）

其目标是：均衡系统中每一个类的责任，避免类过大或过小。

- 过大—难于复用；
- 过小—难于理解和管理。

要做以下工作：

- 为了完成某些行为，标识一组紧密协同工作的类。
- 对以上的每个类，标识它的一组职责。
- 从整体上观察这些类，把其中职责过多的类分解为一些较小抽象的类，而把责任过于锁碎的一类合并为一个较的类，继之重新分配责任。
- 考虑这些类的相互协作方式，调整它们的责任，使协作中没有哪个类的职责过多或过少。





例如： 类 Model、类 View、类 Controller 中的责任分布
， 其中没有过大或过小的类

Model
职责 — 管理模型的状态

Controller
职责 -- 同步模型及其视图上的 的变化

View
职责 -- 将模型描绘在屏幕上 -- 管理视图的移动和尺寸变化 -- 截取用户事件



③ 模型化基本类型（对基本类型建模）

在其他极端的情况下，所建模的事物可能直接取自于实现一个解的编程语言。通常这些抽象包括简单类型，如整数、字符和串，乃至自定义的枚举类型。

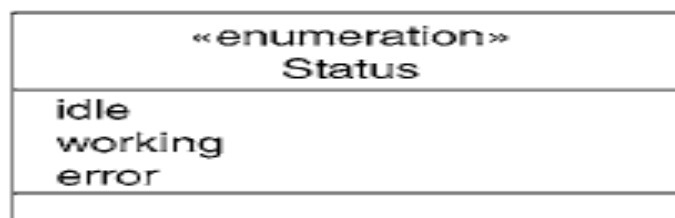
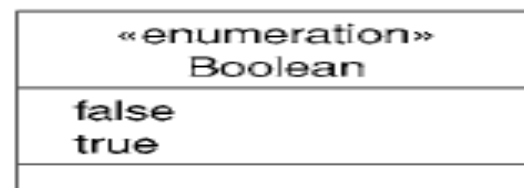
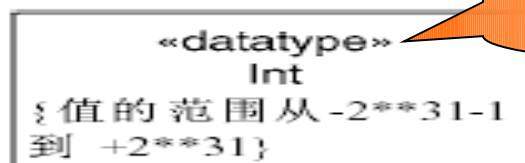
对简单类型建模，要做以下工作：

- 如对抽象为类型或枚举的事物建模，可用带有适当**衍型**的类表示；
- 若需要详述与该类型相关的值域，可使用约束。

可以把衍型看作元类型
(一种定义其他类型的类型),

例如：

衍型表示为：用双尖括号括起来的
名字，并放在别的元素名称之上





(9) 类在使用中应注意的问题

每个类都应是某个有形的事物或概念的抽象。一个结构良好的类，应符合以下条件：

- 是对问题域或解域中的事物的明确抽象。
- 嵌入了一个小的、明确定义的责任集，并能很好地实现之。
- 清晰地分离了抽象的规约和实现。





主动类（active class）

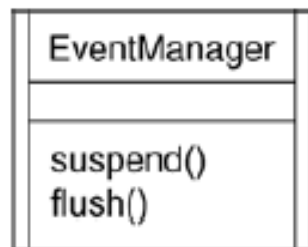
-- 体现并发行为抽象

是一种至少具有一个进程或线程的类，因此它能够启动控制活动。
——UML 用户指南（第二版）

主动对象 (active object)：至少有一个操作不需要接收消息就能主动执行的对象，用于描述具有主动行为的事物。
主动对象的类叫做主动类。

——面向对象的系统分析（第二版），邵维忠等编著

表示：



（1）从事物的主动行为认识主动对象；

（2）从系统的执行认识主动对象。



北京大学



关于识别类、属性和操作



北京大学



一、识别类

1、研究问题域和用户需求

(1) 研究用户需求，明确系统责任

阅读：阅读一切与用户需求有关的书面材料

交流：与用户交流，澄清疑点，

纠正用户不切实的要求或不确切的表达

调查：到现场调查（只限于澄清需求）

记录、整理：产生一份符合工程规范、确切表达系统责任的需求文档

(2) 研究问题域

亲临现场调查，掌握第一手资料

听取问题域专家的见解

阅读与问题域有关材料

借鉴相同或类似问题域已有的系统开发经验及文档



北京大学



(3) 确定系统边界

就是划出被开发的系统和该系统打交道的人或物之间的明确界限，并确定它们之间的接口。

在系统边界之内，是系统本身所包含的对象。在系统边界以外，是系统外部的活动者。主要是人、设备和外系统三种外部活动者。





2、策略与启发

(1) 考虑问题域：



可以启发分析员发现对象的因素包括：人员、组织、物品、设备、抽象事物、事件、文件及结构等。

●人员：（a）需要由系统保存和管理其信息的人员，如户籍管理系统中的每个居民；（b）应该在系统中完成某些功能，提供某些服务的人员，如户籍管理员。符合上述情况之一者，应考虑用相应的人员对象来描述。





- **组织：**在系统中发挥一定作用的组织结构。如工作班组等。
- **物品：**需要由系统管理的各种物品。如经营的商品等。
- **设备：**在系统中动态地运行、由系统进行监控或供系统使用的各种设备、仪表、机器及运输工具等。
- **抽象事物：**指没有具体的物理形态，却对用户的业务具有实际意义的逻辑上的事物。
- **事件：**指那些需要由系统长期记忆的事件。
- **文件：**泛指在人类日常的管理和业务活动中使用的各种各样的表格、档案、证件和票据等文件。
- **结构：**通过考虑结构可以得到一种启发——从已经发现的对象联想到其他更多的对象





(2) 考虑系统边界：

考虑系统边界，可以启发分析员发现一些与系统边界以外的参与者进行交互，并且处理系统对外接口的对象。

人员

设备

外系统

从不同的
角度考虑
人员和设备

人员：作为系统以外的参与者与系统进行直接交互的各类人员，如系统的操作员、直接使用系统的用户等。

设备：作为系统以外的参与者与系统相连并交换信息的设备。

外系统：与系统相连并交换信息的其他系统。

(3) 考虑系统责任：

检查每一项功能需求是否有相应的对象提供，发现新的对象



北京大学



3、审查与筛选

(1) 舍弃无用的对象

通过属性判断：

是否通过属性记录了一些对参与者或对系统的其他对象有用的信息？（即这个对象所对应的事物，是否有些信息需要在系统中进行保存和处理？）

通过操作判断：

是否通过操作提供了某些有用的功能？（即这个对象所对应的事物，是否有某些行为需要在系统中模拟，并在系统中发挥一份作用？）

二者都不是——无用



北京大学

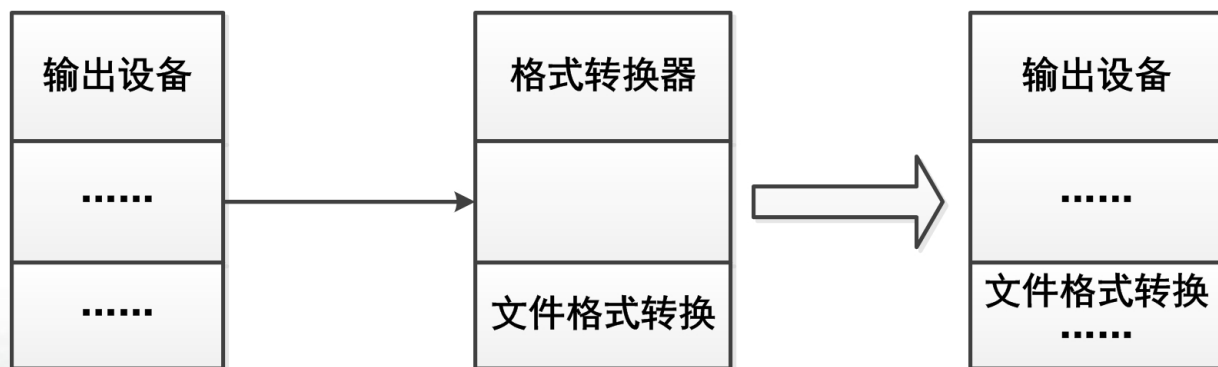


(2) 对象的精简

只有一个属性的对象



只有一个操作的对象





(3) 与实现条件有关的对象

例如：与

图形用户界面（ GUI ）系统、

数据管理系统、

硬件 及

操作系统有关的对象

—— 推迟到 OOD 考虑



北京大学



4、识别主动对象

(1) 考虑问题域和系统责任
哪些对象需呈现主动行为?

(2) 从需求考虑系统的执行情况是否需要并发执行?
控制线程的起点在哪个对象?

(3) 考虑系统边界以外的参与者与系统中哪些对象直接进行交互?

如果一个交互是由系统外的参与者发起的，第一个处理该交互的对象是主动对象

在分析阶段
不能完全确定



北京大学



5、对象分类，建立类图中的类

(1) 对象分类

使用问题域和系统责任知识，为每组具有相同属性和操作的对象定义一个类

(2) 异常情况的检查和调整

* 类的属性或操作不适合全部对象实例

例：“汽车”类的“乘客限量”属性

问题：分类不够详细——进一步划分特殊类

* 属性及操作相同的类

经过抽象，差别很大的事物可能只保留相同的特征

例如“吸尘器”和“电子琴”作为商品销售

——考虑能否合并为一个类





* 属性及操作相似的类

—— 考虑能否提升出一个一般类或部分类（如轿车和货车，提取增加一般类“汽车”；机床和抽风机，提取部分类“电动机”）

* 同一事物的重复描述

例：“职员”和“工作证”
—— 取消其中一个





(3) 类的命名

- * 使用名词，避免无意义的符号
- * 反映个体而不是群体（如书而非书籍）
- * 适合该类及其特殊类的全部对象实例
- * 使用问题域通用、规范的词汇
- * 在中国：可用中、英文双重命名

(4) 建立类图的对象层

- * 用类符号表示每个对象类，填写类的名称；
- * 对于已经确认的主动对象，注意标注为主动类
- * 填写类规约中关于每个类的详细信息；
- * 发现的属性与操作，可以随时加到类符号中。





二、识别属性

(1) 策略与启发

- * 按常识这个对象应该有哪些属性?(例如人的姓名、职业、地址等)
- * 在当前的问题域中, 对象应该有哪些属性?(例如商品的条形码)
- * 根据系统责任, 这个对象应具有哪些属性?(持卡人的使用地点)
- * 建立这个对象是为了保存和管理哪些信息?
- * 对象为了实现操作的功能, 需要增设哪些属性?
(例如传感器对象, 为了实现其定时采集信号的功能, 需要一个“时间间隔”属性, 为了实现其报警功能, 需要一个“临界值”属性)
- * 对象是否需要通过专设的属性描述其状态?
(例如设备对象, 在关闭、待命、运行、故障等不同状态将呈现不同的行为, 需要为其设置一个“状态”属性)
- * 用什么属性表示聚合和关联?
(对于关联, 应该在关联一端的类中定义一个属性, 来指出另一端的哪个对象与本端的对象发生关联, 其数据类型是指向另一端对象的指针或对象标识)





(2) 审查与筛选

- * 是否体现了以系统责任为目标的抽象
(例：书的重量)？
- * 是否描述对象本身的特征
(例：课程—电话号码)？
- * 是否破坏了对对象特征的“原子性”
- * 是否可通过继承得到？
- * 可以从其它属性直接导出的属性

(3) 与实现条件有关的问题都推迟到 OOD 考虑

- * 规范化问题 (OOA 中定义的对象属性可以是任何数据类型，数据类型的规范化工作在 OOD 中考虑)

- * 对象标识问题

- * 性能问题 (如为了提高操作的执行速度，增加一些属性来保持操作的阶段性执行结果)



北京大学



(4) 属性的命名：原则与类的命名相同：

- 使用名词或带定语的名词；
- 使用规范的、问题域通用的词汇；
- 避免使用无意义的字符和数字。

语言文字的选择要和类的命名要一致。

定位原则：一个类的属性必须适合这个类和它的全部特殊类的对象，并在此前提下充分运用继承。

(5) 属性的详细说明

要在类规约中对属性进行详细说明，其中包括：属性的解释、数据类型和具体限制等。

* 属性的文字解释：例如“课程”对象的“学时”属性，其解释为“课堂讲授学时数，每学时为 50 分钟”

* 属性的数据类型：常用的数据类型；表示整体 - 部分结构或关联的属性类型可以是类或某一类对象的指针。



北京大学



*** 属性所体现的关系：**用于表示整体 - 部分关系或关联的属性，应该特别指明并加以解释。

例如对“课程”对象的“主讲教师”属性，可说明为：“表示课程与教师对象间的关联，指出该课程由哪个教师主讲。”

*** 实现要求及其它：**如属性的取值范围、精度要求、初始值、度量单位、数据完整性及安全性要求、存取限制条件等。

如果一个属性实现时应作为类属性处理，也要在这里明确指出。

