中国石油大学（华东）
CHINA UNIVERSITY OF PETROLEUM

软件工程

主要内容

第一章 软件工程学概述
第二章 可行性研究
第三章 需求分析
第四章 总体设计
第五章 详细设计
第六章 编码与测试
第七章 软件维护
第八章 面向对象方法学
第九章 面向对象分析设计与实现
第十章 软件项目管理

第八章　面向对象方法学

第一节　面向对象方法学概述

第二节　面向对象的概念

第三节　面向对象建模

第八章　面向对象方法学

第一节　面向对象方法学概述

- 面向对象方法学的出发点和基本原则，是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解决问题的方法与过程，使描述问题的问题空间(也称为问题域)与实现解法的解空间(也称为求解域)在结构上尽可能一致。
- 客观世界中的实体既具有静态的属性又具有动态的行为。面向对象方法是以数据或信息为主线，把数据和处理相结合的方法。
- 面向对象方法把对象作为由数据及可施加在这些数据上的操作所构成的统一体。对象不仅能被动地等待外界对它进行操作，而且也可以主动处理相关事件的请求。
- 面向对象方法把程序看作是相互协作而又彼此独立的对象集合。

第八章　面向对象方法学

一、面向对象方法学的要点

（1）客观世界是由各种对象组成的，任何事物都是对象，复杂的对象可以由比较简单的对象以某种方式组合而成。

面向对象方法用对象分解取代了传统方法的功能分解

（2）把所有对象都划分成类(class)，每个类都定义了一组数据和一组方法。数据用于表示对象的静态属性，方法是该类对象共享的、允许施加在该类对象上的操作。

（3）按照父类(或称为基类)与子类(或称为派生类)的关系，把若干个相关类组成一个层次结构的系统(也称为类等级)。

具有继承特性

(4)对象彼此间仅能通过发送消息互相联系。

第八章　面向对象方法学

- **什么是面向对象**

根据Coad 和 Yourdon 的定义，按照以下4个概念设计和实现的系统，称为是面向对象的。

面向对象=对象 (object)

+类 (classification)

+继承(inheritance)

+通信 (communication with messages)

二、面向对象方法学的优点

- 与人类习惯的思维一致
- 稳定性好

  传统方法的结构依赖于功能，易变；面向对象方法以object模拟实体，而实体相对稳定。
- 可重用性好

  传统方法的标准函数缺少必要的“柔性”；继承机制实现了重用，且易于修改和扩充。
- 可维护性好

第二节　面向对象的概念

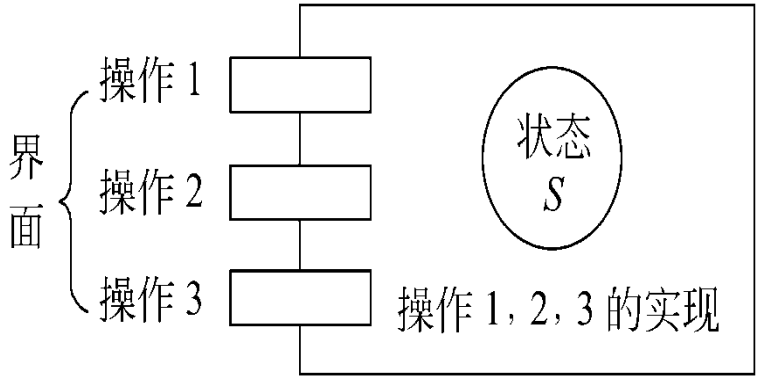
理解面向对象的基本概念对于学习和掌握面向对象的开发方法是十分重要的。

- 对象(Object)
- 类（Class）
- 继承（Inheritance）
- 消息(Information)
- 多态性(Polymorphism)
- 永久对象(Persistent object)

1. 对象(Object)

- 对象是封装了数据结构及可以施加在这些数据结构上的操作的封装体，这个封装体有可以惟一地标识它的名字，而且向外界提供一组服务(即公有的操作)。通常把对象的操作称为服务或方法。

Object : = ID + Method + Attribute + Message



- 对象的特点

① 以数据为中心，不设与数据无关的操作；

② Object是主动的：不被动地等待被处理，外部只能通过message请求它执行它的某个操作，处理它的私有数据；

③ 实现了数据封装，具有黑盒性：外部操作时，无须知道该object内部的数据结构及算法；

④ 具有并行性：不同object各自独立地处理自身数据，彼此间仅通过传递message完成通信；

⑤ 模块独立性好：内聚强、耦合松

2. **类（Class）**：又称对象类（Object Class），是一组具有相同属性和相同操作的对象的集合。

3. **实例(Instance)**：某个class描述的具体对象。

4. **消息(Message)**：消息就是向对象发出的服务请求（互相联系、协同工作等）。

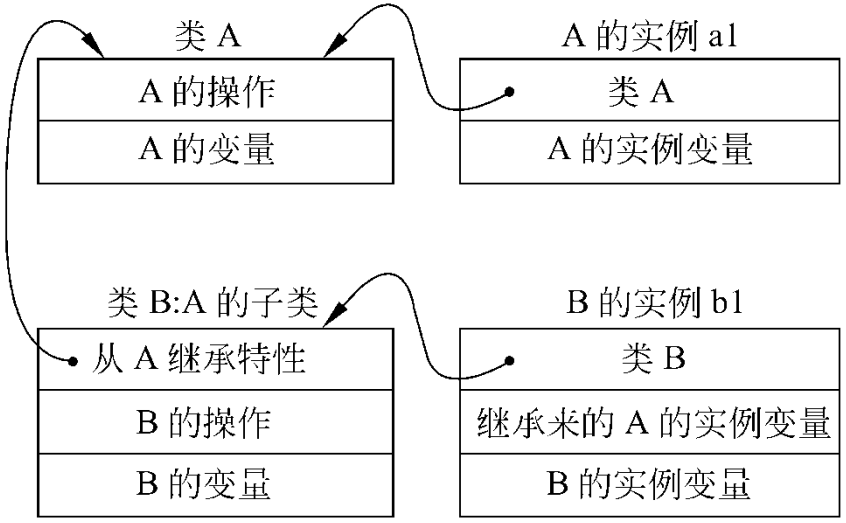
   object_ID. method_ID (parameter(s))。

5. **方法(Method)**：object能做的操作,亦称为服务、响应， 在class 中须定义相应的代码。

6. **属性(Attribute)** ：就是类中所定义的数据，它是对客观世界实体所具有的性质的抽象。类的每个实例都有自己特有的属性值。

7. 继承 （Inheritance）

- 子类自动共享父类的attributes和methods，而不必重复定义。
- 当一个类只允许有一个父类时，也就是说，当类等级为树形结构时，类的继承是单继承；当允许一个类有多个父类时，类的继承为多重继承。



实现继承机制的原理

7. 多态性(polymorphism)

- 在类等级的不同层次中可以共享一个方法的名字，不同层次中的每个类按自己的需要来实现这个方法。因此，相同的操作的消息发送给不同的对象时，每个对象将根据自己所属类中所定义的方法去执行，产生不同的结果。

第三节　面向对象建模

- 模型是为了理解事物而对该事物做出的一种抽象，是对事物的一种无歧义的书面描述。通常，模型由一组图示符号和组织这些符号的规则组成，利用它们来定义和描述问题域中的术语和概念。
- 使用面向对象方法开发软件，通常需要建立3种形式的模型：
  ① 描述系统数据结构的对象模型
  ② 描述系统控制结构的动态模型
  ③ 描述系统功能的功能模型
- 对象模型是最重要、最基本、最核心的。

一、对象模型

定义构成系统的类和对象及对象彼此间的关系的映射，它描述了系统的静态结构。常用UML提供的类图来建立对象模型。

1. 类图的基本符号

（1）定义类

类图描述类及类与类之间的静态关系--静态模型。

类(Class)

学生类

（2）定义属性

UML描述属性的语法格式如下：

可见性　属性名：类型名 = 初值{性质串}

- 属性的可见性（即可访问性）通常有下述3种：

  公有的(public)　　用加号（+）表示

  私有的(private)　　用减号（-）表示

  保护的(protected)　用井号（#）表示

- 注意:如果未声明可见性，则表示该属性的可见性尚未定义，没有默认的可见性。

- 枚举类型的属性往往用性质串列出可以选用的枚举值，不同枚举值之间用逗号分隔；也可以用性质串说明属性的其他性质，例如，约束说明{只读}表明该属性是只读属性。
- 类的属性中还可以有一种能被该类所有对象共享的属性，称为类作用域属性，也称为类变量。C++语言中的静态数据成员就是这样的属性。类变量在类图中表示为带下划线的属性，例如，发货单类的类变量“货单数”，用来统计发货单的总数，在该类所有对象中这个属性的值都是一样的。

（3）定义服务

服务也就是操作，UML描述操作的语法格式如下：

可见性 操作名（参数表）：返回值类型{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格式如下：

参数名： 类型名=默认值

- 当操作的调用者未提供实在参数时，该参数就使用默认值。
- 与属性类似，在类中也可定义类作用域操作，在类图中表示为带下划线的操作。这种操作只能存取本类的类作用域属性。

2. 表示关系的符号

（1）关联

关联（association）表示两个类的对象之间存在某种语义上的联系。

① 普通关联

- 普通关联是最常见的关联关系，只要在类与类之间存在连接关系就可以用普通关联表示。普通关联的图示符号是连接两个类之间的直线。
- 关联是双向的，可在一个方向上为关联起一个名字，在另一个方向上起另一个名字（也可不起名字）。为避免混淆，在名字前面（或后面）加一个表示关联方向的黑三角。

- 关联的重数

重数(multiplicity)表示多少个对象与对方对象相连接，常用的重数符号有：

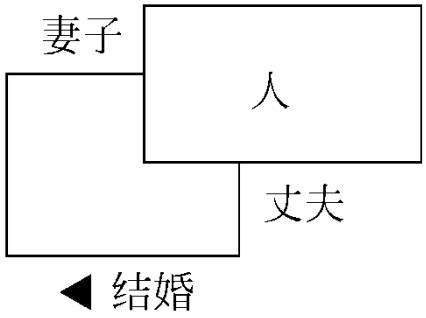
“0..1”	表示0到1个对象
“0..*”或“*”	表示0到多个对象
“1..*”或1+	表示1到多个对象
“1,3,7”	表示1或3或7个对象（枚举型）

重数的默认值为1。

上图表示：一个作家可以使用1到多台计算机，一台计算机可被0至多个作家使用

（2）关联的角色

在任何关联中都会涉及到参与此关联的对象所扮演的角色（即起的作用），在某些情况下显式标明角色名有助于别人理解类图。

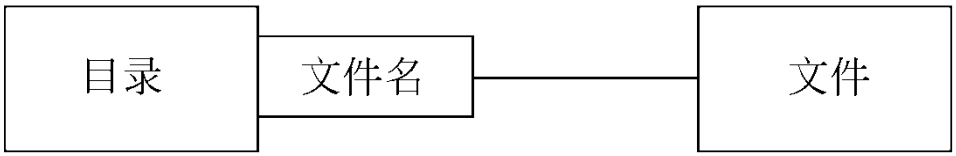


上图是一个递归关联（即一个类与它本身有关联关系）的例子：一个人与另一个人结婚，必然一个人扮演丈夫的角色，另一个人扮演妻子的角色。如果没有显式标出角色名，则意味着用类名作为角色名。

（3）限定关联

限定关联通常用在一对多或多对多的关联关系中，可以把模型中的重数从一对多变成一对一，或从多对多简化成多对一。在类图中把限定词放在关联关系末端的一个小方框内。

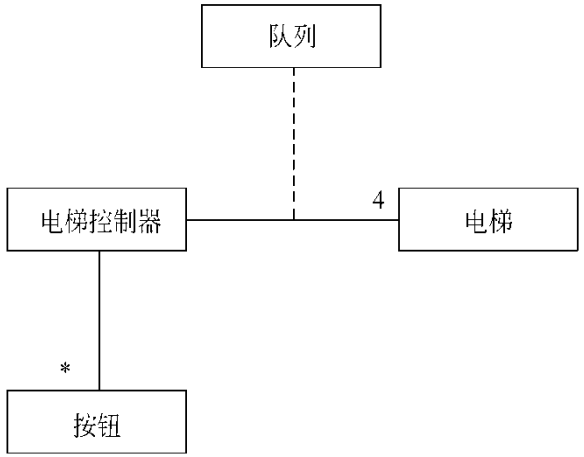
例如，某操作系统中一个目录下有许多文件，一个文件仅属于一个目录，在一个目录内文件名确定了惟一一个文件。利用限定词“文件名”表示了目录与文件之间的关系，可见，利用限定词把一对多关系简化成了一对一关系。



（4）关联类

为了说明关联的性质可能需要一些附加信息。可以引入一个关联类来记录这些信息。关联中的每个连接与关联类的一个对象相联系。关联类通过一条虚线与关联连接。

例如：一个电梯系统的类模型，队列就是电梯控制器类与电梯类的关联关系上的关联类。从右图可以看出，一个电梯控制器控制着4台电梯，这样，控制器和电梯之间的实际连接就有4个，每个连接都对应一个队列（对象），每个队列（对象）存储着来自控制器和电梯内部按钮的请求服务信息。电梯控制器通过读取队列信息，选择一个合适的电梯为乘客服务。关联类与一般的类一样，也有属性、操作和关联。



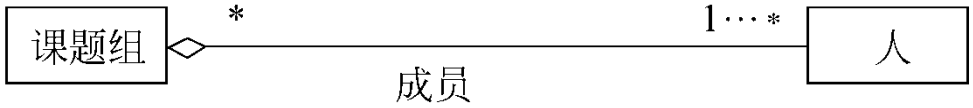
2. 表示关系的符号

（2）关联

聚集也称为聚合，是关联的特例。聚集表示类与类之间的关系是整体与部分的关系。除了一般聚集之外，还有两种特殊的聚集关系：共享聚集和组合聚集。

① 共享聚集

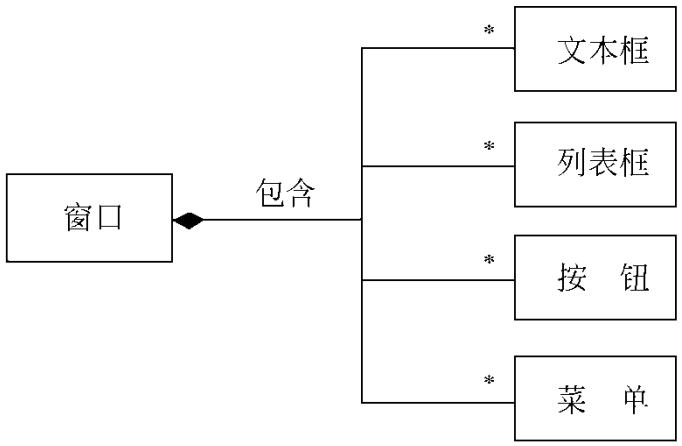
- 如果在聚集关系中处于部分方的对象可同时参与多个处于整体方对象的构成，则该聚集称为共享聚集。
- 一般聚集和共享聚集的图示符号，都是在表示关联关系的直线末端紧挨着整体类的地方画一个空心菱形。



② 组合聚集

组成/构成

- 如果部分类完全隶属于整体类，部分与整体共存，整体不存在了部分也会随之消失（或失去存在价值了），则该聚集称为组合聚集（简称为组成）。
- 例如，在屏幕上打开一个窗口，它就由文本框、列表框、按钮和菜单组成，一旦关闭了窗口，各个组成部分也同时消失，窗口和它的组成部分之间存在着组合聚集关系。可以看出，组合聚集用实心菱形表示。



3. 泛化

UML中的泛化关系就是通常所说的继承关系，它是通用元素和具体元素之间的一种分类关系。具体元素完全拥有通用元素的信息，并且还可以附加一些其他信息。

在UML中，用一端为空心三角形的连线表示泛化关系，三角形的顶角紧挨着通用元素。

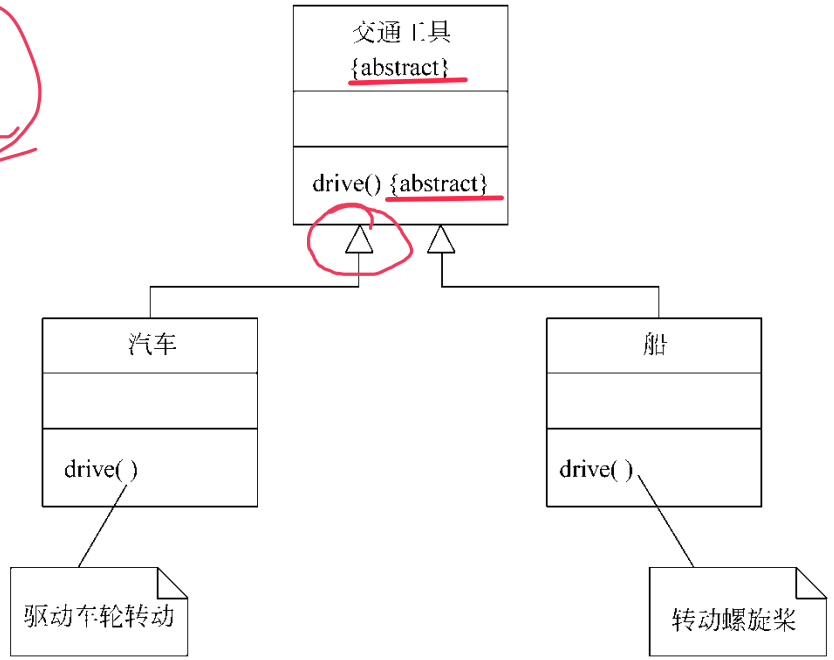
注意，泛化针对类型而不针对实例，一个类可以继承另一个类，但一个对象不能继承另一个对象。实际上，泛化关系指出在类与类之间存在“一般-特殊”关系。泛化可进一步划分成普通泛化和受限泛化。

3. 泛化

（1）普通泛化

- 没有具体对象的类称为抽象类。抽象类通常作为父类，用于描述其他类（子类）的公共属性和行为。图示抽象类时，在类名下方附加一个标记值{abstract}。

- 右图的两个折角矩形是模型元素“笔记”的符号，其中的文字是注释，分别说明两个子类的操作drive的功能。

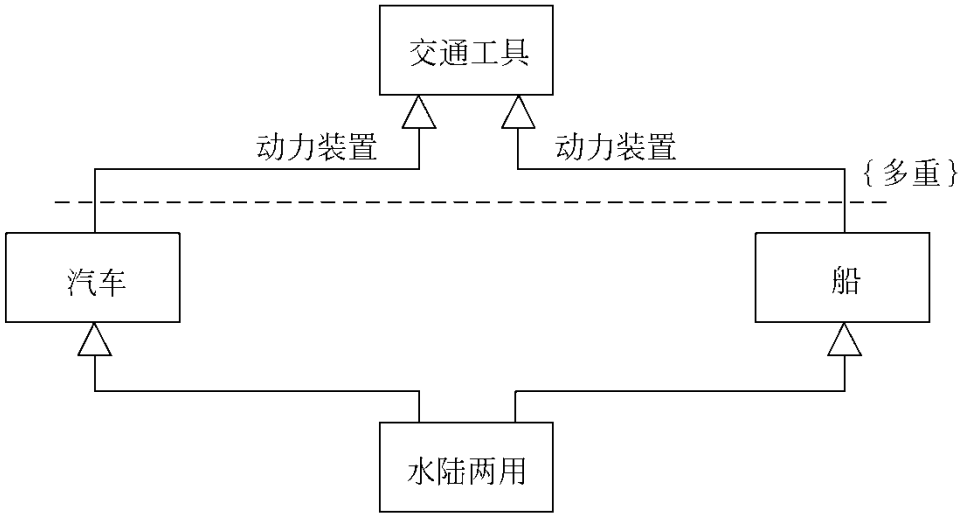


3. 泛化

（2）受限泛化

可以给泛化关系附加约束条件，以进一步说明该泛化关系的使用方法或扩充方法，这样的泛化关系称为受限泛化。预定义的约束有4种，都是语义约束：

- 多重继承：一个子类可以同时多次继承同一个上层基类。



- 不相交继承：一个子类不能多次继承同一个基类。如果图中没有指定{多重}约束，则是不相交继承，一般的继承都是不相交继承。
- 完全继承：父类的所有子类都已在类图中穷举出来了，图示符号是指定{完全}约束。
- 不完全继承：父类的子类并没有都穷举出来，随着对问题理解的深入，可不断补充和维护，这为日后系统的扩充和维护带来很大方便。不完全继承是一般情况下默认的继承关系。

4. 依赖和细化

（1）依赖

- 依赖关系描述两个模型元素（类、用例等）之间的语义连接关系：　其中一个模型元素是独立的，另一个模型元素不是独立的，它依赖于独立的模型元素，如果独立的模型元素改变了，将影响依赖于它的模型元素。
- 在UML的类图中，用带箭头的虚线连接有依赖关系的两个类，箭头指向独立的类。在虚线上可以带一个版类标签，具体说明依赖的种类。

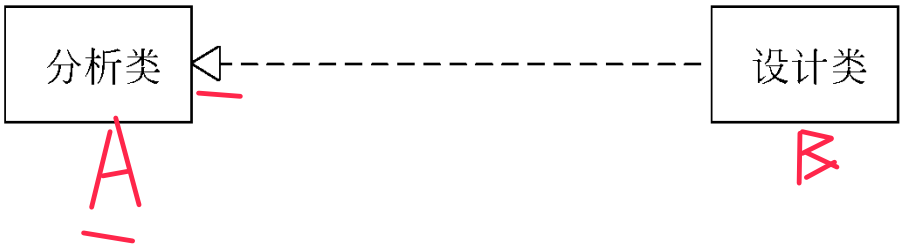
上图表示一个友元依赖关系，该关系使得B类的操作可以使用A类中私有的或保护的成员。

4. 依赖和细化

（2）细化

当对同一个事物在不同抽象层次上描述时，这些描述之间具有细化关系。

- 若A和B描述的是同一事物，它们的区别是抽象层次不同，若B是在A基础上的更详细的描述，则称B细化了A，或称A细化成了B。
- 细化的图示符号为由元素B指向元素A的、一端为空心三角形的虚线（注意：不是实线）。



二、动态模型

- 动态模型表示瞬时的、行为化的系统的“控制”性质。
- 常用状态图来描绘对象的状态、触发状态转换的事件以及对象的行为（对事件的响应）。
- 每个类的动态行为用一张状态图来描绘，各个类的状态图通过共享事件合并起来，从而构成系统的动态模型。

三、功能模型

- 功能模型表示变化的系统的功能性质，它指明了系统应该做什么，因此更直接地反映了用户对目标系统的需求。
- 面向对象方法中建立功能模型的方法：

  ① 由一组数据流图组成；

  ② UML的用例图也是进行需求分析和建立功能模型的强有力工具，用用例图建立起来的功能模型也称为用例模型。

- 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，它描述了开发者和用户对需求规格所达成的共识。

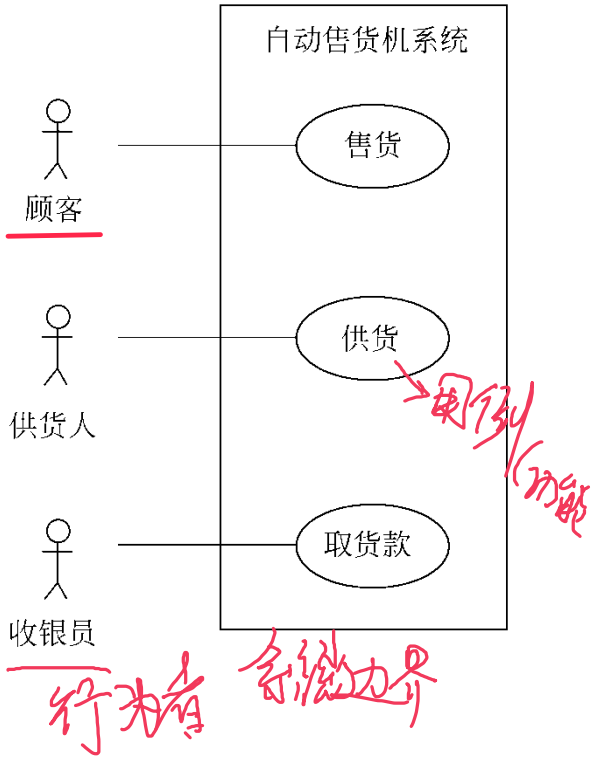
三、功能模型

1. 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：

（1）系统

系统被看作是一个提供用例的黑盒子。代表系统的方框的边线表示系统的边界，用于划定系统的功能范围，定义了系统所具有的功能。描述该系统功能的用例置于方框内，代表外部实体的行为者置于方框外。



（2）用例

- 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。
- 用例通过关联与行为者连接，关联指出一个用例与哪些行为者交互，这种交互是双向的。
- 用例具有下述特征：

① 用例代表某些用户可见的功能，实现一个具体的用户目标；

② 用例总是被行为者启动的，并向行为者提供可识别的值；

③ 用例必须是完整的。

注意，用例是一个类，它代表一类功能而不是使用该功能的某个具体实例。用例的实例是系统的一次具体执行过程，通常把用例的实例称为脚本。

（3）行为者

- 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，而不是某个具体的人或物。事实上，一个具体的人可以充当多种不同角色。

（4）用例之间的关系

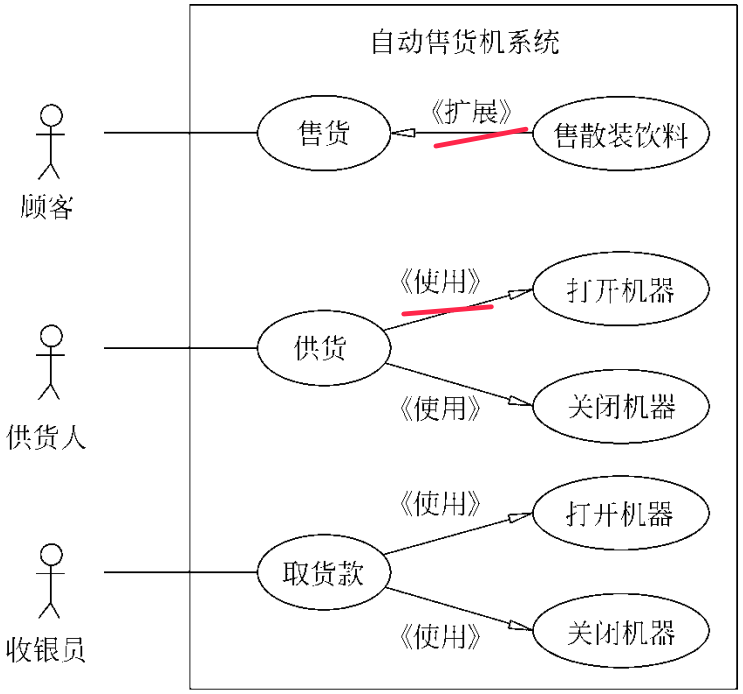
UML用例之间主要有扩展和使用两种关系，它们是泛化关系的两种不同形式：

① 扩展关系：向一个用例中添加一些动作后构成了另一个用例，这两个用例之间的关系就是扩展关系，后者继承前者的一些行为，通常把后者称为扩展用例。

② 使用关系：当一个用例使用另一个用例时，这两个用例之间就构成了使用关系。

例如：在自动售货机系统中，“售货”是一个基本的用例，如果顾客购买罐装饮料，售货功能完成得很顺利，但是，如果顾客要购买用纸杯装的散装饮料，则不能执行该用例提供的常规动作，而要做些改动。

- 可以修改售货用例，使之既能提供售罐装饮料的常规动作，又能提供售散装饮料的非常规动作。
- 把常规动作放在“售货”用例中，而把非常规动作放置于“售散装饮料”用例中，这两个用例之间的关系就是扩展关系。在用例图中，用例之间的扩展关系图示为带版类《扩展》的泛化关系。



扩展与使用之间的异同：

- 这两种关系都意味着从几个用例中抽取那些公共的行为并放入一个单独的用例中，而这个用例被其他用例使用或扩展，但是，使用和扩展的目的是不同的。
- 通常在描述一般行为的变化时采用扩展关系；在两个或多个用例中出现重复描述又想避免这种重复时，可以采用使用关系。

三、功能模型

2. 用例建模

一个用例模型由若干幅用例图组成。创建用例模型的工作包括：定义系统，寻找行为者和用例，描述用例，定义用例之间的关系，确认模型。其中，寻找行为者和用例是关键。

（1）寻找行为者

下述问题有助于发现行为者：

- 谁将使用系统的主要功能（主行为者）？
- 谁需要借助系统的支持来完成日常工作？
- 谁来维护和管理系统（副行为者）？
- 系统控制哪些硬件设备？
- 系统需要与哪些其他系统交互？
- 哪些人或系统对本系统产生的结果（值）感兴趣？

（2）寻找用例

可以通过请每个行为者回答下述问题来获取用例：

- 行为者需要系统提供哪些功能？行为者自身需要做什么？
- 行为者是否需要读取、创建、删除、修改或存储系统中的某类信息？
- 系统中发生的事件需要通知行为者吗？行为者需要通知系统某些事情吗？从功能观点看，这些事件能做什么？
- 行为者的日常工作是否因为系统的新功能而被简化或提高了效率？

还有一些不是针对具体行为者而是针对整个系统的问题，也能帮助建模者发现用例，例如：

- 系统需要哪些输入输出？输入来自何处？输出到哪里去？
- 当前使用的系统（可能是人工系统）存在的主要问题是什么？

注意：最后这两个问题并不意味着没有行为者也可以有用例，只是在获取用例时还不知道行为者是谁。事实上，一个用例必须至少与一个行为者相关联。


Thank
You