用例图主要用来**描述“用户、需求、系统功能单元”**之间的关系。它展示了一个**外部用户能够观察到的系统功能模型**图。用例就是软件的功能模块用例名一般为动宾短语。

二.用例图包含的元素

1.参与者(Actor)：表示与您的应用程序或系统进行交互的用户、组织或外部系统。用一个小人表示。2.用例(Use Case)：用例就是外部可见的系统功能，对系统提供的服务进行描述。用椭圆表示。3. 容器代表着一个系统

用例图中涉及的关系有：**关联、泛化、包含、扩展。**

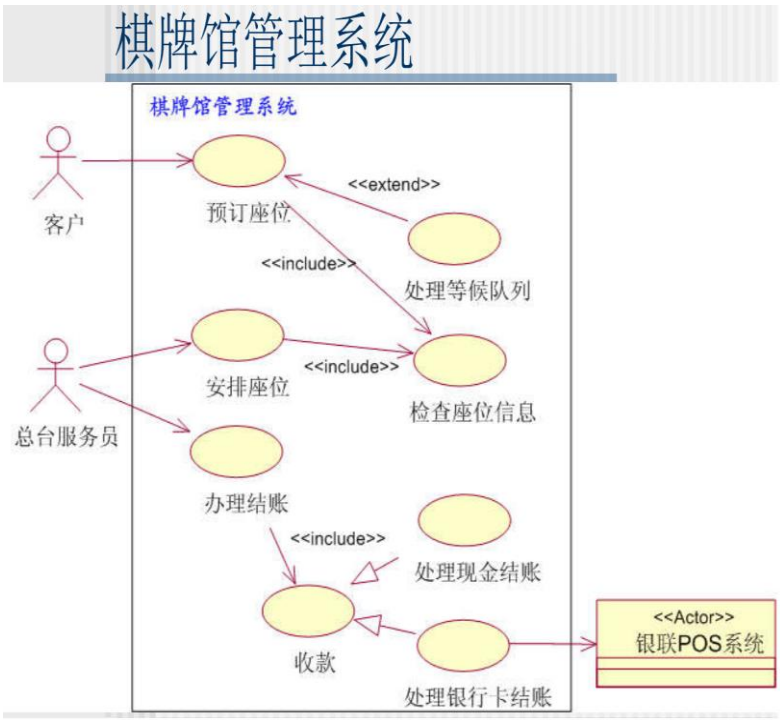
关系类型	说明	表示符号
关联	参与者与用例间的关系	——→
泛化	参与者之间或用例之间的关系	——▷
包含	用例之间的关系	«包括»⇢
扩展	用例之间的关系	«扩展»⇢

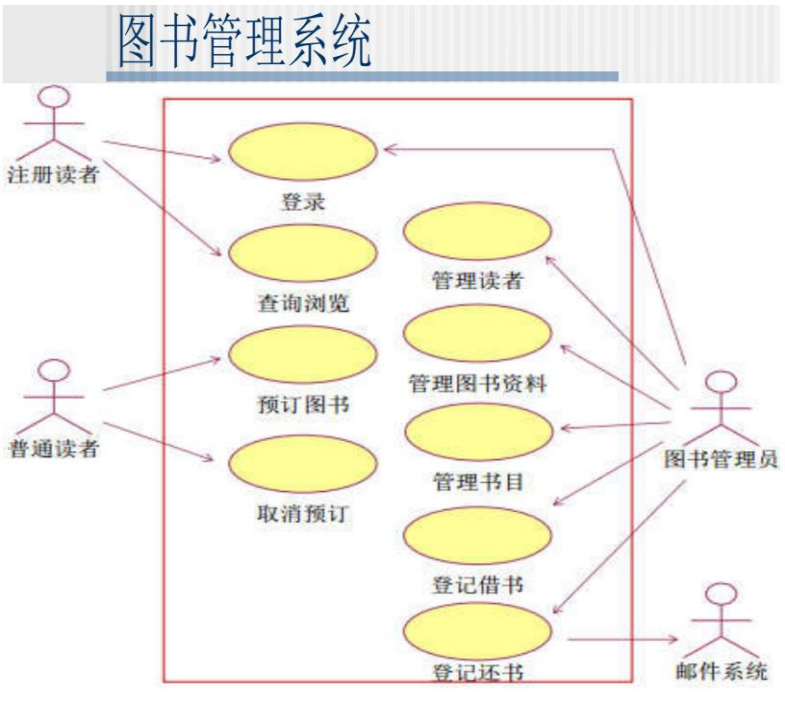
a. **关联(Association)**表示参与者与用例之间的通信，任何一方都可发送或接受消息。【箭头指向】：指向消息接收方

b. **泛化(Inheritance)**：就是通常理解的**继承关系，子用例和父用例相似**，但表现出**更特别的行为；子用例将继承父用例的所有结构、行为和关系**。子用例可以使用父用例的一段行为，也可以**重载它**。父用例通常是**抽象的**。【箭头指向】：指向父用例

c. **包含(Include)**包含关系用来把一个较复杂用例所表示的功能分解成较小的步骤。【箭头指向】：指向分解出来的功能用例

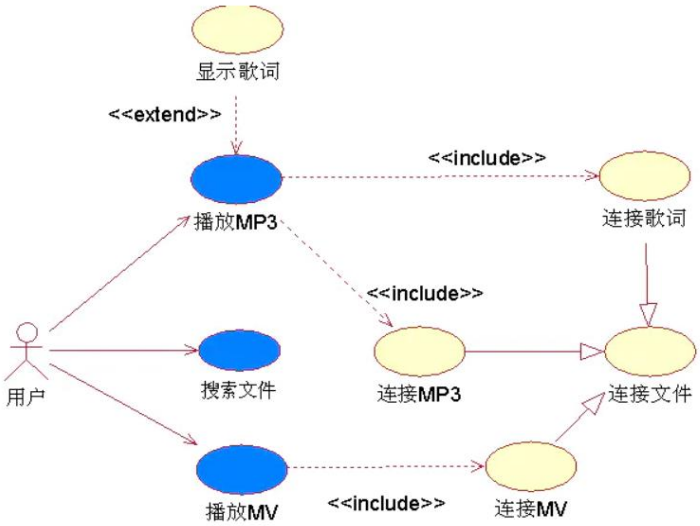
d. **扩展(Extend)**扩展关系是指用例功能的延伸，相当于为基础用例提供一个附加功能。【箭头指向】：指向基础用例

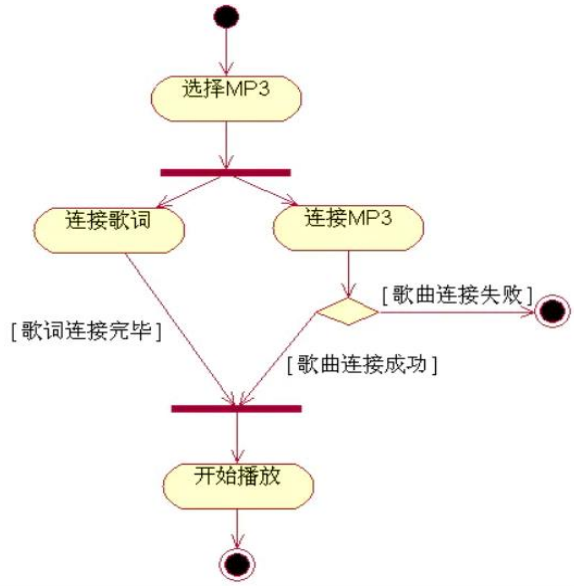




活动图本质上是一种流程图，它描述活动的序列，即系统从一个活动到另一个活动的控制流。活动图聚焦一个用例。

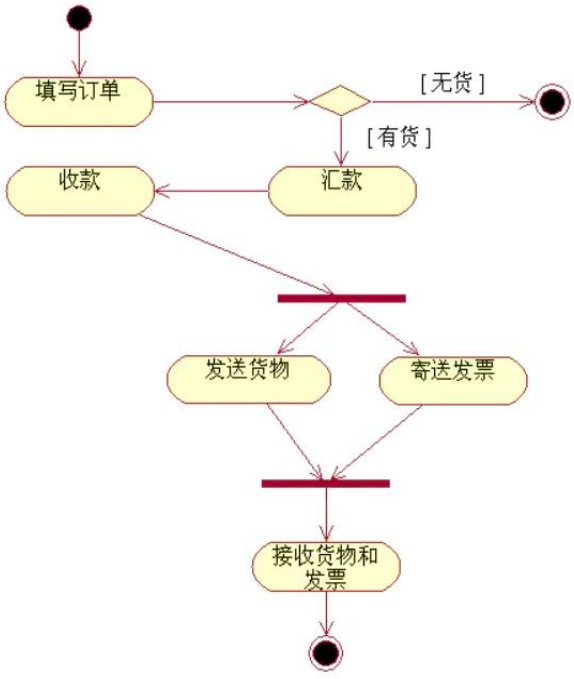
场景描述：假如现在有一个简单的音乐盒，用户根据歌曲名称搜索自己喜爱的音乐，找到后，用户可以以 MP3 模式播放（同步显示歌词），也可以以 MV 模式播放。由于音乐和歌词都来自于互联网，因此在播放之前应连接这些文件。**描述 MP3 播放功能这个用例的活动图。**

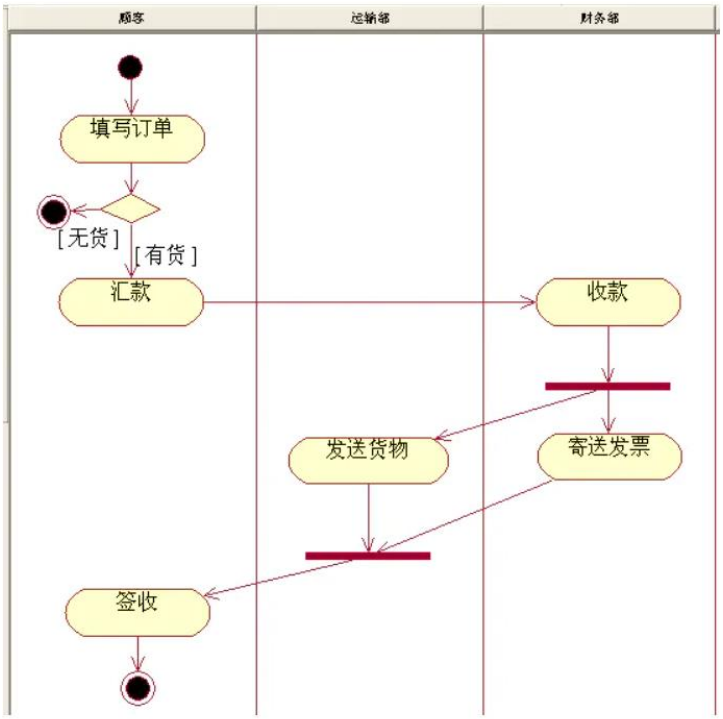


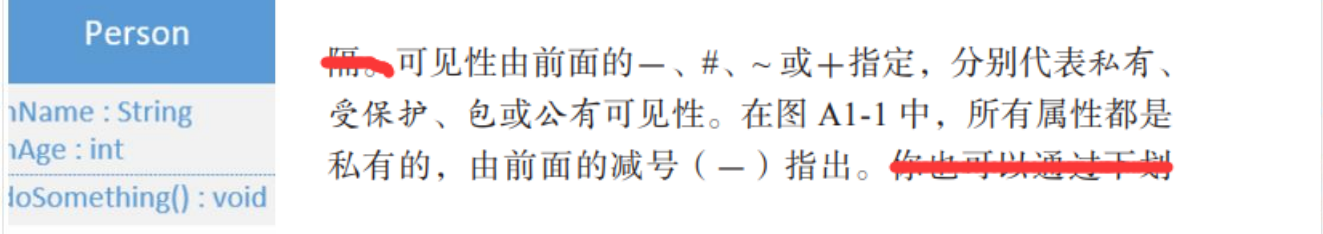


泳道表明每个活动是由哪些人或哪些部门负责完成。在包含泳道的活动图中每个活动只能明确地属于一个泳道。

简单模拟卓越网的购物流程







类与类（接口）间共有**六种**关系。泛化（Generalization）、实现（Realization）、关联（Association）（又分一般关联、聚合（Aggregation）、组合（Composition））、依赖（Dependency）

1.泛化关系：关系表示父类和子类间的关系，它的符号（空心三角形 + 实线）为：由子类指向父类，即子类 -> 父类

实现关系：接口和类之间存在一种实现关系，在这种关系中，类实现了接口，类中的操作实现了接口中所声明的操作。UML 中用带空心三角形的虚线来表示

2.关联(Association)关系

A、 一般关联

a、 关联关系是**类与类之间的联结**，它使一个类知道另一个类的属性和方法，指明了事物的对象之间的联系，如：老师与学生、丈夫与妻子。关联可以是双向的，也可以是单向的，还有自身关联。

b、 用**带普通箭头的实心线**表示。双向的关联可以有两个箭头或者没有箭头，单向的关联有一个箭头，如下图：

B、 聚合（Aggregation）

a、 它是整体与部分（整体 has a 部分）的关系，**且部分可以离开整体而单独存在，如车和轮胎是整体和部分的关系，轮胎离开车仍然可以存在。**聚合关系是关联关系的一种，是强的关联关系，**关联和聚合在语法上无法区分，必须考察具体的逻辑关系。**

b、 用带空心菱形的实线**表示，菱形指向整体，如下图：**

C、 组合（Composition）

a、 它是整体与部分的关系，但部分不能离开整体而单独存在。如公司和部门是整体和部分的关系，没有公司就不存在部门。组合关系是关联关系的一种，是比聚合关系还要强的关系，它要求普通的聚合关系中代表整体的对象负责代表部分的对象的生命周期。

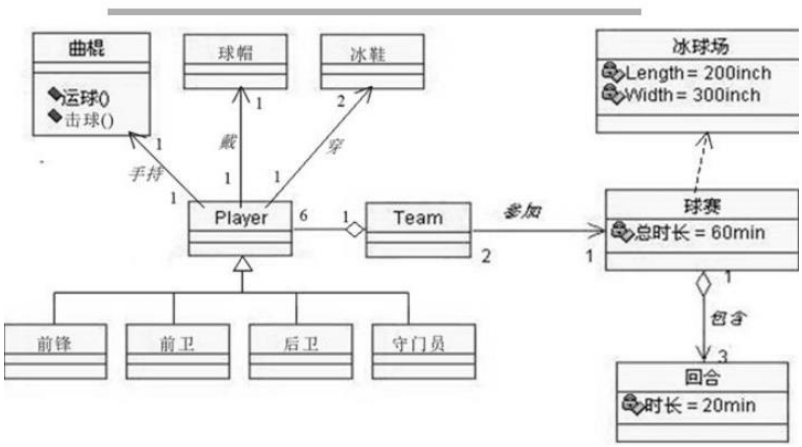
b、 用带**实心菱形的实线**表示，菱形指向整体，如下图：

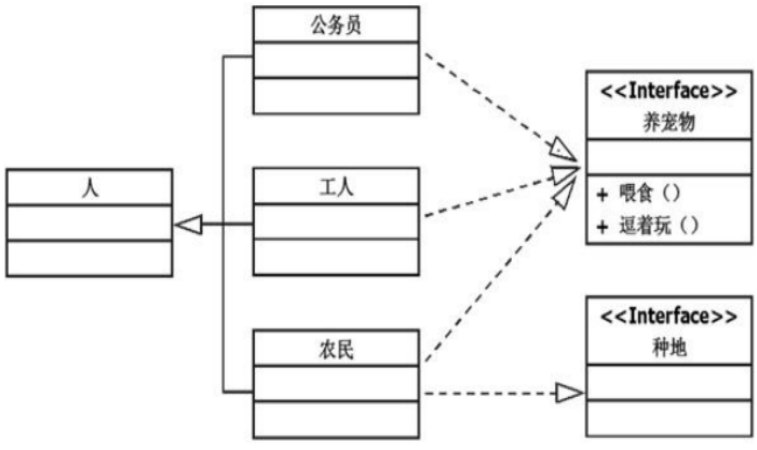
3、 依赖（Dependency）

表示一个事物使用另一个事物时的依赖关系。

大多数情况下，体现在某个类的方法使用另一个类的对象作为参数(或返回值，或局部变量)。

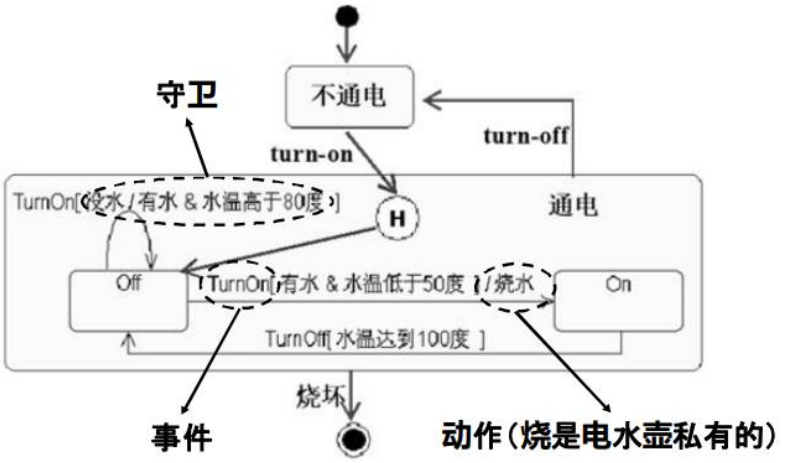
UML 图中用带箭头的虚线表示，由依赖的一方指向被依赖的一方。

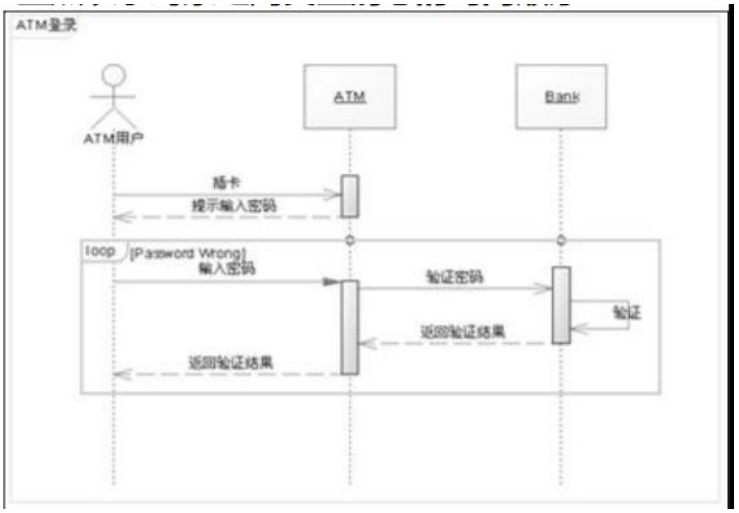




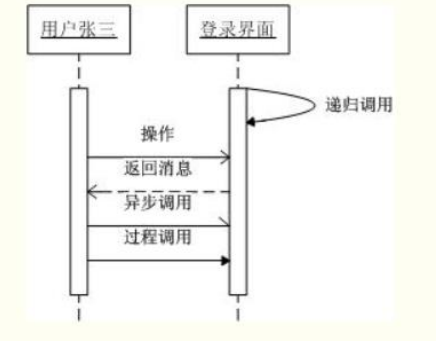
围描述。通过“0..1”描述的多重性是指在关联的一端存在 0 或 1 个对象。例如，世界上的每个人要么有社会保险号，要么没有，因此，多重性 0..1 就可以在类图中的 Person 类和 SocialSecurityNumber 类之间的关联中使用。由“1..*”描述的多重性是指一个或更多，由“0..*”或只是“*”描述的多重性是指 0 个或更多。在图 A1-2 中，与 Person 类关联的

状态图主要用于描述**一个对象**在其生存期间的动态行为，表现为一个对象所经历的状态序列，引起状态转移的事件（Event），以及因状态转移而伴随的动作（Action），守卫是判定条件。**（事件，守卫，动作）**



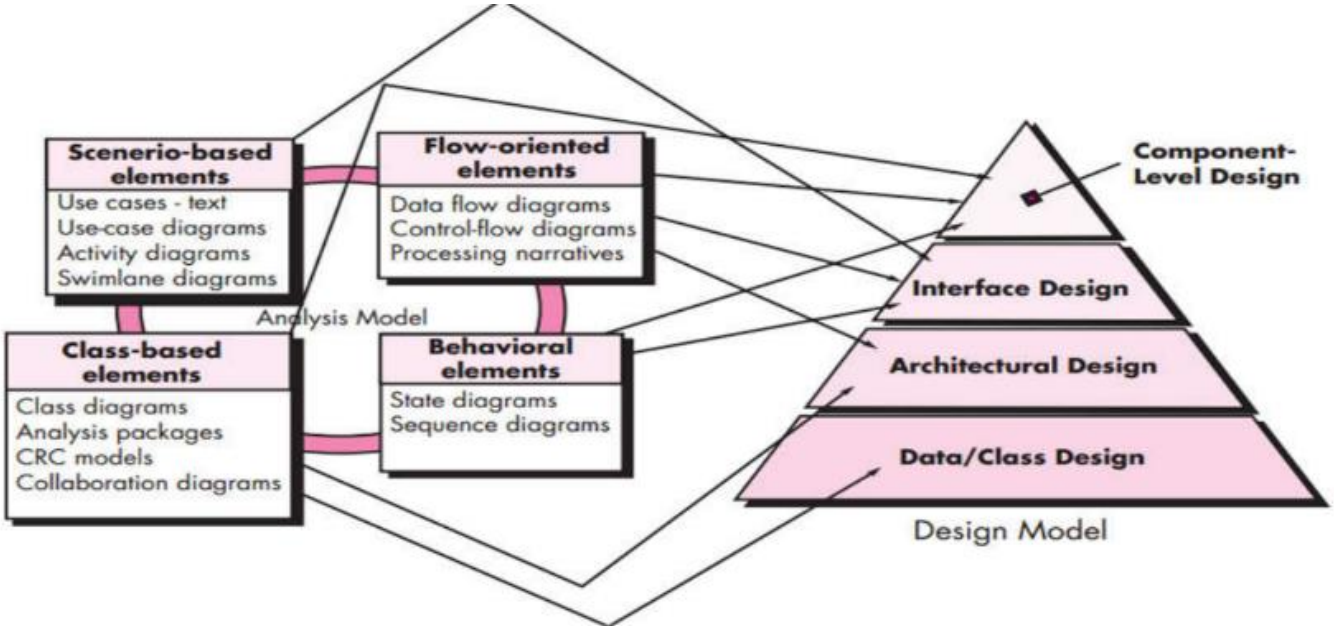


序列图，它按时间顺序（纵向上）显示**多个对象之间**的交互。
时序图中关注生命线之间的通信，这些通信就是对象发送的消息。消息分为 5 类：递归调用、普通操作、返回消息、异步调用的消息、同步调用的消息。同步（过程）消息：意味着阻塞和等待。异步消息：就意味着是非阻塞。如：A 向 B 发送消息后，直接可以执行下面代码，无需等待 B 的执行。
同步消息用实心箭头表示，异步消息用开放式箭头表示。



图顶端行中的每个方框通常对应一个对象，虽然方框也有可能模拟其他东西，比如类。如果方框表示对象（如我们所有例子中的情况），那么在方框内， 我们可以有选择地规定冒号前的对象名。也可在冒号之后写上类名

状态图和顺序图的区别: In the context of behavioral modeling, two different characterizations of states must be considered :
(1) the state of each class as the system performs its function
·状态图: how an individual class changes state based on external events
(2) the state of the system as observed from the outside as the system performs its function
·时序图: shows the behavior of the software as a function of time.
分析模型和设计模型总结

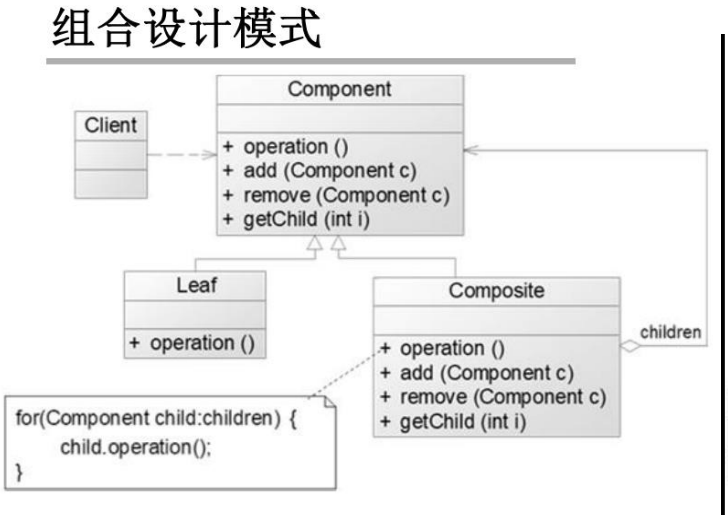


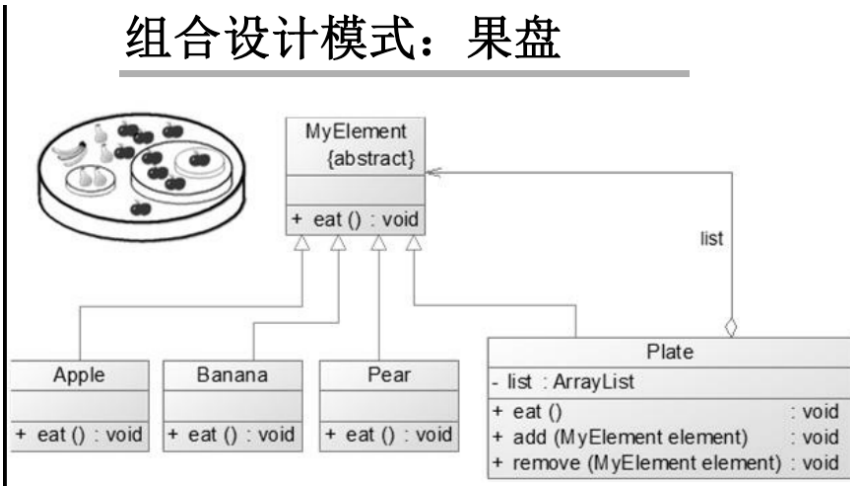
11.3.3（设计）模式/ 12.3.2 体系结构模式

组合设计模式：（Composite Design Pattern）将一组对象组织(Compose)成树形结构，以表示一种部分-整体的层次结构。组合让客户端可以统一单个对象和组合对象的处理逻辑。

优点：调用者调用简单，不管针对叶子节点还是树枝节点，调用者的调用逻辑是一样的方便扩展，无论是新增叶子节点还是树枝节点，都可以很方便的扩展

组合模式的实现方式主要分为三步：定义一个接口，并在接口中定义要实现的功能；叶子节点（Leaf）实现这个接口，并重写接口中的方法；树枝节点中有一个集合或者数组，可以对接口对象进行管理。同时，树枝节点（Composite）还要实现这个接口，在重写接口的方法时可以循环集合或数组得到接口对象，并对其进行调用





11.3.13 依赖倒置原则的核心思想是面向接口编程（一个函数的形参为抽象类）

13.2.1 构件级的设计思想（面向对象的 7 大设计原则）

1.开闭原则 OCP（多态）

我们需要对一个类的功能进行扩展、增加方法的时候，不用对原本的类进行修改，而是通过继承，去重写，将父类原本方法的行为改造为我们现在阶段需要的行为；也可以通过实现和父类一样的借口来完成一样的效果。造成的结果就是，我们没有去修改原来的类(对修改关闭)，却实现了功能的扩展(对扩展开放)。

2.里氏替换原则

思想：子类型（subtype）必须能够替换掉他们的基类型（base type）。在不了解派生类的情况下,仅通过接口或基类的方法,即可清楚的知道方法的行为,而不管哪种派生类的实现，都与接口或基类方法的期望行为一致。或者说接口或基类的方法是一种契约，使用方按照这个契约来使用，派生类也按照这个契约来实现。这就是与期望行为一致的替换。

3.依赖倒置原则的核心思想是面向接口编程（一个函数的形参为抽象类） **4.接口分离原则**

内聚和耦合: 一个模块应当尽可能独立完成某个功能，模块内部的元素关联性越强，则内聚越高，模块单一性就越强；

如果模块之间存在依赖，则可能会导致一个模块的改动影响了另外的模块的问题，甚至是相互影响。两个模块之间的关系越紧密，耦合就越强，模块的独立性就会越差。

面向对象思想

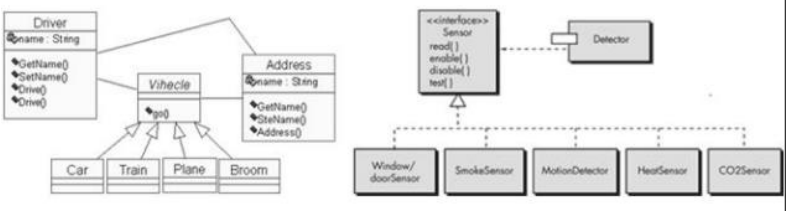
重载 overloaded 和多态 polymorphism

- 函数重载：同一个作用域中，若干个参数特征不同的函数，使用相同的函数名
- 运算符重载：同一个运算符可以施加于不同类型的操作数
- 重载机制
  - 静态联编（static binding）
  - 在程序编译时，根据函数变元的个数和类型，决定使用同名函数的哪个实现代码
  - 提高OO程序灵活性和可读性

多态

- 通过重写(override)父类的同名操作，使不同的子类对象和父类对象接受同一消息，却提供不同服务
- 不同层次的类共享一个行为(函数名、参数和返回值类型都相同)，但行为实现却不同
- 多态机制
  - 动态联编(dynamic binding)，也称滞后联编(late binging)
  - 运行时，消息发送对象不知道谁将接收消息，依赖接收对象以一种恰当的方式解释消息，根据消息接收对象所属的类，决定实现什么样的行为，享受什么样的服务
  - 增强OO系统的灵活性，减少信息冗余，提高软件的可重用性和可扩充性

- 多态的发生有3个条件：
  - （1）继承或实现
  - （2）子类中重写父类（接口）的方法
  - （3）父类引用指向子类对象



OO基本概念

- 类、对象、实例
- 封装与信息隐藏
- 继承和类等级
- 消息、协议和服务
- 重载
- 多态