UML 关系(泛化,实现,依赖,关联(聚合,组合))

UML 的构造快包含 3 种：

(1) 事物（4 种）：结构事物，行为事物，分组事物，注释事物

(2) 关系（4 种）：泛化关系，实现关系，依赖关系，关联关系

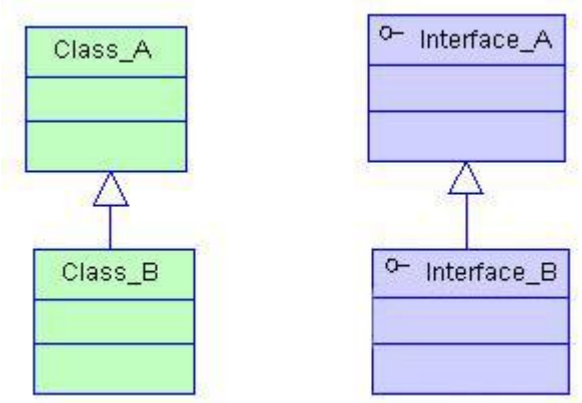
(3) 图（10 种）：用例图，类图，对象图，包图，组件图，部署图，状态图，活动图，序列图，协作图

事物是对模型中最具代表性的成分的抽象；关系把事物结合在一起；图聚集了相关的事物。

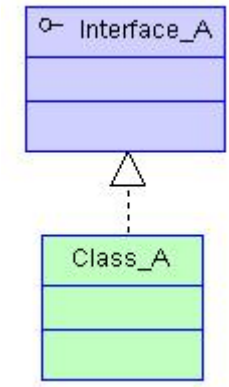
(2) 关系（4 种）

UML 中类与类, 类与接口, 接口与接口这间的关系有: 泛化(generalization) 关系, 关联(association)关系(关联, 聚合, 合成), 依赖(dependency)关系，实现(realization)关系.

泛化(generalization)关系是一个类（称为子类、子接口）继承另外的一个类（称为父类、父接口）的功能，并可以增加它自己的新功能的能力，继承是类与类或者接口与接口之间最常见的关系；在 Java 中此类关系通过关键字 extends 明确标识，在设计时一般没有争议性。

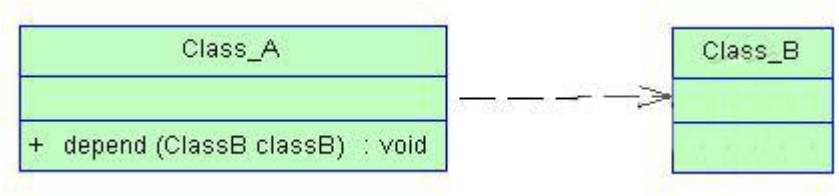


实现(realization)关系指的是一个 class 类实现 interface 接口（可以是多个）的功能；实现是类与接口之间最常见的关系；在 Java 中此类关系通过关键字 implements 明确标识，在设计时一般没有争议性；



依赖(dependency)关系: 也是类与类之间的连接. 表示一个类依赖于另一个类的定义. 依赖关系总是单向的 。可以简单的理解，就是一个类 A 使用到了另一个类 B，而这种使用关系是具有偶然性的、、临时性的、非常弱的，但是 B 类的变化会影响到 A；比如某人要过河，需要借用一条船，此时人与船之间的关系就是依赖；表现在代码层面，为类 B 作为参数被类 A 在某个 method 方法中使用。

在 java 中. 依赖关系体现为: 局部变量, 方法中的参数, 和对静态方法的调用.



关联(association)关系: 表示类与类之间的联接, 它使一个类知道另一个类的属性和方法.

关联可以使用单箭头表示单向关联, 使用双箭头或不使用箭头表示双向关联, 不建议使用双向关联. 关联有两个端点, 在每个端点可以有一个基数, 表示这个关联的类可以有几个实例. 常见的基数及含义:

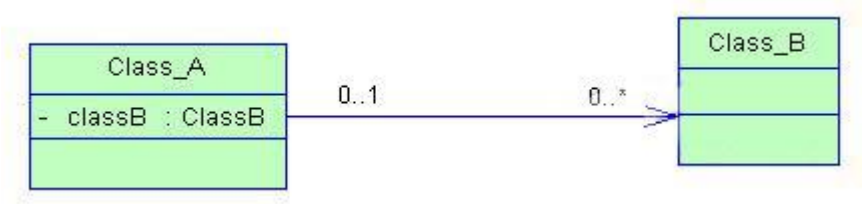
0..1:0 或 1 个实例.

0..*: 对实例的数目没有限制.

1: 只能有一个实例.

1..*: 至少有一个实例.

他体现的是两个类、或者类与接口之间语义级别的一种强依赖关系，比如我和我的朋友；这种关系比依赖更强、不存在依赖关系的偶然性、关系也不是临时性的，一般是长期性的，而且双方的关系一般是平等的,表现在代码层面，为被关联类 B 以类属性的形式出现在关联类 A 中，也可能是关联类 A 引用了一个类型为被关联类 B 的全局变量；在 java 语言中关联关系是使用实例变量实现的.



聚合(aggregation)关系: 关联关系的一种特例, 是强的关联关系. 聚合是整体和个体之间的关系,即 has-a 的关系，此时整体与部分之间是可分离的，他们可以具有各自的生命周期，部分可以属于多个整体对象，也可以为多个整体对象共享；比如计算机与 CPU、公司与员工的关系等；表现在代码层面，和关联关系是一致的，只能从语义级别来区分；

聚合关系也是使用实例变量实现的. 从 java 语法上是分不出关联和聚合的.

关联关系中两个类是处于相同的层次, 而聚合关系中两不类是处于不平等的层次, 一个表示整体, 一个表示部分.

组合(合成)关系(composition): 也是关联关系的一种特例，他体现的是一种 contains-a 的关系，这种关系比聚合更强，也称为强聚合；他同样体现整体与部分间的关系，但此时整体与部分是不可分的，整体的生命周期结束也就意味着部分的生命周期结束；比如你和你的大脑；合成关系不能共享. 。表现在代码层面，和关联关系是一致的，只能从语义级别来区分。

组合跟聚合几乎相同，唯一的区别就是“部分”不能脱离“整体”单独存在，就是说，“部分”的生命期不能比“整体”还要长。

总结：

对于继承、实现这两种关系没多少疑问，他们体现的是一种类与类、或者类与接口间的纵向关系；其他的四者关系则体现的是类与类、或者类与接口间的引用、横向关系，是比较难区分的，有很多事物间的关系要想准备定位是很难的，前面也提到，这几种关系都是语义级别的，所以从代码层面并不能完全区分各种关系；但总的来说，后几种关系所表现的强弱程度依次为：组合>聚合>关联>依赖。