**一、**UML**简介**

UML （ Unified Modeling Language ）即统一建模语言，是 OMG （ Object Management Group ）发表的图标式软件设计语言。

UML 的功能：

可视化：使用图表的形式来表现业务关系或者物理关系，可以促进对问题的理解和解决。

说明： UML 提供了一种通用的、精通的、没有歧义的通信机制进行。

建造： UML 通过自己的语法规则使得可以通过使用建模工具软件将设计模式映射到一种语言上。

建文档：使用 UML 进行设计可以同时产生系统设计文档。

UML 包括：

用例图（ Use case diagrams ）

类图（ Class diagrams ）

序列图（ Sequence diagrams ）

合作图（ Collaboration diagrams ）

状态图（ Statechart diagrams ）

活动图（ Activity diagrams ）

构件图（ Component diagrams ）

部署图（ Deployment diagrams ）

（蓝色为较重要并且常用的图）

类图是最常用的 UML 图，显示出类、接口以及它们之间的静态结构和关系；它用于描述系统的结构化设计。

类图最基本的元素是类或者接口。

**二、类图**

**1. 类（ Class ）**

一般包含 3 个组成部分。第一个是类名；第二个是属性（ attributes ）；第三个是该类提供的方法（ 类的性质可以放在第四部分；如果类中含有内部类，则会出现第五个组成部分）。类名部分是不能省略的，其他组成部分可以省略。

类名书写规范：正体字说明类是可被实例化的，斜体字说明类为抽象类。

属性和方法书写规范：修饰符 [ 描述信息 ] 属性、方法名称 [ 参数 ] [ ：返回类型 | 类型 ]

属性和方法之前可附加的可见性修饰符：

加号（ + ）表示 public ；减号（ - ）表示 private ； # 号表示 protected ；省略这些修饰符表示具有 package （包）级别的可见性。

如果属性或方法具有下划线，则说明它是静态的。

描述信息使用 << 开头和使用 >> 结尾。

类的性质是由一个属性、一个赋值方法和一个取值方法组成。书写方式和方法类似。

例 1 ：

public class TaxCalculator

{

private long taxRate ;

private int salary ;

public TaxCalculator(long taxRate)

{

this.taxRate = taxRate ;

}

public long countTax()

{

return taxRate\*salary;

}

public int getSalary()

{

return salary;

}

public void setSalary(int salary)

{

this.salary = salary;

}

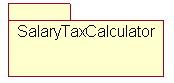
}

|  |
| --- |
| TaxCalculator |
| -taxRate:long |
| +TaxCalculator(a:long)  +countTax():long |
| salary:long |

**2. 包（ Package ）**

包是一种常规用途的组合机制。 UML 中的一个包直接对应于 Java 中的一个包。在 Java 中，一个包可能含有其他包、类或者同时含有这两者。进行建模时，通常使用逻辑性的包，用于对模型进行组织；使用物理性的包，用于转换成系统中的 Java 包。每个包的名称对这个包进行了惟一性的标识。

例：



**3.** **接口（ Interface ）**

接口是一系列操作的集合，它指定了一个类所提供的服务。它直接对应于 Java 中的一个接口类型。接口的表示有大概两种方式。具体画法见下例：

例：

public interface TaxCalculator

{

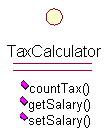
public long countTax();

public int getSalary();

public void setSalary(int salary);

}

画法一：



画法二:

画法二 :

画法二:

|  |
| --- |
| Interface  *TaxCalculator* |
| +countTax():long  +getSalary():int  +setSalary() |

**4. 关系**

常见的关系有：一般化关系、实现关系、聚合关系、合成关系和依赖关系。请注意不同关系在图中连线表示的不同。

**4.1 一般化关系（ Generalization ）**

在有的书籍和资料中也叫做“泛化关系”。

一般化关系表示类与类、接口与接口之间的继承关系。关系中的箭头由子类指向父类。在 Java 中，用 ***extends***关键字来直接表示这种关系。

例：

public abstract class Employee{

}

public class Programmer extends Employee{

}



**4.2 实现（ Realization ）**

实例关系指定两个实体之间的一个合同。换言之，一个实体定义一个合同，而另一个实体保证履行该合同。关系中的箭头由实现接口的类指向被实现的接口。在 Java 中，实现关系可直接用 ***implements***关键字来表示。

例：

public interface CollegePerson{

}

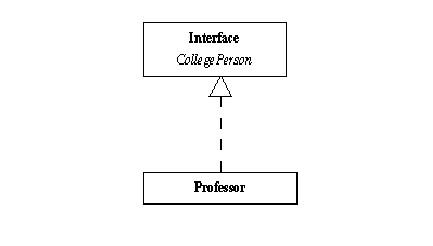
public class Professor implements CollegePerson{

}

表示方法一：（在 Rose2003 下实现如下，连接线没有箭头，因为方向显而易见）



表示方法二：



**4.3 关联（ Association ）**

表示类与类之间的连接。它使一个类的可见属性和方法被另一个类使用。关联可以是双向或者单向的。双向的关联箭头是可选的，单向的箭头指向遍历或者查询的方向。在 Java 中，关联使用实例变量来实现。在关联关系中可以使用附加的基数来说明类之间对应的个数。

|  |  |
| --- | --- |
| 基数 | 含义 |
| 0..1 | 零个或者一个实例 |
| 0..\* 或者 \* | 没有限制，任意 |
| 1 | 有且只能一个实例 |
| 1..\* | 至少有一个实例 |

常见的基数

例：以权限中的用户组、用户角色为例，一个用户角色可以属于一个或者多个用户组，一个用户组则可以包含多个用户角色。如果用户组采用下面的方法来获得用户角色的权限 …

public class UserGroup{

private UserRole uRole ;

......

}

public class UserRole{

}

则表现在图中为：（注意箭头的画法）



注意：一个关联关系往往是聚合关系或者是合成关系。

注意：一个关联关系往往是聚合关系或者是合成关系。

**4.4 聚合（ Aggregation ）**

聚合是关联的一种形式，代表两个类之间的整体 / 局部关系。聚合暗示着整体在概念上处于比局部更高的一个级别，而关联暗示两个类在概念上位于相同的级别。在 Java 中，聚合也是使用实例变量来实现的。   
关联和聚合的区别纯粹是概念上的，在 Java 语法上分辨不出来的。聚合还暗示着实例图中不存在回路。换言之，只能是一种单向关系。

例：汽车与轮胎的关系可以很好的说问题。

public class Car{

private Tyres tyres;

}

public class Tyres{

}



**4.5 合成（ Composition ）:[可以认为关系更强烈]**

合成是聚合的一种特殊形式，暗示 “ 局部 ” 在 “ 整体 ” 内部的生存期职责。合成关系是不能共享的。所以，虽然局部不一定要随整体的销毁而被销毁，但整体要么负责保持局部的存活状态，要么负责将其销毁。局部不可与其他整体共享。但是，整体可将所有权转交给另一个对象，后者随即将承担生存期职责。

例：人和他的腿就是一个好的例子。

public class Man{

private Legs legs;

}

public class Legs{

}



**4.6 依赖（ Dependency ）**

依赖也是类与类之间的连接，并且依赖总是单向的。实体之间一个 “ 使用 ” 关系暗示一个实体的规范发生变化后，可能影响依赖于它的其他实例。更具体地说，它可转换为对不在实例作用域内的一个类或对象的任何类型的引用。其中包括一个局部变量，对通过方法调用而获得的一个对象的引用（如下例所示），或者对一个类的静态方法的引用（同时不存在那个类的一个实例）。也可利用 “ 依赖 ” 来表示包和包之间的关系。由于包中含有类，所以你可根据那些包中的各个类之间的关系，表示出包和包的关系。

例：给一个雇员计算薪水的时候 , 要使用计算器的例子

public class Employee{

public void calcSalary(Calculator cSalary)

{

}

}

