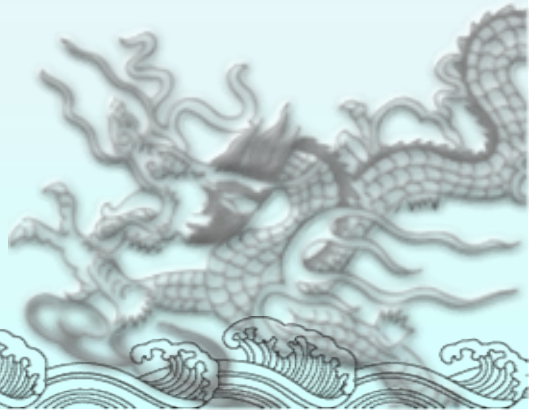
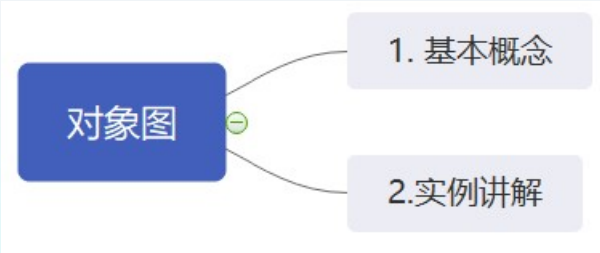
类图和对象图

内容提纲

- 类图
  - 相关概念
  - 类间关系
  - 代码映射
  - 实例
- 对象图
  - 相关概念
  - 实例

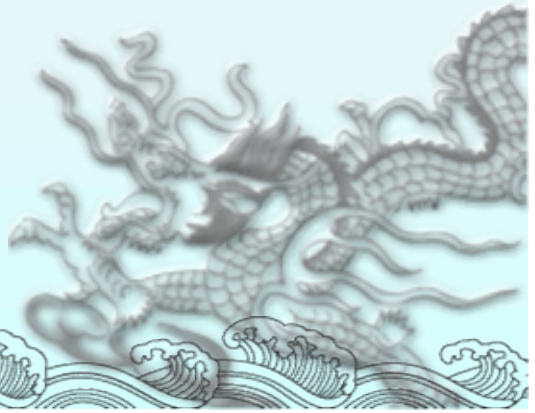


思维导图



内容提纲

- 类图
  - 相关概念
  - 类间关系
  - 代码映射
  - 实例讲解
- 对象图
  - 相关概念
  - 实例讲解



类图-相关概念

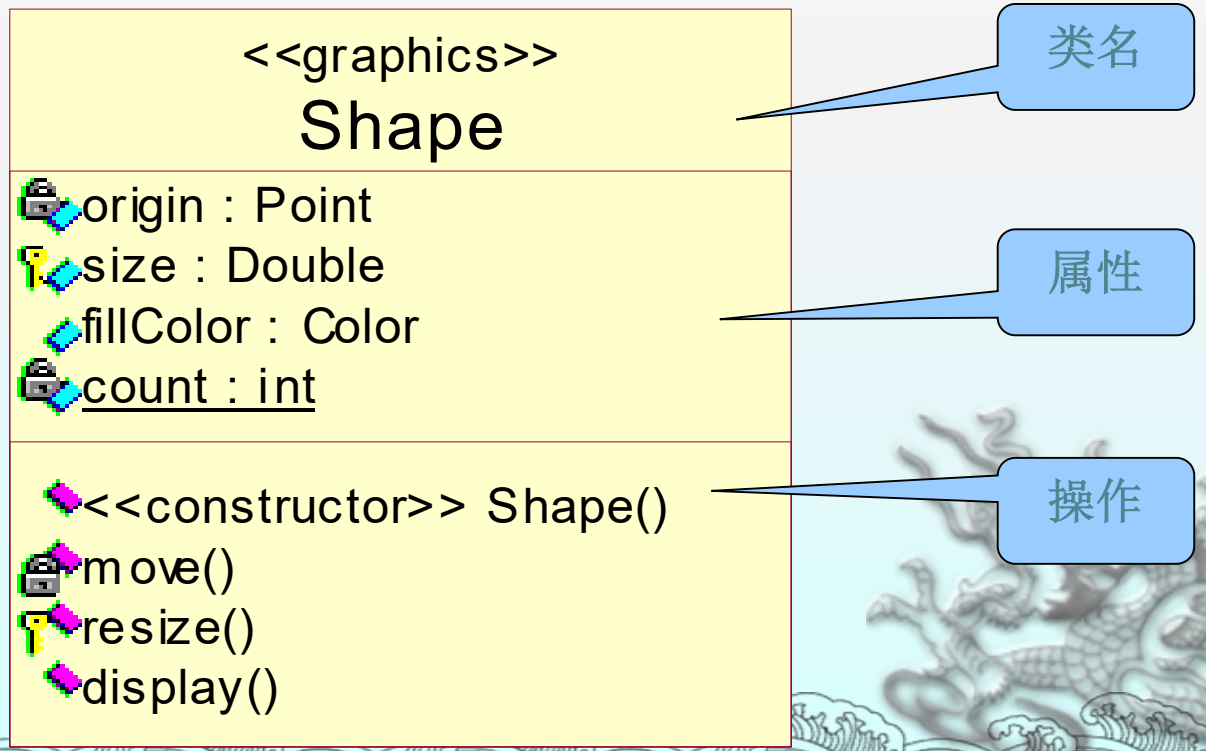
- 类图表达一组类和它们之间的联系。
  - 描述各个类本身的组成，即类的属性、操作和对对象的约束条件等。
- 类图是一种静态结构图，它描述的是系统的静态结构，而不是系统的行为。
- 类之间的静态联系主要类型有：关联、聚合、组合、泛化、依赖、实现等

类图-相关概念

什么是类（class）？

具有相似结构、行为和关系的一组对象的描述符

类的组成：UML中，类表示为划分为三格的矩形：

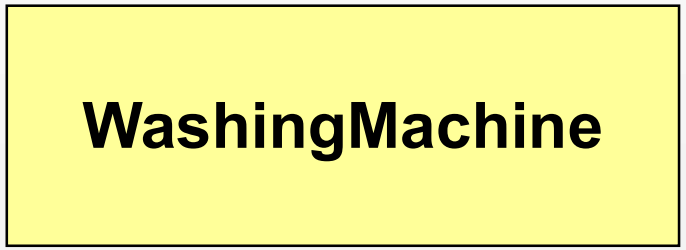


类图-相关概念

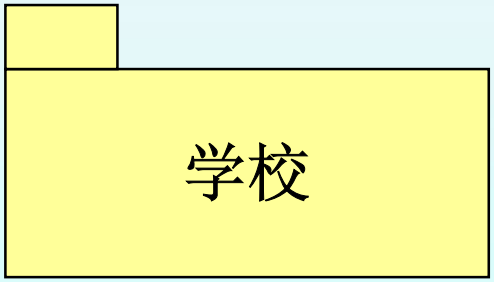
类名：

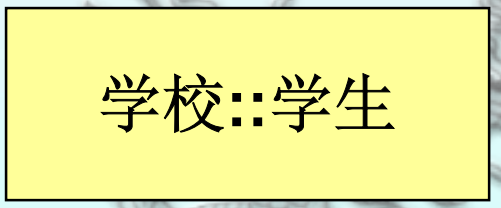
类名必须唯一，可以是简单名，也可以是路径名。

如果用英文，则单词首字母大写，多单词合并写。



路径名：包名在左，类名在右，中间用双冒号隔开。





类图-相关概念

属性是类的一个特性。

一个类可具有零个到多个属性；属性名必须唯一。

属性的格式

- [可见性] 属性名 [:类型]['['多重性[次序]']'][=缺省值][{特性}]

  可见性：可访问性

  多重性：属性值个数格式

  次序：属性值顺序

  特性：对该属性性质的一个约束说明，如{只读}

- 选取类的属性时只考虑系统用到的特征，不必将所有属性都表示出来，原则上，由类的属性应能区分每个特定的对象。

类图-相关概念

可见性

- 属性的可访问性，四类：
  - 公共（public）
  - 私有（private）
  - 保护（protected）
  - 实现（implementation）

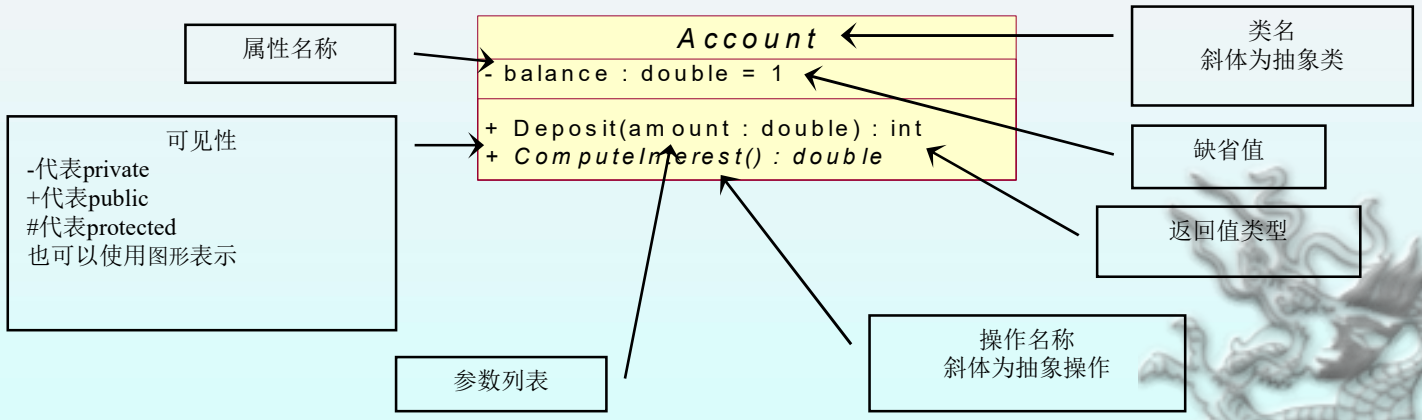
- 子类无法继承和访问父类的私有属性和实现属性

类图-相关概念

类的操作：

用于修改、检索类的属性或执行某些动作，通常也称为功能。

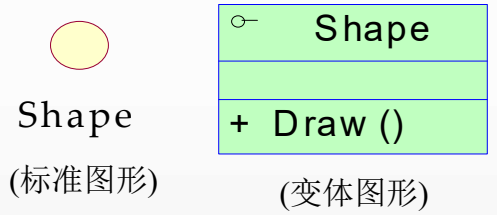
格式：[可见性]操作名[(参数列表)][：返回值类型][{特性}]
参数列表：参数名1：类型=缺省值，参数名2：类型=缺省值，…



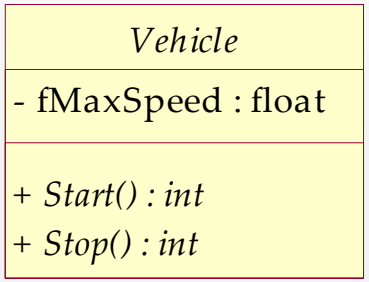
类图-相关概念

类图中的事物及解释：

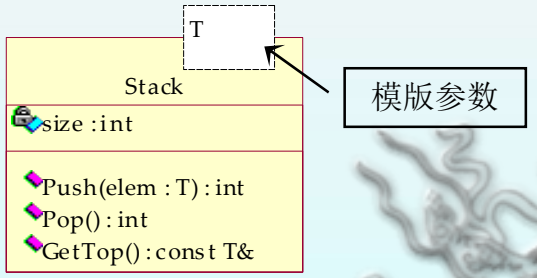
- 接口：一组操作的集合，只有操作的声明而没有实现
- 抽象类：不能被实例化的类，一般至少包含一个抽象操作
- 模版类：一种参数化的类，在编译时把模版参数绑定到不同的数据类型，从而产生不同的类



接口



抽象类

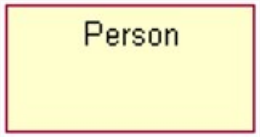


模版类

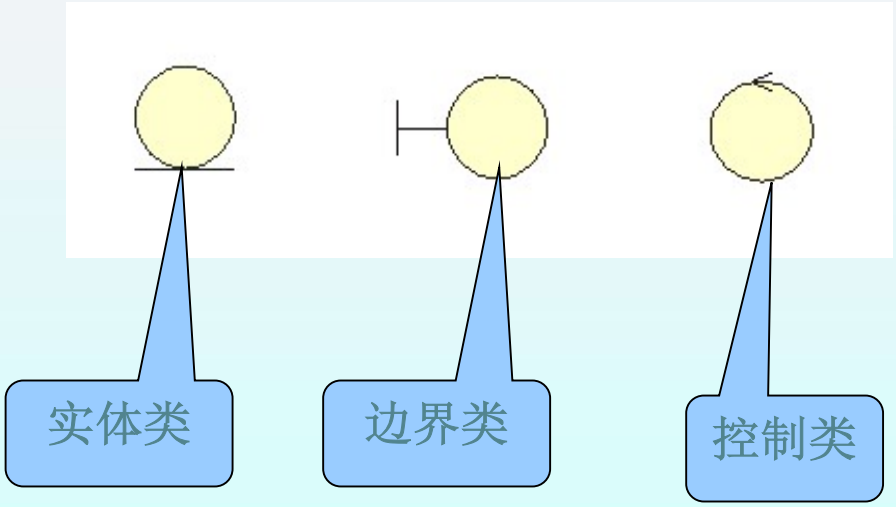
类图-相关概念

类的其他几种表示形式：

① 简化表示



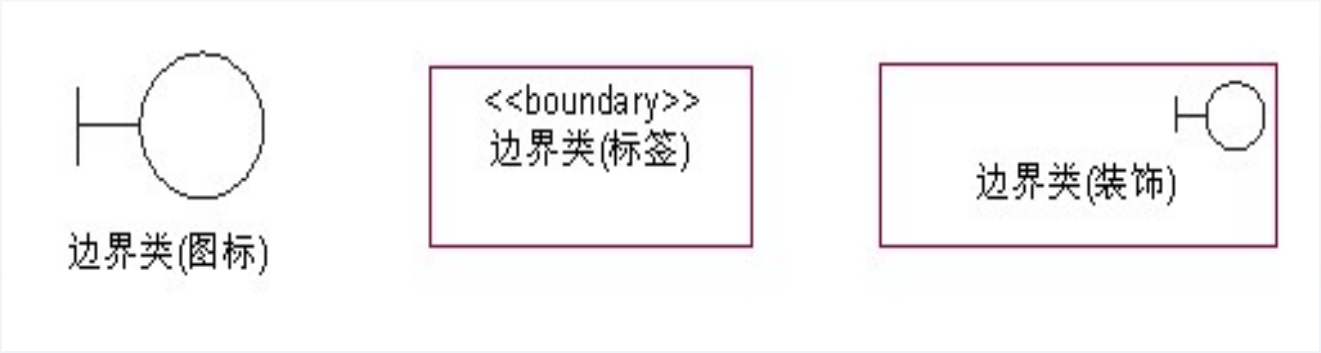
② 缩略表示



类图-相关概念

边界类：

边界类位于系统与外界的交界处，承担系统与外界的信息功能.



边界类处在用例图中，参与者与用例的关联处，可以根据用例图发现边界类。

类图-相关概念

实体类：

实体类对应着现实中的客观实物，用来保存信息，一般对应着数据表、文件等。

实体类可以从现实中存在的客观事物，以及需要持久存放的信息两方面来发现。

类图-相关概念

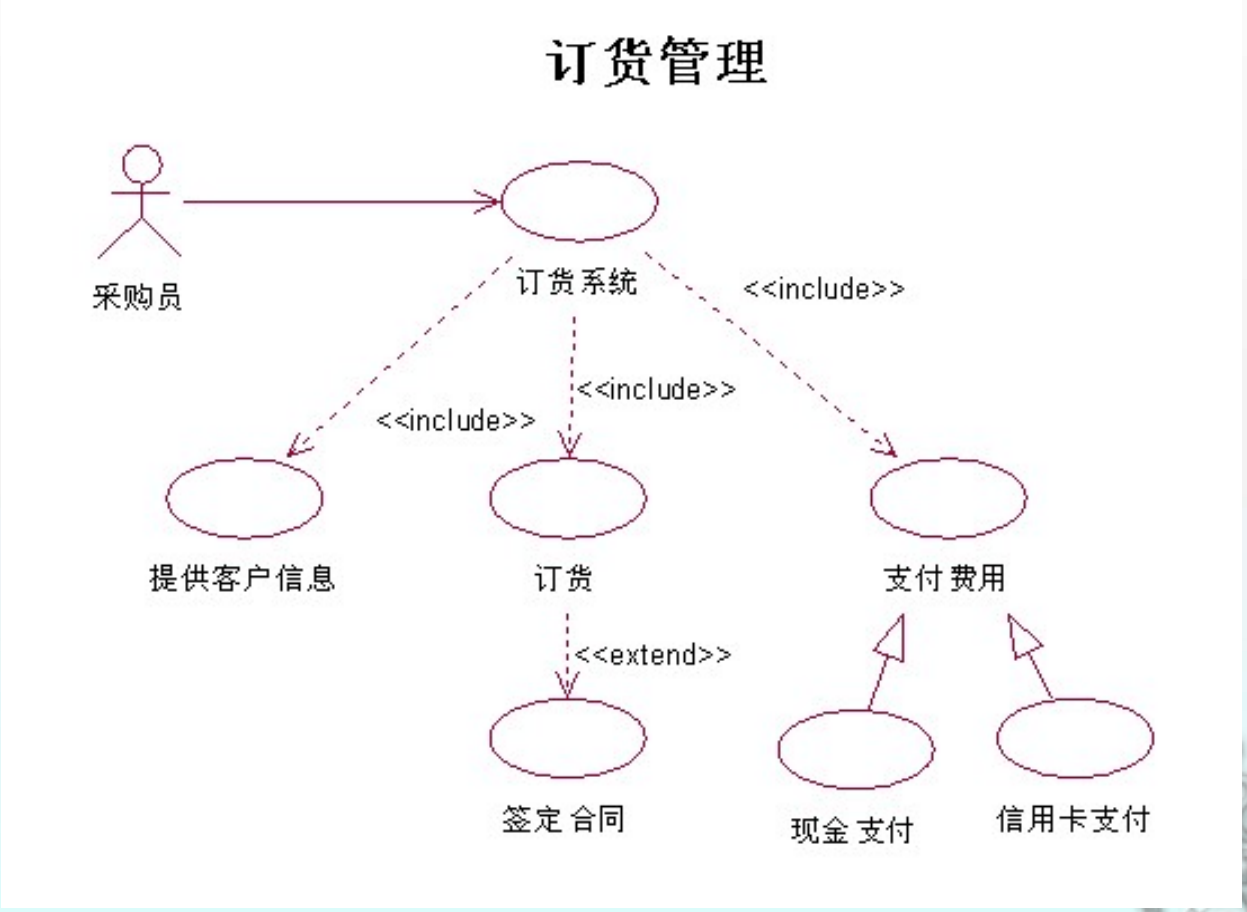
控制类：

控制类承担着事务处理，控制调控的控制作用。

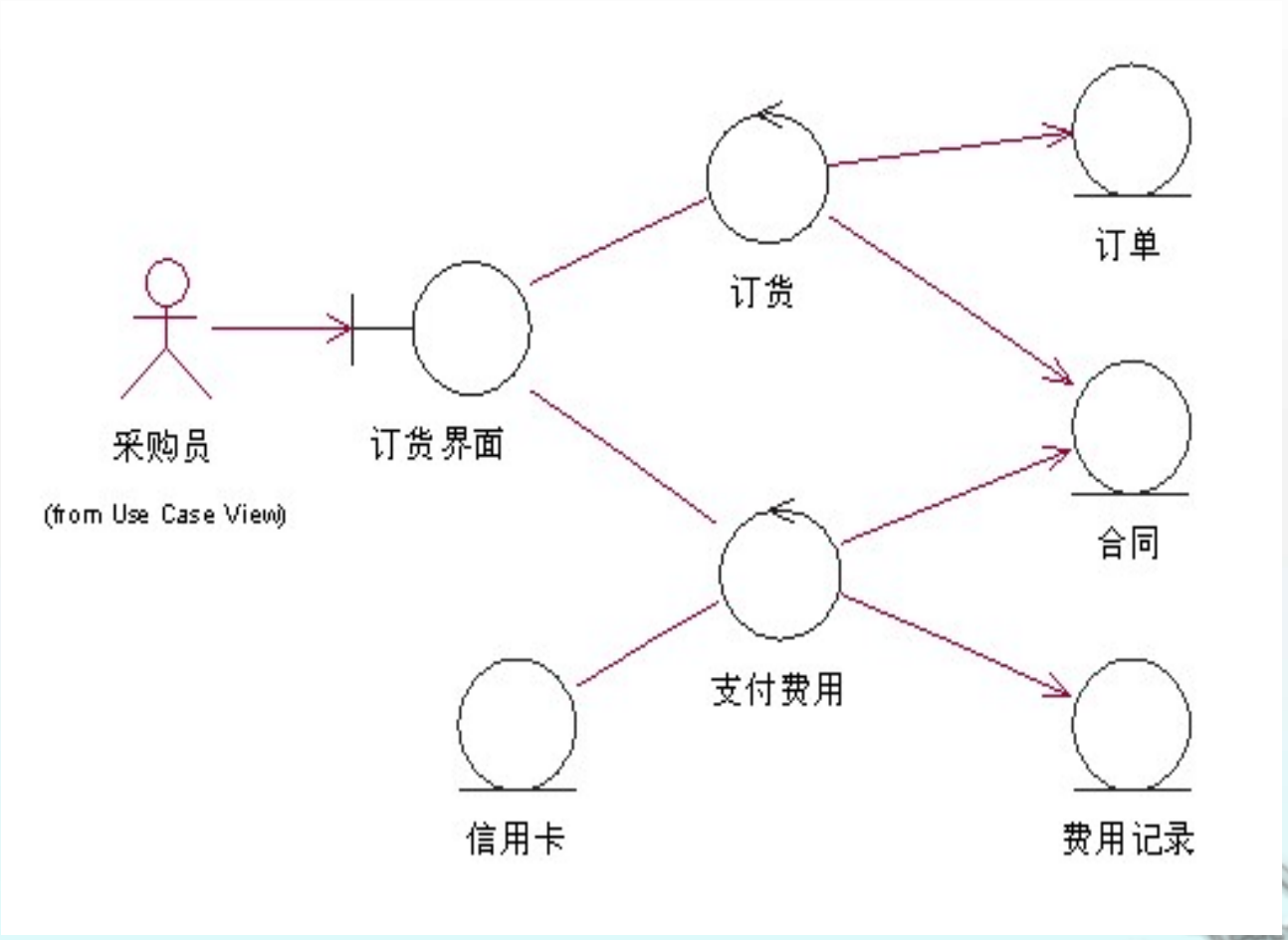
一个用例中最少会有一个控制类，用来控制用例中的事件顺序，也可以在多个用例之间协调用例之间的联系。

类图-相关概念

例子：订货系统



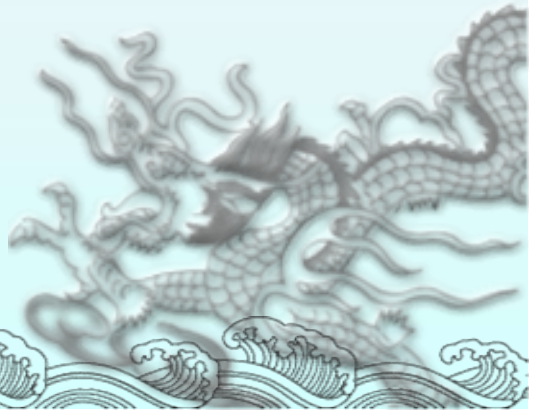
类图-相关概念



类图-相关概念

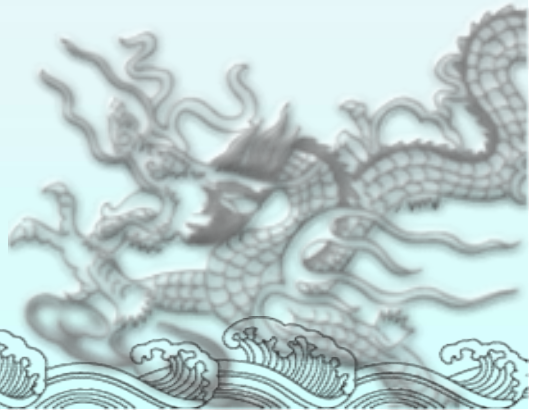
建立类图的一般步骤：

- 研究分析问题领域，确定系统需求；
- 确定类，明确类的含义和职责，确定类的属性和操作；
- 确定类之间的关系。关联，泛化，聚合，组合，依赖；
- 调整和细化类及其关系，解决重复和冲突；
- 绘制类图，并增加相应说明。



内容提纲

- 类图
  - 相关概念
  - 类间关系
  - 代码映射
  - 实例讲解
- 对象图
  - 相关概念
  - 实例讲解



类图-类间关系

关联关系

- 关联(Association)：模型元素之间的一种语义联系，它是对具有共同的结构特性、行为特性、关系和语义的链接的描述。
- 在类图使用带箭头的实线表示，箭头从使用类指向被关联的类。可以是单向或双向。
- 两个类之间的关联将在所生成的代码中体现出来，表现为类中有一个方法可以访问到另一个类。
- 关联又分为一般关联、聚合关联与组合关联。

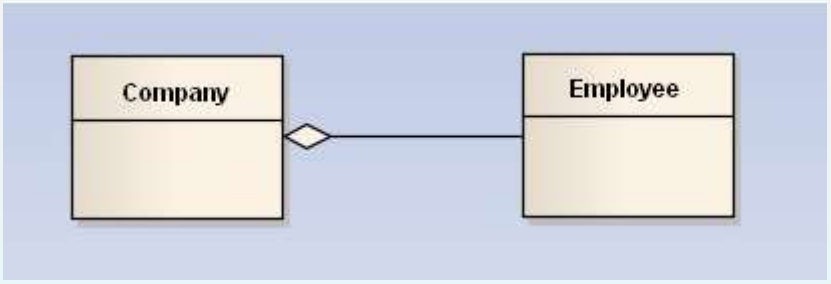
UML表示法

类图-类间关系

聚合关系

- 聚合(Aggregation)：表示has-a的关系，是一种不稳定的包含关系。较强于一般关联,有整体与局部的关系,并且没有了整体,局部也可单独存在。
- 在类图使用空心的菱形表示。

UML表示法

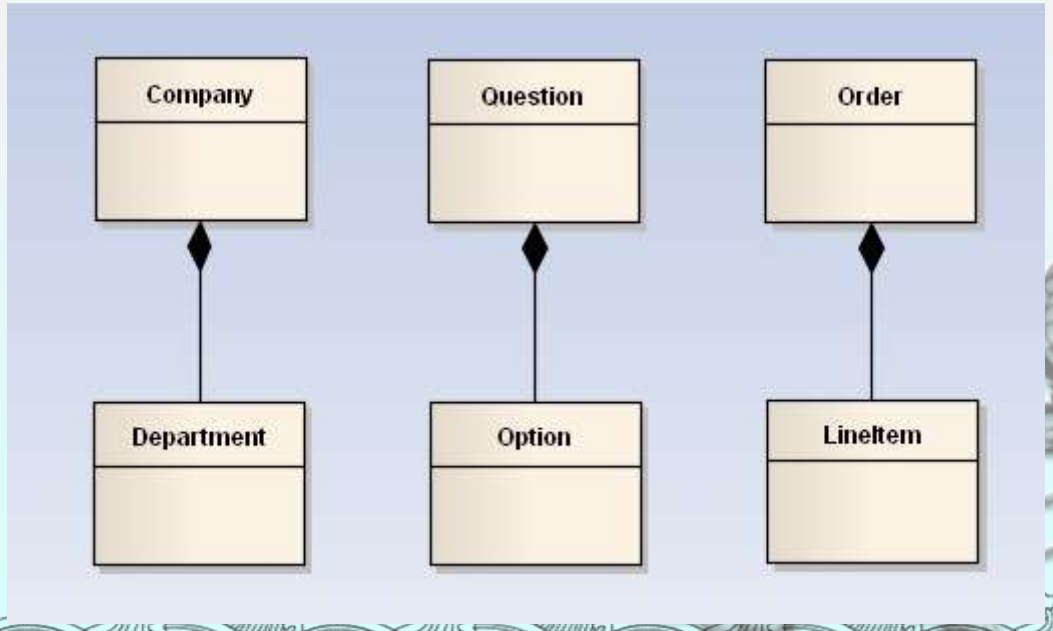


类图-类间关系

组合关系

- 组合(Composition)：表示contains-a的关系，是一种强烈的包含关系。组合类负责被组合类的生命周期。是一种更强的聚合关系。部分不能脱离整体存在。
- 如公司和部门的关系，没有了公司，部门也不能存在了；调查问卷中问题和选项的关系；订单和订单选项的关系。
- 在类图使用实心的菱形表示。

UML表示法

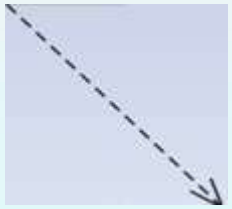


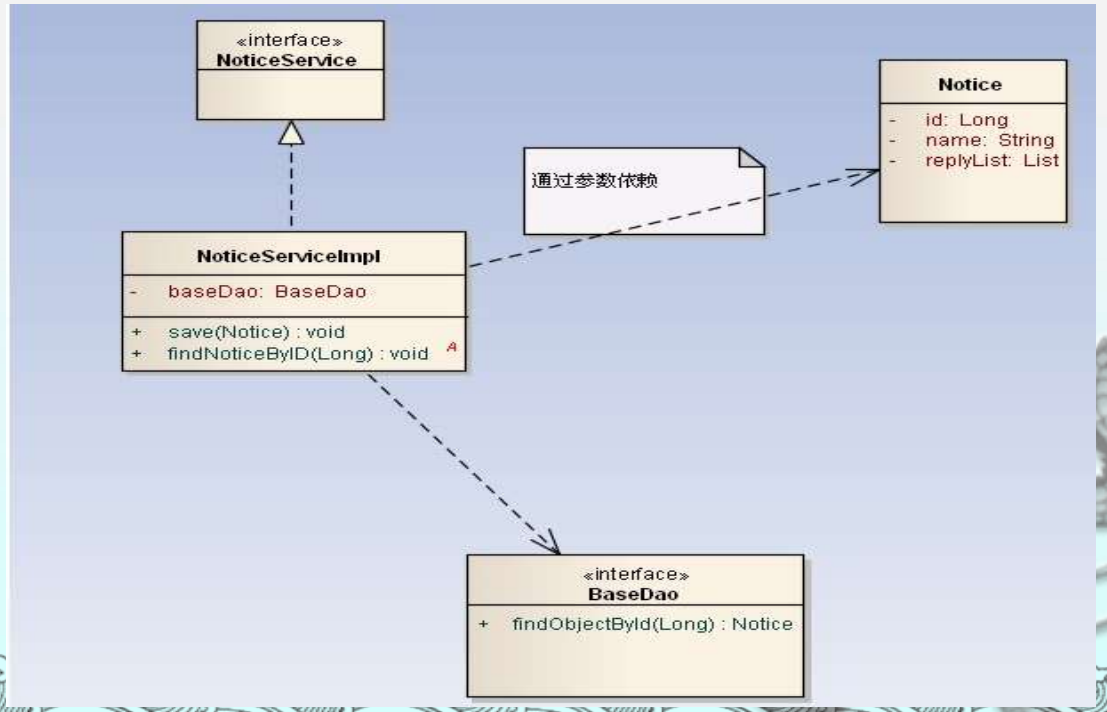
类图-类间关系

依赖关系

- 依赖(Dependency)：对象之间最弱的一种关联方式，是临时性的关联。代码中一般指由局部变量、函数参数、返回值建立的对于其他对象的调用关系。一个类调用被依赖类中的某些方法而得以完成这个类的一些职责。
- 在类图使用带箭头的虚线表示，箭头从使用类指向被依赖的类。

UML表示法



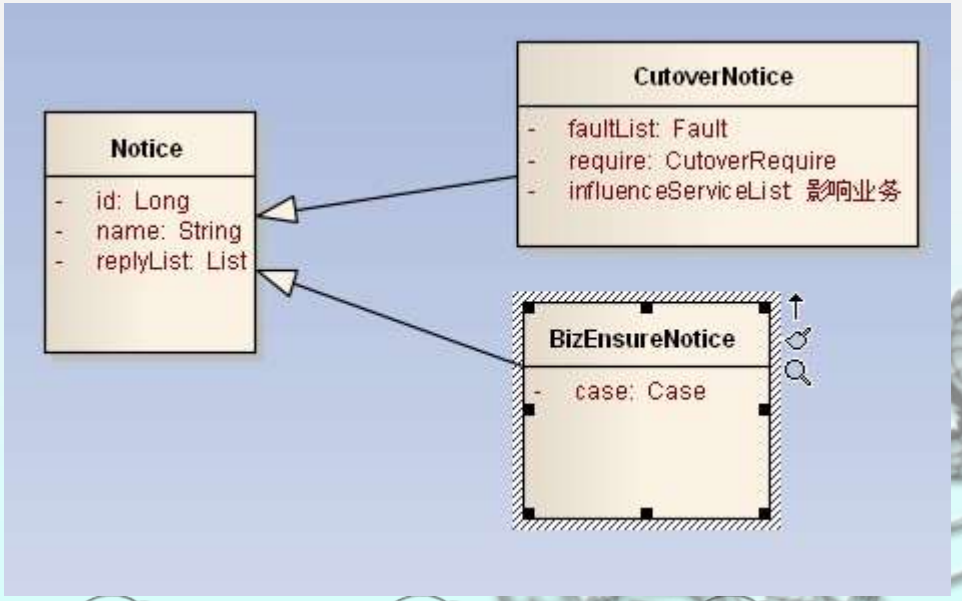


类图-类间关系

泛化关系

- 泛化(generalization)：表示is-a的关系，是对象之间耦合度最大的一种关系，子类继承父类的所有细节。直接使用语言中的继承表达。
- 在类图中使用带三角箭头的实线表示，箭头从子类指向父类。

UML表示法

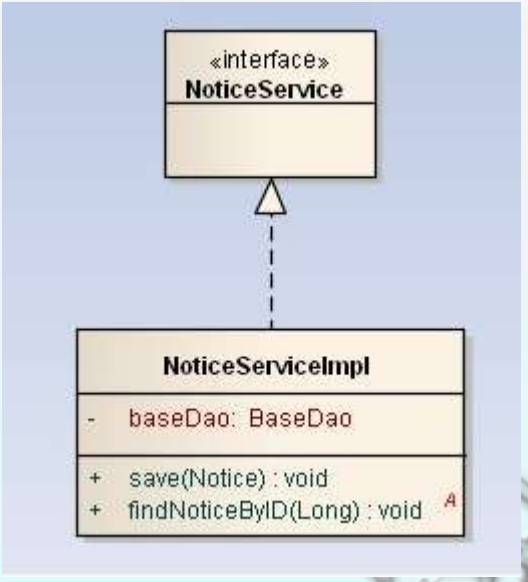


类图-类间关系

实现关系

- 实现（Realization）:在类图中就是接口和实现的关系。
- 在类图中使用带三角箭头的虚线表示，箭头从实现类指向接口。

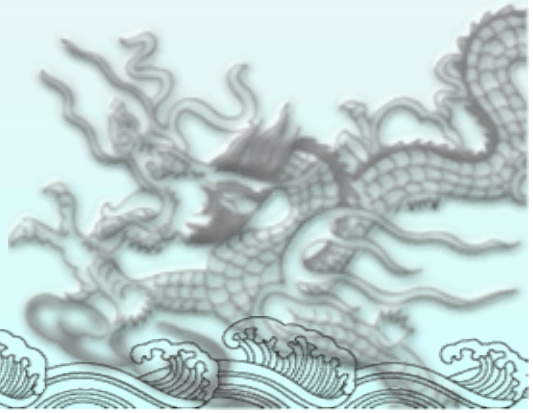
UML表示法



类图-类间关系

实例

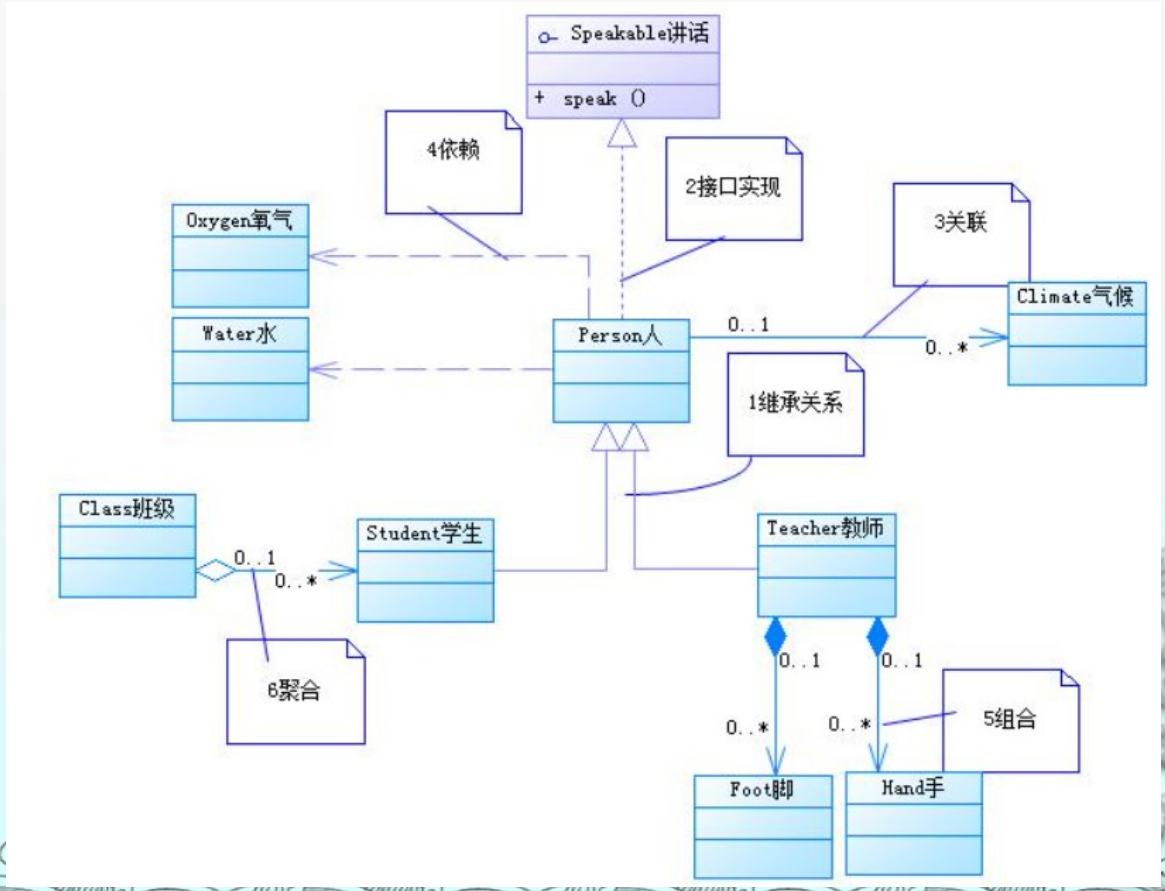
- 人类的生存需要氧气和水，需要适宜的气候。人类能够通过说话进行交流。老师和学生都属于人类，老师和学生都有手有脚，学生存在于某一个班级。



类图-类间关系

实例

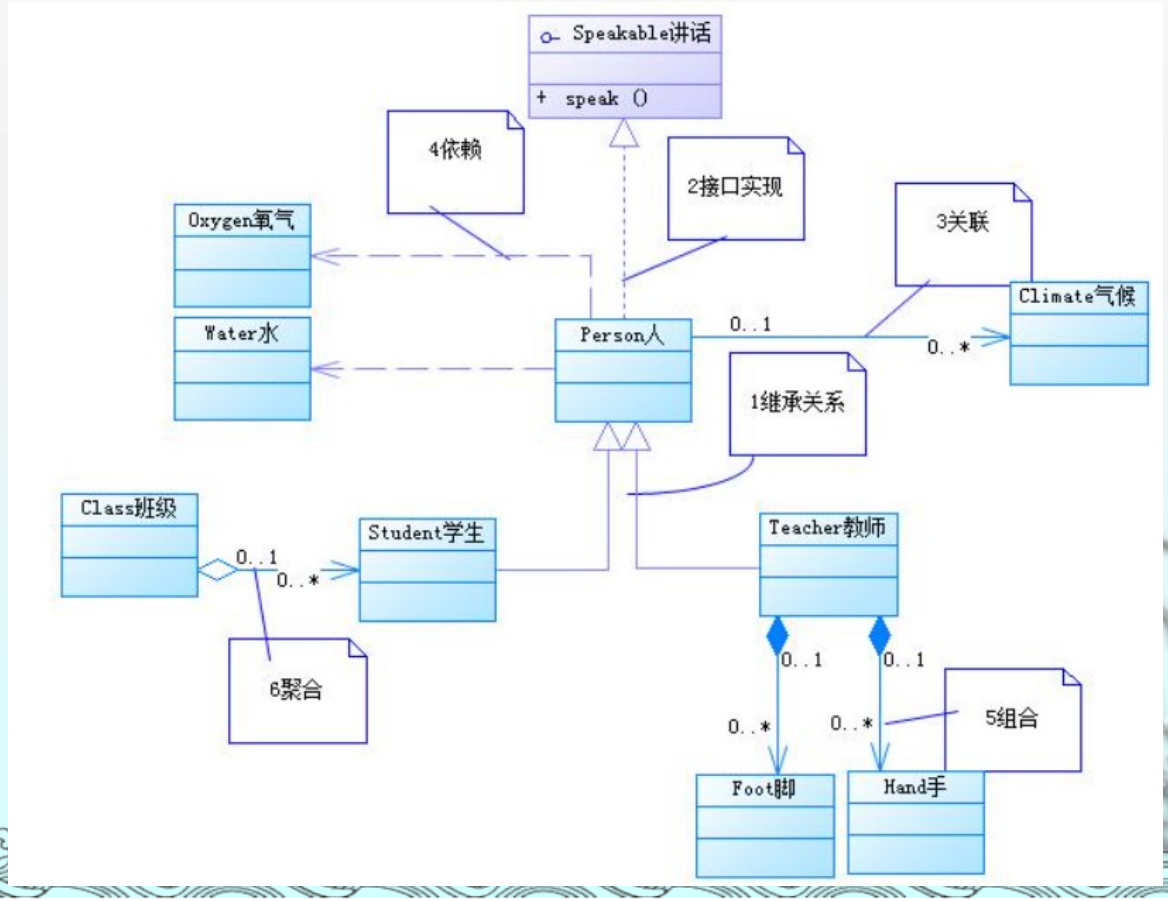
- 人类的生存需要氧气和水，需要适宜的气候。人类能够通过说话进行交流。老师和学生都属于人类，老师和学生都有手有脚，学生存在于某一个班级。



类图-类间关系

问题

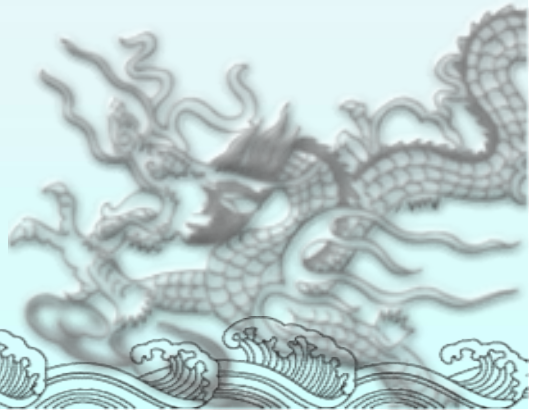
- 1、这些图对于理解需求有帮助吗？
- 2、这些图对于开发系统有帮助吗？
- 3、这些图对于维护系统有帮助吗？



类图-类间关系

问题

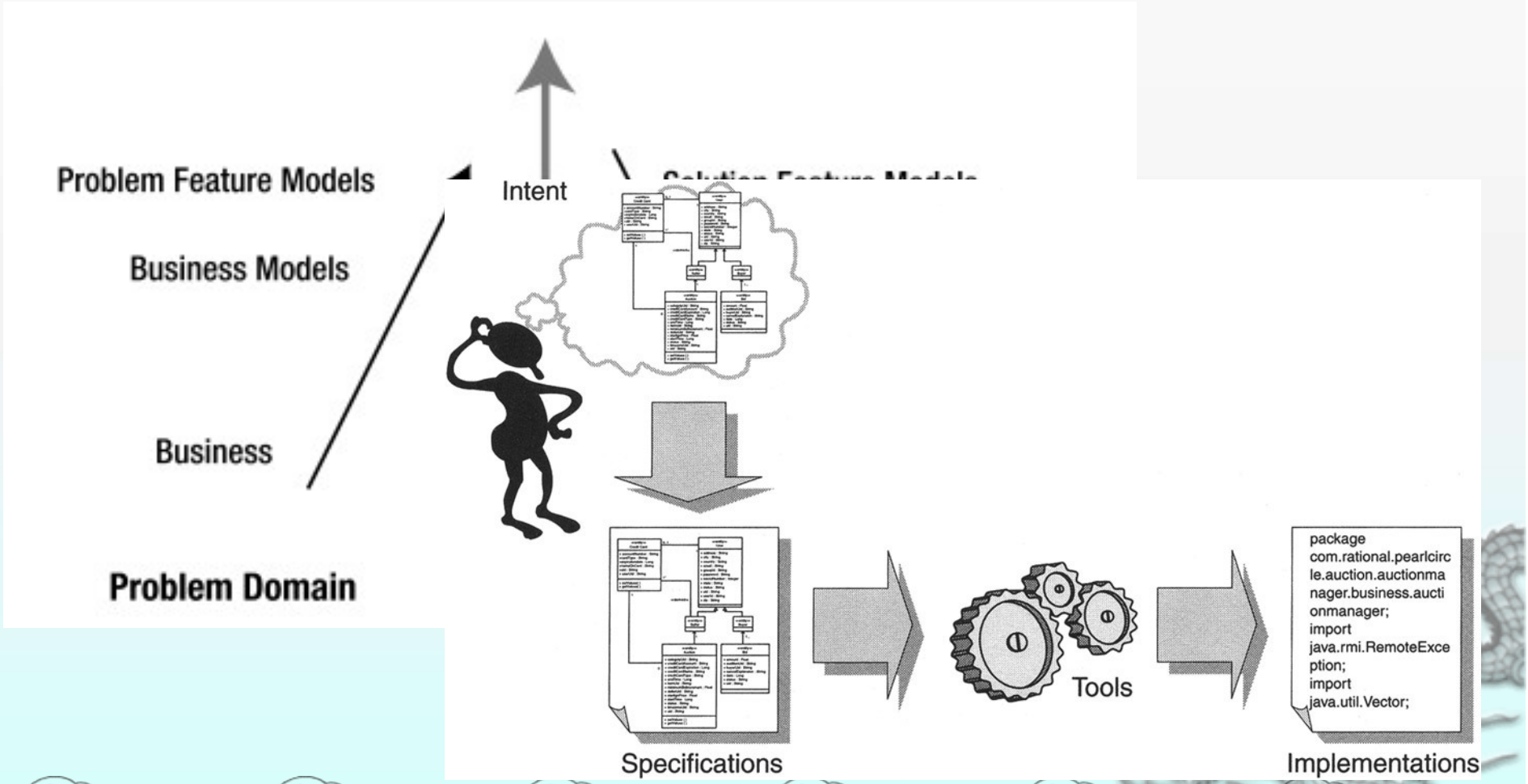
能否根据图直接生成代码？



类图-类间关系

补充知识

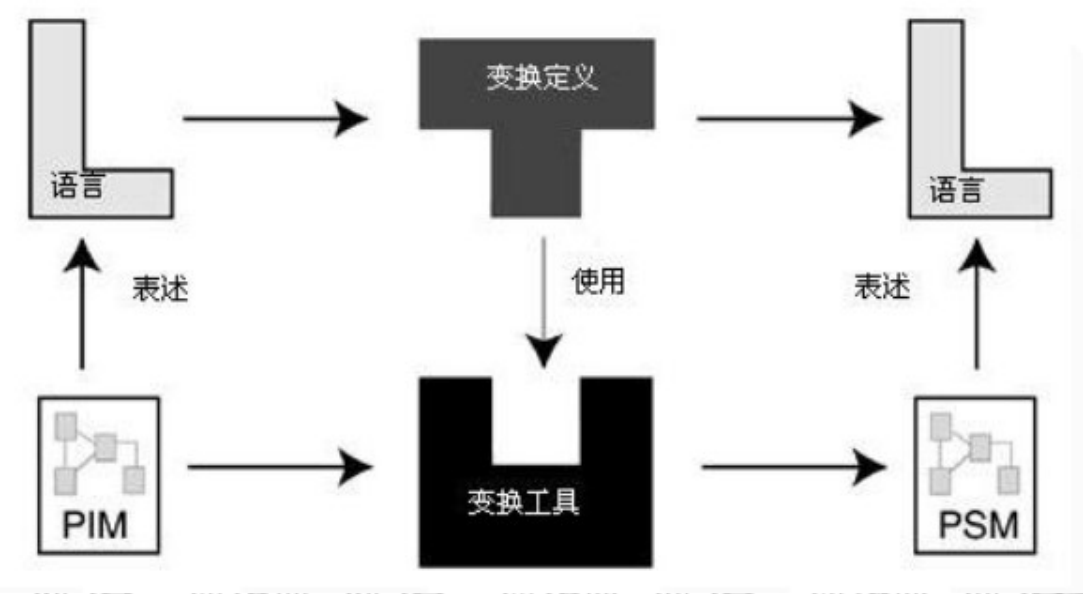
- MDA



类图-类间关系

补充知识

- MDA
  - MDA 将软件系统的模型分离为平台无关模型PIM和特定平台模型PSM，同时又能通过转换规则将它们统一起来。
  - 平台无关模型PIM 是对系统高层次的抽象，其中不包括任何与实现技术相关的信息；特定平台模型PSM是特定平台相关的模型。
  - 在MDA 框架中，首先使用平台无关的建模语言来搭建平台无关的模型PIM，然后根据特定平台和实现语言的映射规则，将PIM 转换以生成平台相关的模型PSM，最终生成应用程序代码和测试框架。



类图-类间关系

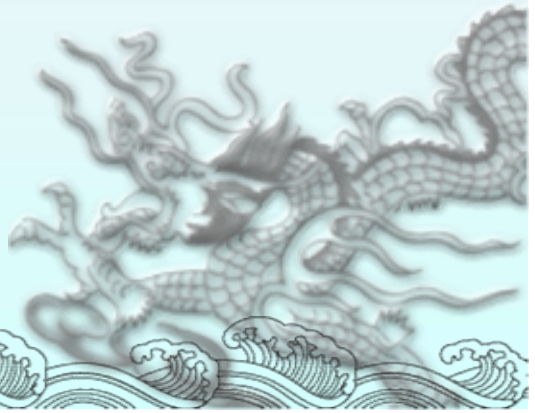
问题

能否根据图直接生成代码？

关键：模型转换！

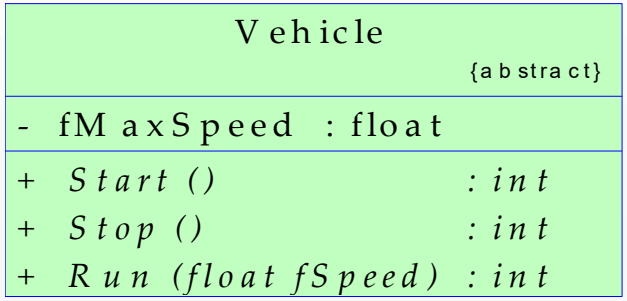
内容提纲

- 类图
  - 相关概念
  - 类间关系
  - 代码映射
  - 实例讲解
- 对象图
  - 相关概念
  - 实例讲解



类图-代码映射

类的映射



C++代码

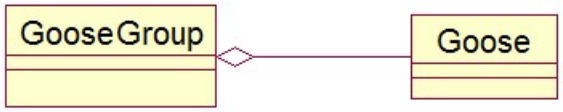
```
class Vehicle
{
public:
   virtual int Start() = 0;
   virtual int Stop() = 0;
   virtual int Run(float fSpeed) = 0;
private:
   float fMaxSpeed;
};
```

Java代码

```
public abstract class Vehicle
{
  public abstract int Start();
  public abstract int Stop();
  public abstract int Run(float fSpeed);

  private float fMaxSpeed;
}
```

类图-代码映射

聚合的映射：雁群类是由大雁类聚合而成




```
Public class GooseGroup {
    public Goose goose;
    Public GooseGroup(Goose goose) {
            this.goose = goose;
    }
}
```

类图-代码映射

组合的映射


```
Public class Goose {
   public Wings wings;
   public Goose() {
      wings = new Wings();
    }
}
```

（1）构造函数不同

聚合类的构造函数中包含了另外一个类作为参数，GooseGroup的构造函数中用到Goose作为参数传递进来，Goose可以脱离GooseGroup独立存在

组合类的构造函数中包含了一个类的实例化，表明大雁类在实例化之前，一定要先实例化Wings类，这两个类紧密的耦合在一起，同生共灭。wings类是不可以脱离大雁类独立存在的

（2）信息的封装性不同

聚合关系中，客户端可以同时了解GooseGroup和大雁类，因为他们是独立的

在组合关系中，客户端只认识大雁类，根本就不知道wings类的存在，因为wings类被封装在Goose类中

类图-代码映射

依赖的映射


```
Public class Animal()
{
      Public Animal(){}
}

Public class Water()
{
      public Water(){}
}
```

依赖关系如何体现?

1. water类是animal类的某个方法中的变量

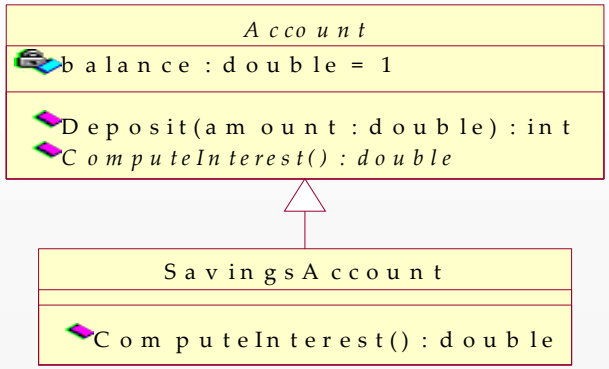
```
Public class Animal {
    Public void Grownup() {
          Water water =null;
    }
}
```

2. water类是作为animal类中某个方法的参数或者返回值

```
Public Animal {
  Public Water
Grownup(Waterwater) {
          return null;
    }
}
```

类图-代码映射

泛化的映射



C++代码

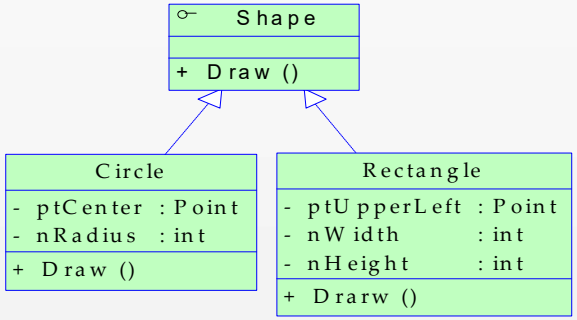
```
class SavingsAccount : public
Account
{};
```

Java代码

```
public class SavingsAccount extends
Account
{}
```

类图-代码映射

实现的映射



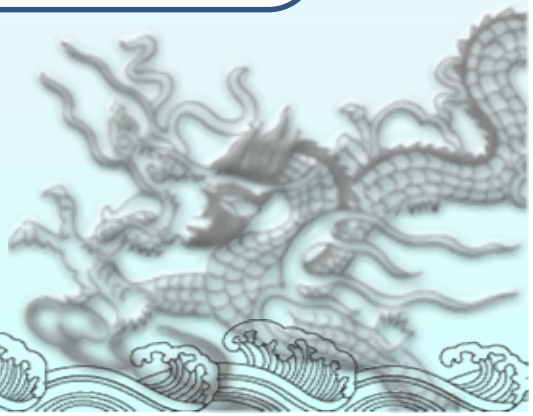
Java代码

```
public interface Shape
{
public abstract void Draw();
}

public class Circle implements Shape
{
public void Draw();

private Point ptCenter;
private int nRadius;
}
```

类图-代码映射

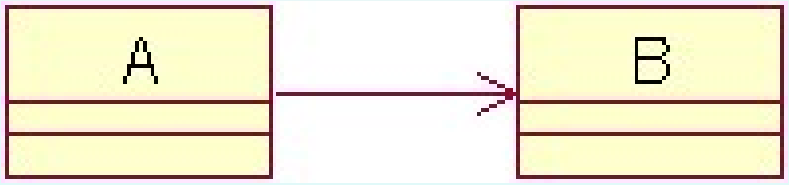
这些图到代码的映射都这么简单，MDA似乎大有可为！

问题：为什么我们现在还需要编码？



类图-代码映射

关联的映射

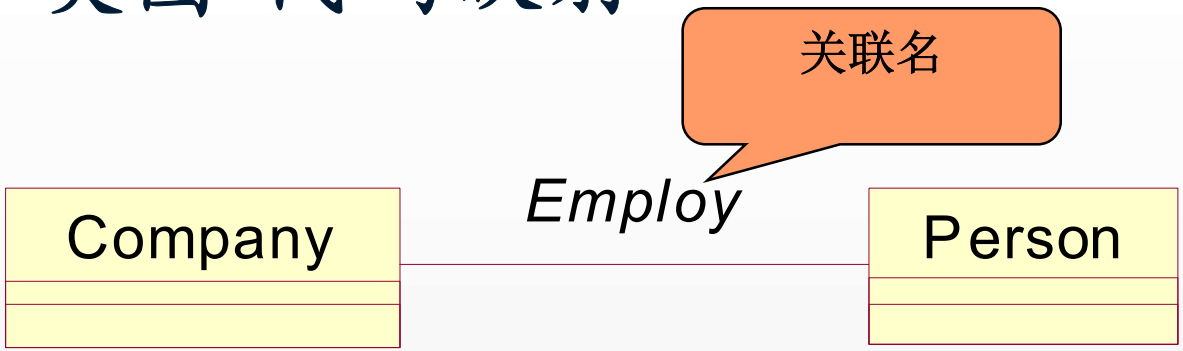



```
// 类A的源码
public class A
{
    public B theB;
    public A() { }
}
// 类B的源码
public class B
{
    public A theA;
    public B() { }
}
```

```
{
    public B() { }
}
```

类图-代码映射

关联的映射



关联名

- 用于明确表达关联的含义。客户？雇员？老板？
- 可有可无 动词短语 斜体

导航性（方向性）

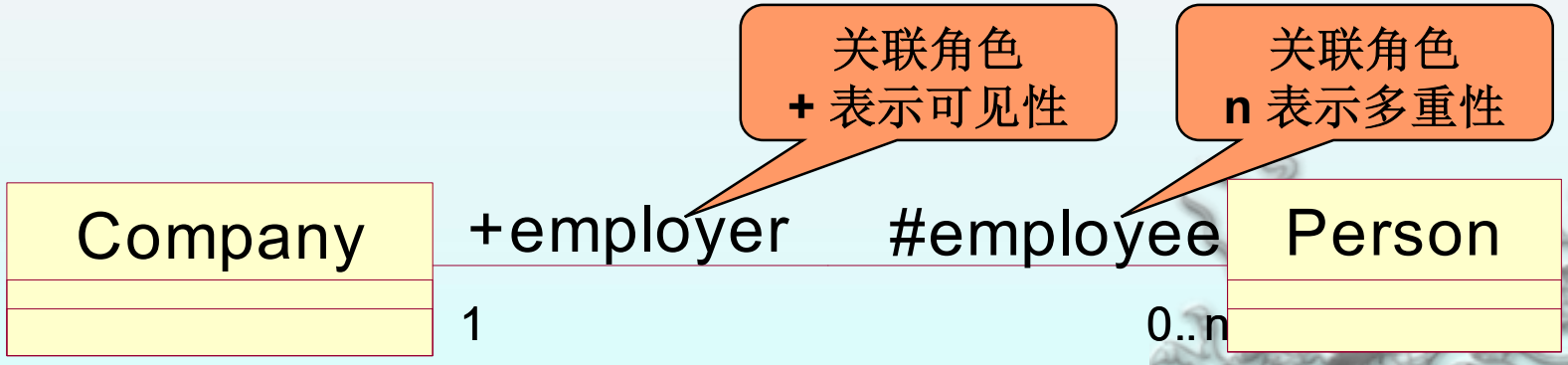
- 从一个类（对象）可以访问到另一个，反过来却不行，用带箭头的实线，表示单向关联，无箭头表示双向关联
- 被关联的对象不知道谁与自己关联，但关联对象知道自己与谁有关联

类图-代码映射

关联的映射

关联角色（role）

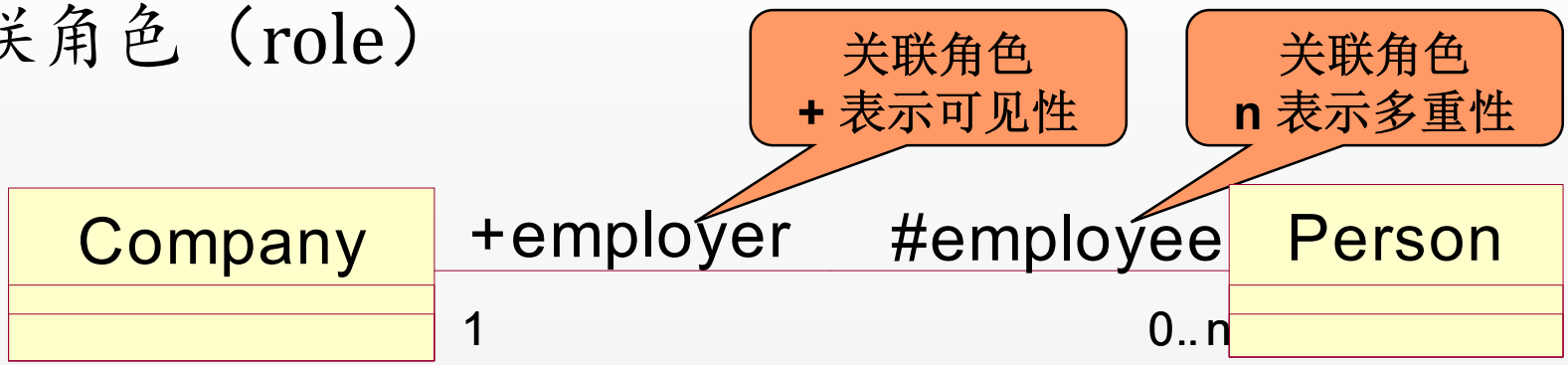
- 关联两端的类可以以某种角色参与关联。
- 如果在关联上没有标出角色名，则隐含地用类名作为角色名。
- 生成代码后会有和角色名相应的属性出现。
- 自返关联通常必须标明角色。



类图-代码映射

关联的映射

关联角色（role）




```
class Company
   {
      public string name;     //公司名称
      public Person[ ] employee = new Person[3];

      public Company(string cName)  //构造函数
      {
         name = cName;
      }
      ……
   }
```

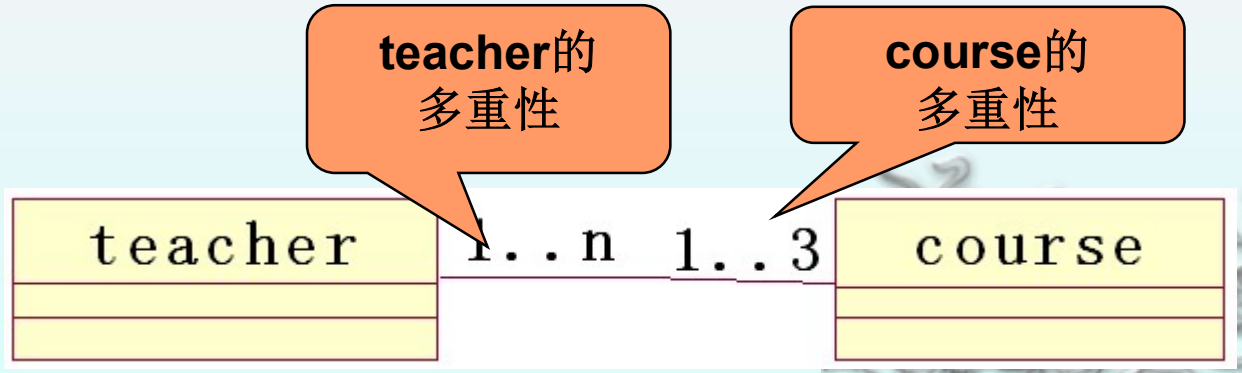
类图-代码映射

关联的映射

关联角色的多重性（multiplicity）

- 多重性表示可以有多少个对象参与该关联
- 非负整数的子集表示
- 0 表示没有实例的关联，一般不用
- 0..1
- 0..n
- 1 即1..1
- 3..6
- 1..n
- n

一个教师可开设1到3门课程，一门课程可以有1到多个教师

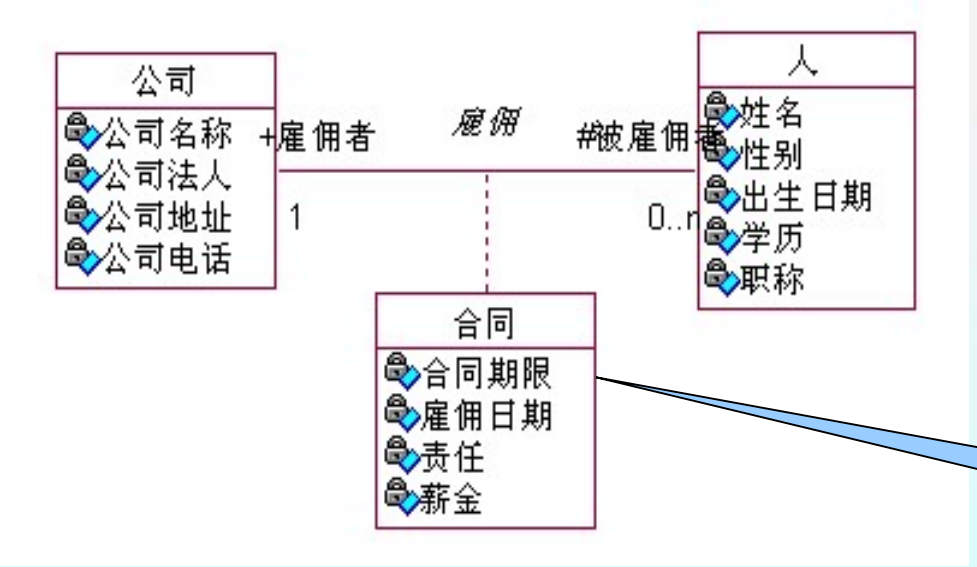


类图-代码映射

关联的映射

关联类（association class）

- 关联本身也可以有特性，通过建立关联类可以进一步描述关联的属性、操作和其他信息。
- 通过虚线与关联连接。

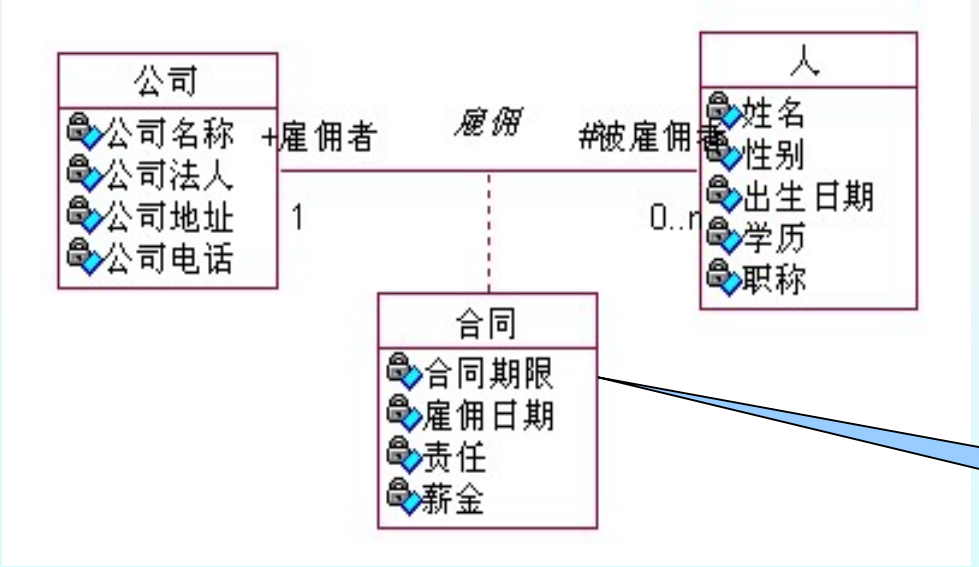


类图-代码映射

关联的映射

关联类（association class）

- 关联本身也可以有特性，通过建立关联类可以进一步描述关联的属性、操作和其他信息。
- 通过虚线与关联连接。

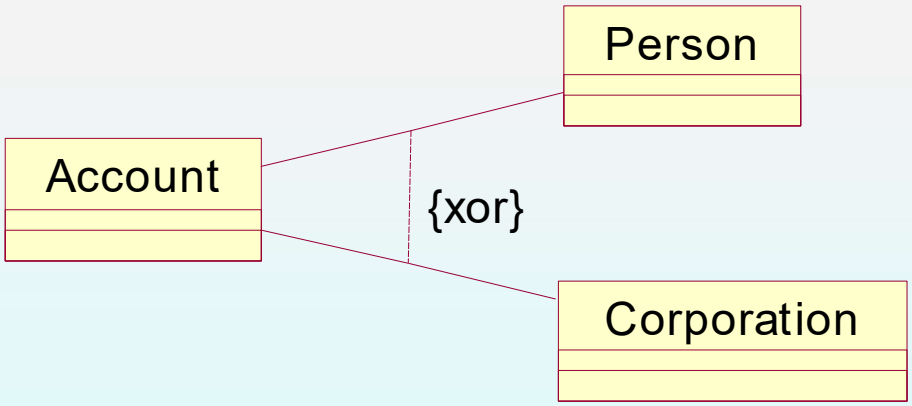


类图-代码映射

关联的映射

关联的约束/依赖

- 给关联加上约束，可以加强关联的含义。
- 例如：“帐户”不能同时与“人”和“公司”有关联。

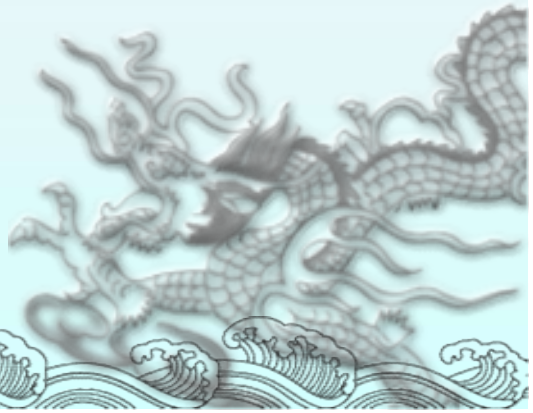


类图-代码映射

关联的映射

限定关联

- 在关联端紧靠源类图标处可以有限定符（qualifier），带有限定符的关联称为受限关联或限定关联。
- 限定符的作用就是在给定关联一端的一个对象和限定符之后，可以唯一确定另一端的一个对象或对象集。
- 受限关联用于一对多或多对多的关联。将目标类的多重性从“多”降为“一”。



类图-代码映射

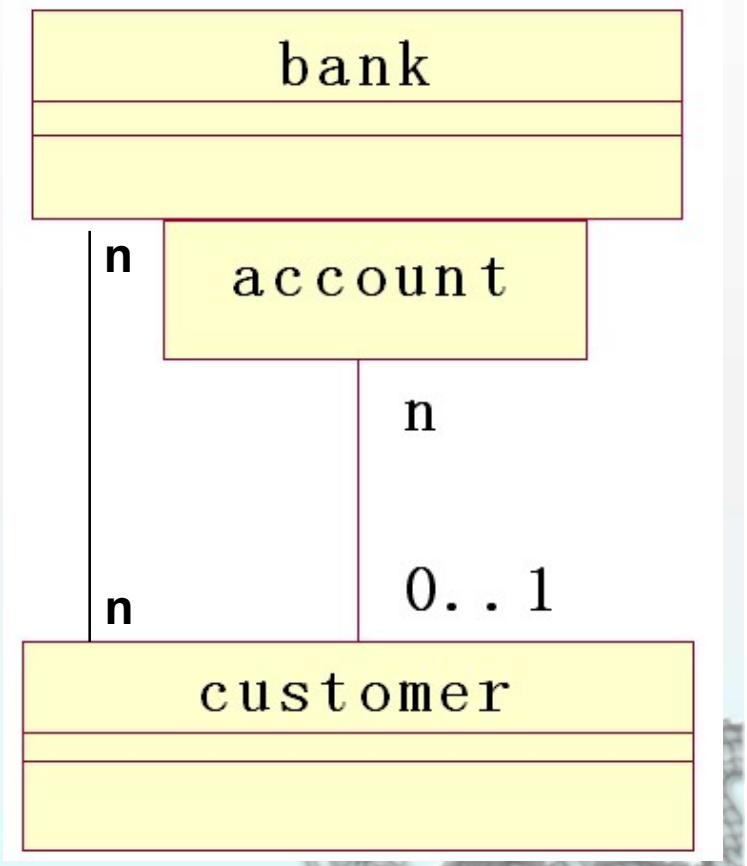
关联的映射

限定关联

- 例：一客户可以在bank中有多个
  - 账户，但给定了一个账户account后，其只对应0..1个客户

  (bank, account) → 0..1个costomer

  costomer → 多个(bank, account)

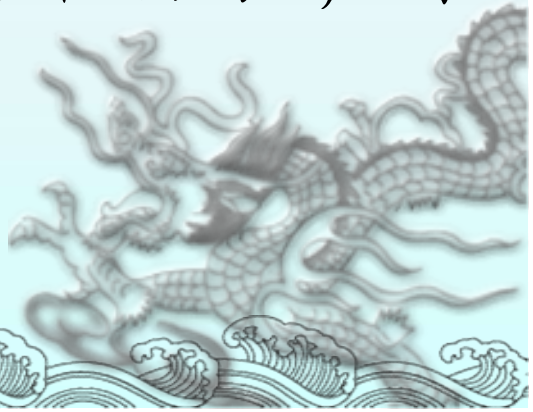


类图-代码映射

关联的映射

限定关联

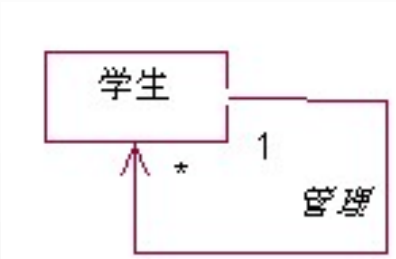
- 为何引入限定符？
  - 在设计软件时十分有用
  - 应用系统根据关键字对数据集作查询，常用到受限关联。引入限定符的一个目的就是把多重性从n降到1或0..1，则查询结果是单个对象，效率高。
  - 如果引入限定符后，另一端的多重性仍为n，则意义不大。



类图-代码映射

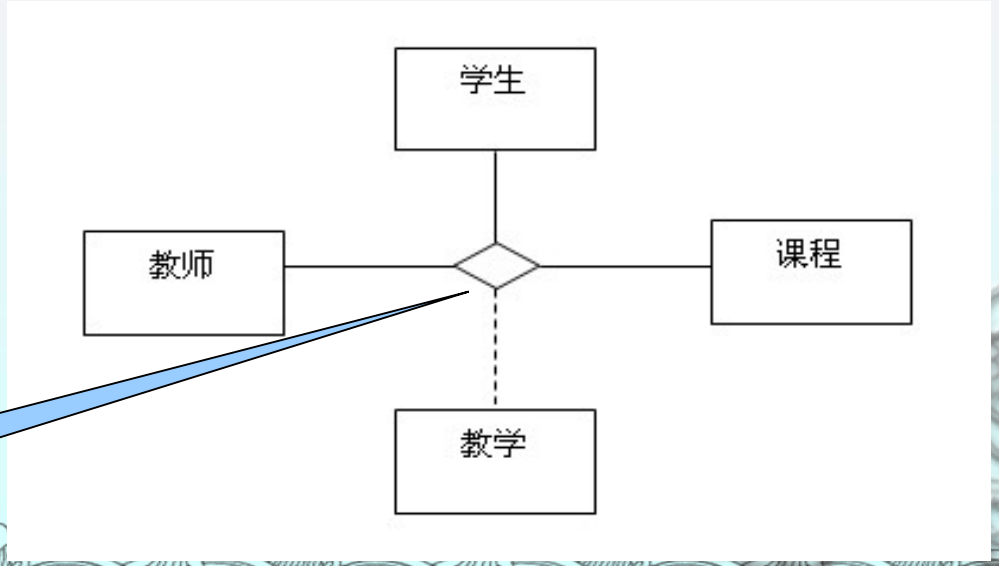
关联的映射

一元(自返)关联



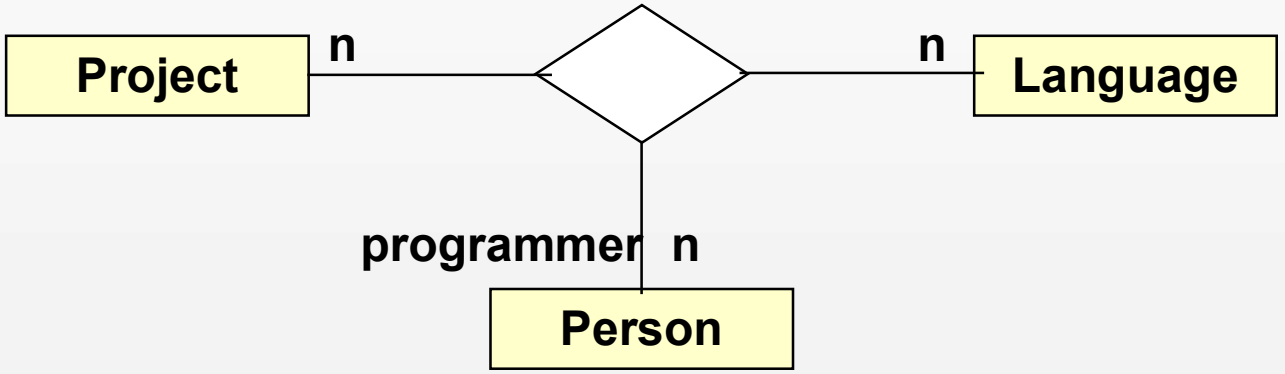
二元关联

多元关联



类图-代码映射

关联的映射



- 一个人在一个项目中可以使用多种开发语言；
- 一个项目中使用某种语言的可以有多个开发者；
- 一个开发者用某种语言可以开发多个项目。

类图-代码映射

关联的映射

自返关联

```
public class Node {
private Node subNode;
……
}
```

二元关联（双向关联）

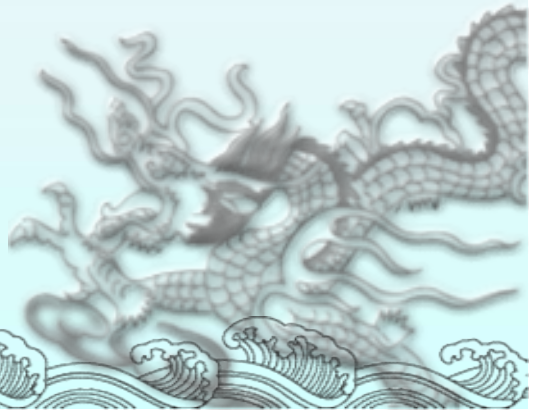
```
public class Customer {
private Product[]
products;
……
}

public class Product {
private Customer
customer;
……
}
```

二元关联（单向关联）

```
public class Customer
{
private Address
address;
……
}

public class Address {
……
}
```

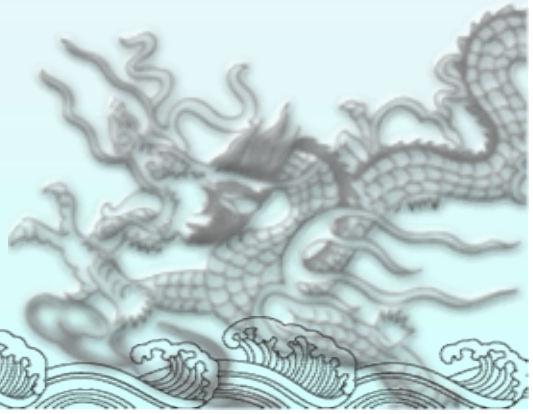
多重性

```
public class Form {
private Button[] buttons;
//定义一个集合对象
……
}

public class Button {
……
}
```



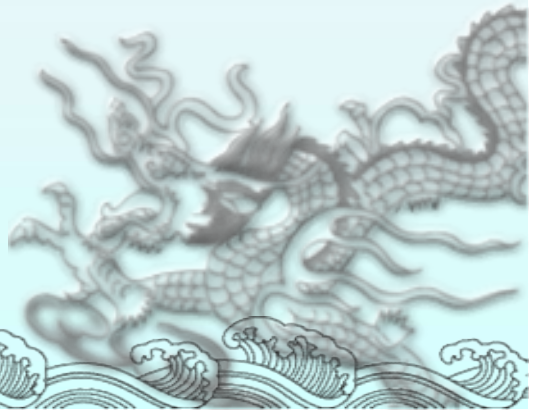
类图-代码映射

- 关联与依赖的区别
  - 依赖关系含义：是类与类之间的连接，表示一个类依赖于另外一个类的定义；
  - 依赖关系仅仅描述了类与类之间的一种使用与被使用的关系。



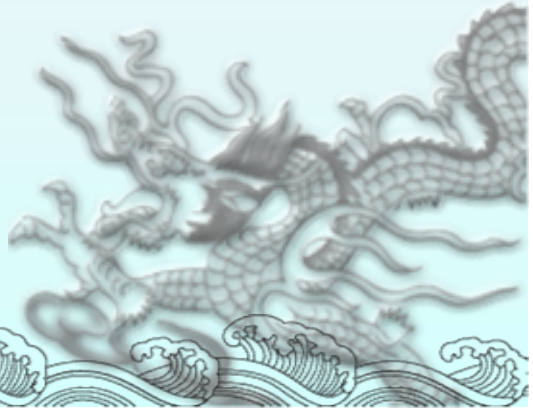
类图-代码映射

- 关联与聚合的区别
  - 聚合含义：是关联关系的一种，是一种强关联关系；聚合关系是整体和个体/部分之间的关系；
  - 关联关系的两个类处于同一个层次上，而聚合关系的两个类处于不同的层次上，一个是整体，一个是个体/部分；
  - 在聚合关系中，代表个体/部分的对象有可能会被多个代表整体的对象所共享；



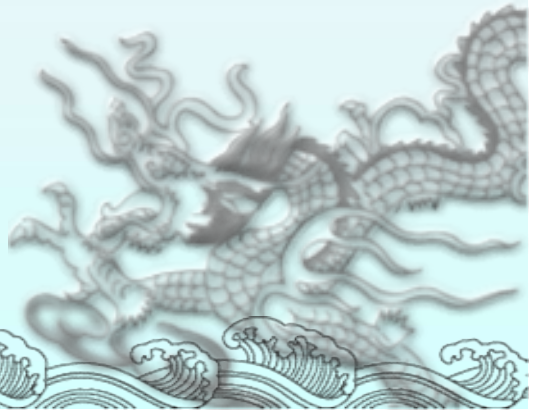
类图-代码映射

- 聚合与组合的区别
  - 组合关系含义：它也是关联关系的一种，但它是比聚合关系更强的关系.组合关系要求聚合关系中代表整体的对象要负责代表个体/部分的对象的整个生命周期；
  - 组合关系不能共享；在组合关系中，如果代表整体的对象被销毁或破坏，那么代表个体/部分的对象也一定会被销毁或破坏，而聚在合关系中，代表个体/部分的对象则有可能被多个代表整体的对象所共享，而不一定会随着某个代表整体的对象被销毁或破坏而被销毁或破坏；
  - 举例：一个人由头、四肢、等各种器官组成，因为人与这些器官具有相同的生命周期；



类图-代码映射

- 总结
  - 当某个类被当作参数传递并且被当作结果返回的时候，或者被当作某个方法内的临时变量使用的时候,可以运用依赖关系；
  - 使用关联来表示一个拥有关系，而不是整体-部分关系；
  - 使用聚合来表示一个动态的整体-部分关系；
  - 用组合来表示一个静态的整体-部分关系。



类图-代码映射

- 总结
  - 需要指出的是，所谓“关系”只是在某个问题域才有效，离开了这个问题域，可能这些关系就不成立了。
  - 例如：

    汽车作为一个整体，轮胎是一定要组合在汽车类中的，因为它离开了汽车就没有意义了。

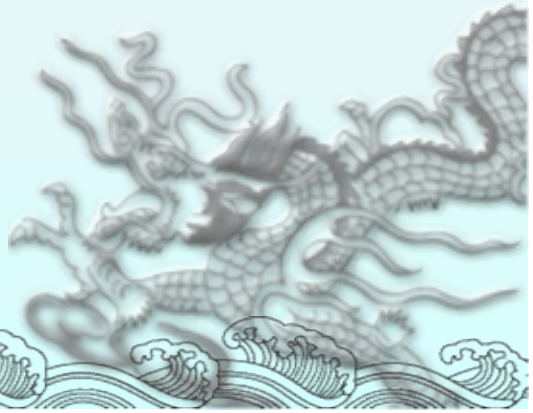
    但是，在卖轮胎的店铺业务里，就算轮胎离开了汽车，它也是有意义的，这就可以用聚合了。

    这说明关系是在特定的问题域中的“关系”，会随着问题域的迁移而改变的。

MDA任重道远：问题域太多，PSM模型很难构建！

内容提纲

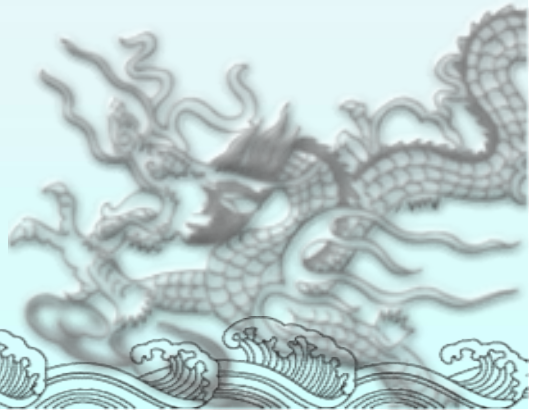
- 类图
  - 相关概念
  - 类间关系
  - 代码映射
  - 实例讲解
- 对象图
  - 相关概念
  - 实例讲解



类图-实例讲解

实例1

“班级”和“班长”两个类之间存在管理关系，一个班级仅可以有一个班长，一个班长只能是一个班级的班长，标出这两个类的关系。



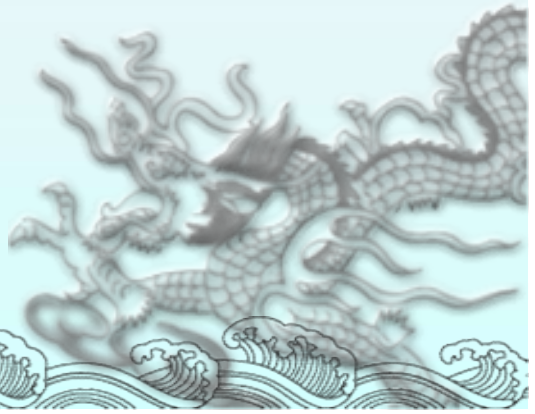
类图-实例讲解

实例1

类图-实例讲解

实例2

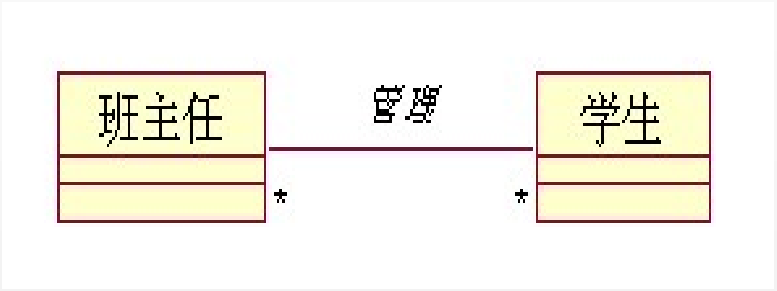
“班主任”和“学生”两个类之间存在管理关系，标出这两个类的关系。

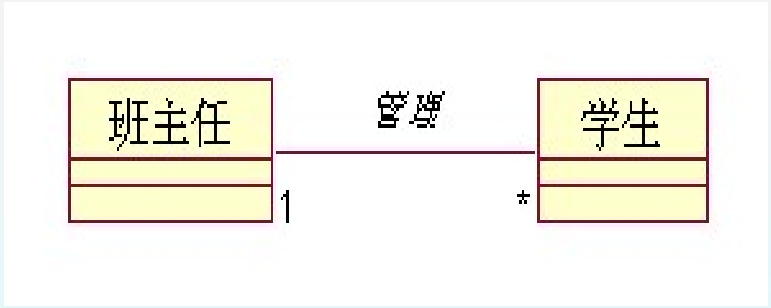


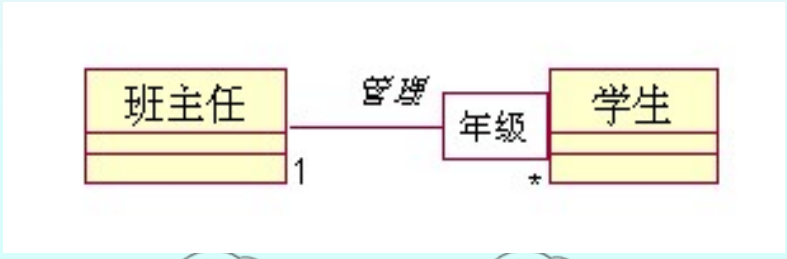
类图-实例讲解

实例2

“班主任”和“学生”两个类之间存在管理关系，标出这两个类的关系。







类图-实例讲解

实例3

“教师”和“学生”两个类之间存在授课关系，一个教师可以教授多个学生，一个学生可以由多个教师授课，标出这两个类的关系。

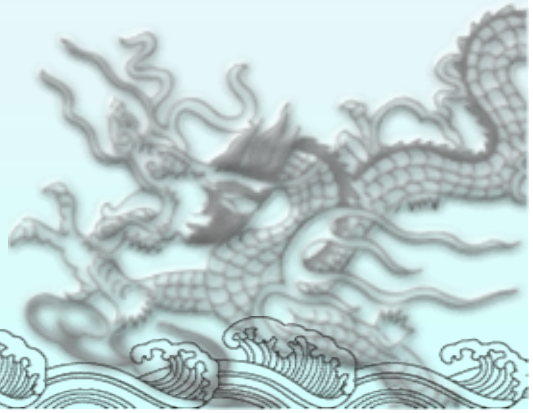
类图-实例讲解

实例4

采购员从供货商处订货，双方需要签订订单，一个采购员可以订多个供货商的货品，一个供货商也可以给多个采购员供货。

要求：

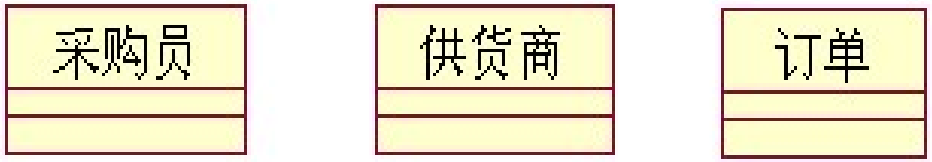
1. 提取这个问题涉及的类；
2. 定义各个类之间的关系，并画出类图。



类图-实例讲解

实例4

采购员从供货商处订货，双方需要签订订单，一个采购员可以订多个供货商的货品，一个供货商也可以给多个采购员供货。

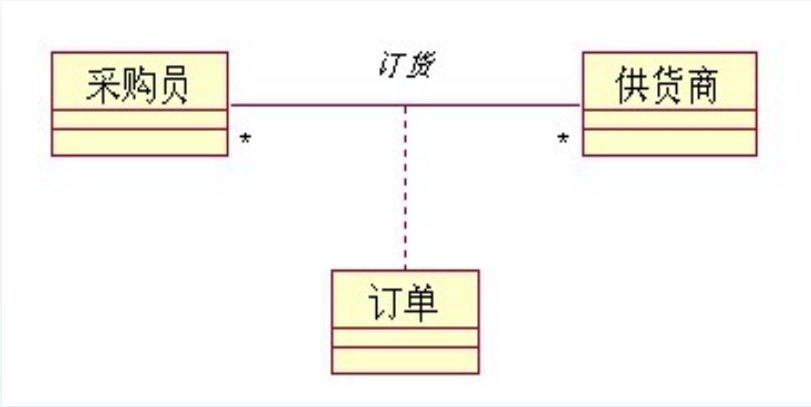


类是否提取完全了，还有没有隐藏的没有提取的类？

类图-实例讲解

实例4

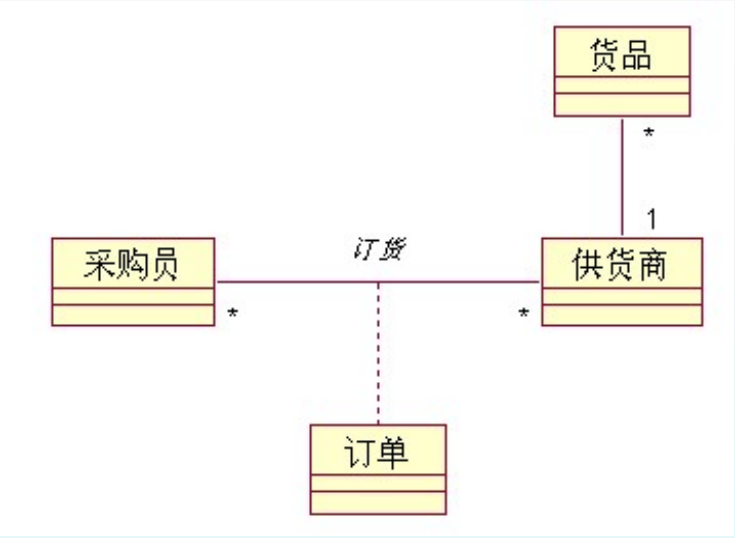
采购员从供货商处订货，双方需要签订订单，一个采购员可以订多个供货商的货品，一个供货商也可以给多个采购员供货。



类图-实例讲解

实例4

采购员从供货商处订货，双方需要签订订单，一个采购员可以订多个供货商的货品，一个供货商也可以给多个采购员供货。

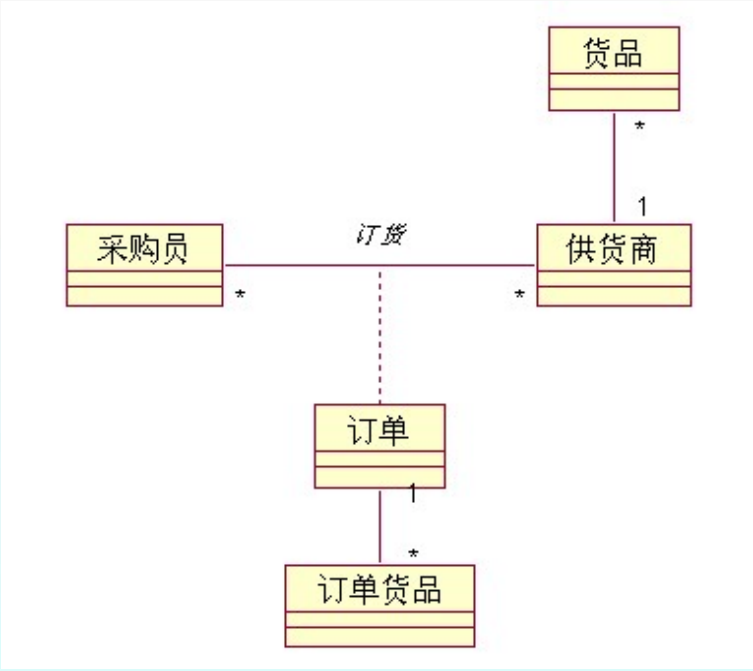


货品是由供应商提供，但订单所订的货品怎么样反映出来？

类图-实例讲解

实例4

采购员从供货商处订货，双方需要签订订单，一个采购员可以订多个供货商的货品，一个供货商也可以给多个采购员供货。



类图-实例讲解

实例5

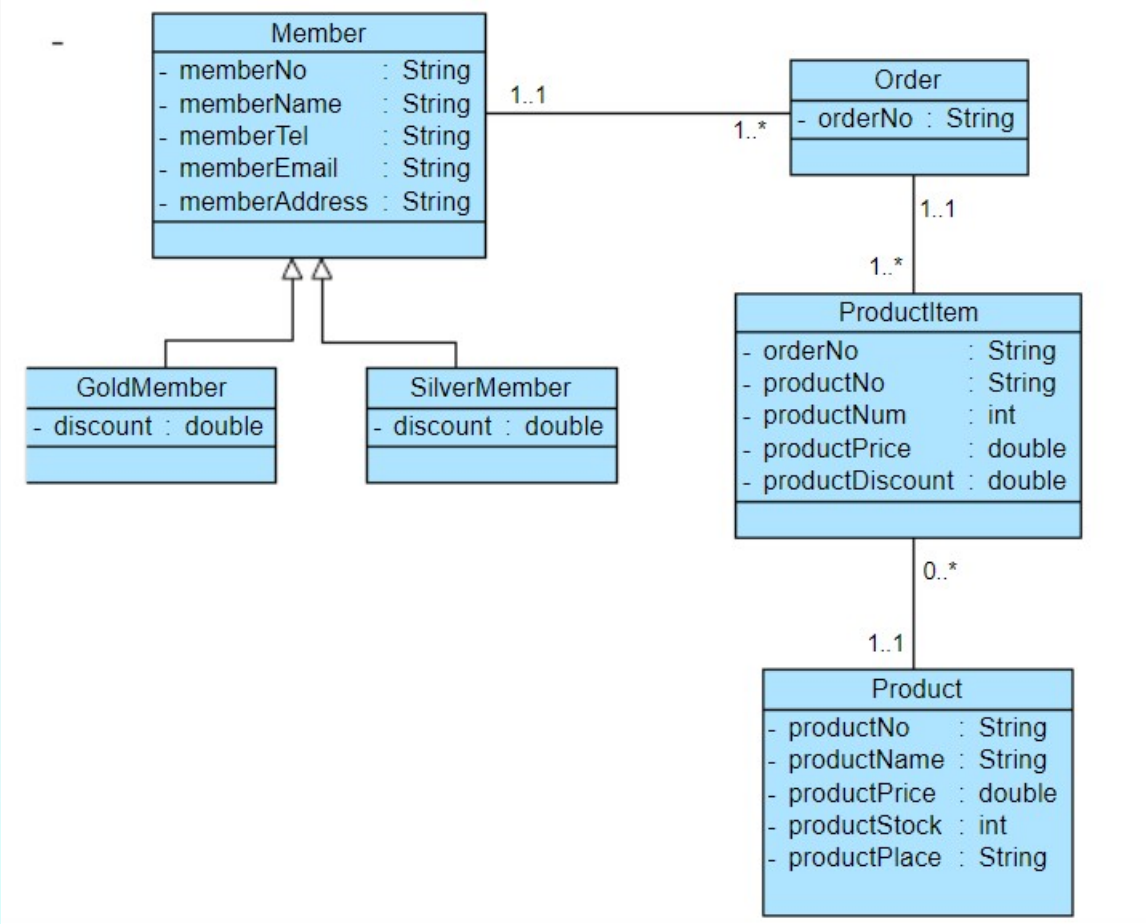
根据以下描述绘制类图：

- 某商场会员管理系统包含一个会员类(Member)，会员的基本信息包括会员编号、会员姓名、联系电话、电子邮箱、地址等；
- 会员可分为金卡会员(GoldMember)和银卡会员(SilverMember)两种，不同类型的会员在购物时可以享受不同的折扣；
- 每个会员可以拥有一个或多个订单(Order)，每一个订单又可以包含至少一条商品销售信息(ProductItem)，商品销售信息包括订单编号、商品编号、商品数量、商品单价和折扣等；
- 每一条商品销售信息对应一类商品(Product)，商品信息包括商品编号、商品名称、商品单价、商品库存量、商品产地等。

类图-实例讲解

实例5

根据以下描述绘制类图：



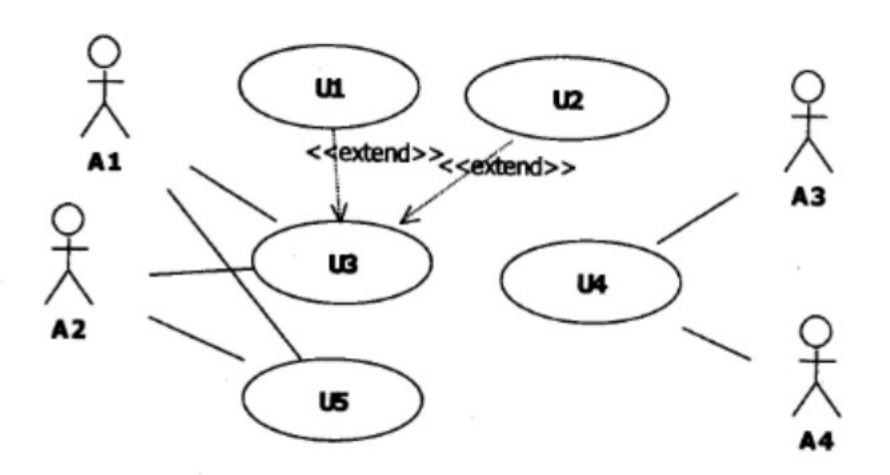
实例

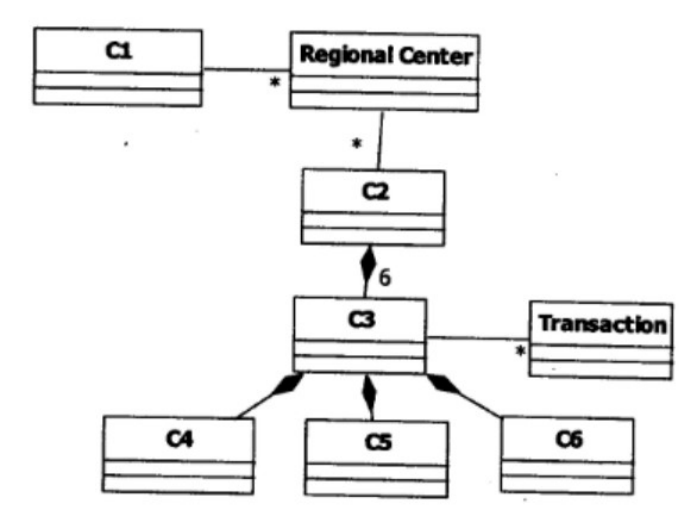
- 某ETC（Electronic Toll Collection，不停车收费）系统在高速公路沿线的特定位置上设置一个横跨道路上空的龙门架（Toll gantry），龙门架下包括6条车道（Traffic lanes），每条车道上安装有雷达传感器（Radar sensor）、无线传输器（Radio transceiver）和数码相机（Digital Camera）等用于不停车收费的设备，以完成正常行驶速度下的收费工作。该系统的基本工作过程如下：
- （1）每辆汽车上安装有车载器，驾驶员（Driver）将一张具有唯一识别码的磁卡插入车载器中。磁卡中还包含有驾驶员账户的当前信用记录。
- （2）当汽车通过某条车道时，不停车收费设备识别车载器内的特有编码，判断车型，将收集到的相关信息发送到该路段所属的区域系统（Regional center）中，计算通行费用，创建收费交易（Transaction），从驾驶员的专用账户中扣除通行费用。如果驾驶员账户透支，则记录透支账户交易信息。区域系统再将交易后的账户信息发送到维护驾驶员账户信息的中心系统（Central system）
- （3）车载器中的磁卡可以使用邮局的付款机进行充值。充值信息会传送至中心系统，以更新驾驶员账户的余额。
- （4）当没有安装车载器或者车载器发生故障的车辆通过车道时，车道上的数码相机将对车辆进行拍照，并将车辆照片及拍摄时间发送到区域系统，记录失败的交易信息；并将该交易信息发送到中心系统。
- （5）区域系统会获取不停车收费设备所记录的交通事件（Traffic events）；交通广播电台（Traffic advice center）根据这些交通事件进行路况分析并播报路况。
- 现采用面向对象方法对上述系统进行分析与设计，分析出对应的用例图和类图。

实例

- 某ETC（Electronic Toll Collection，不停车收费）系统在高速公路沿线的特定位置上设置一个横跨道路上空的龙门架（Toll gantry），龙门架下包括6条车道（Traffic lanes），每条车道上安装有雷达传感器（Radar sensor）、无线传输器（Radio transceiver）和数码相机（Digital Camera）等用于不停车收费的设备，以完成正常行驶速度下的收费工作。该系统的基本工作过程如下：

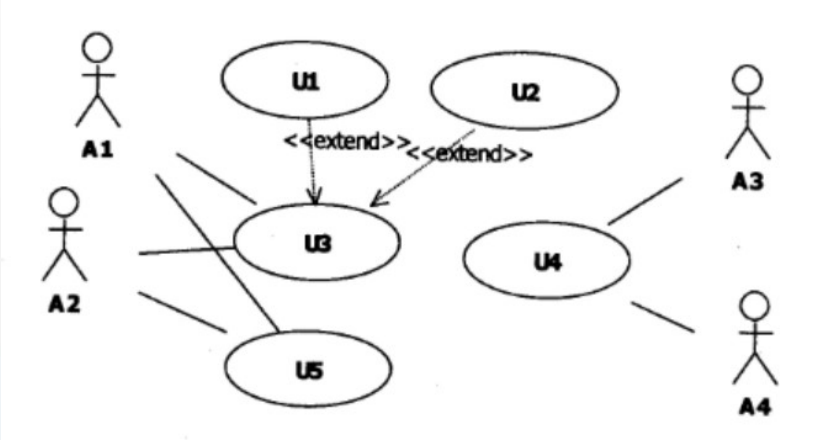
用例名称	说明
Create transaction	记录收费交易
Charge card	磁卡充值
Underpaid transaction	记录透支账户交易信息
Record Illegal use	记录失败交易信息
Record traffic event	记录交通事件

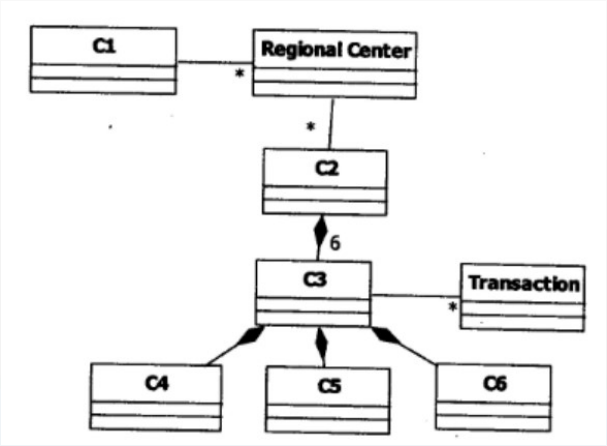




实例

- 某ETC（Electronic Toll Collection，不停车收费）系统在高速公路沿线的特定位置上设置一个横跨道路上空的龙门架（Toll gantry），龙门架下包括6条车道（Traffic lanes），每条车道上安装有雷达传感器（Radar sensor）、无线传输器（Radio transceiver）和数码相机（Digital Camera）等用于不停车收费的设备，以完成正常行驶速度下的收费工作。该系统的基本工作过程如下：





A1：Driver或驾驶员

A2：Central system或中心系统Driver或驾驶员

A3：Traffic advice center或交通广播电台

Regionalcenter或区域系统

A4：Regional center或区域系统

U1：Underpaid transaction

U2：Record Illegal use

U3：Create transaction

U4：Record traffic event

U5：Charge card

C1：Center system

C2：Toll gantry

C3：Traffic lane

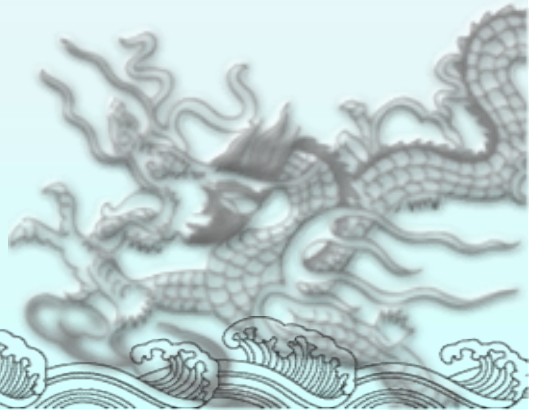
C4：Radar sensors

C5：Digital Camera

C6：Radio transceiver

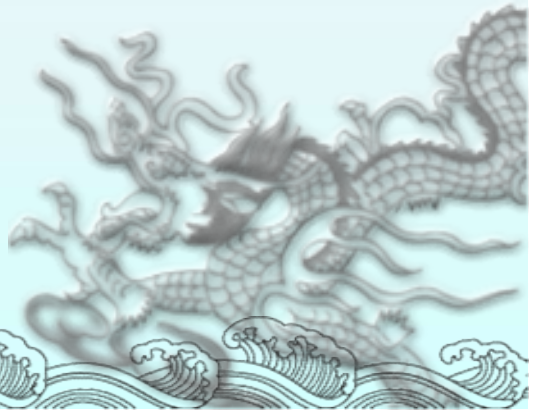
内容提纲

- 类图
  - 相关概念
  - 类间关系
  - 代码映射
  - 实例讲解
- 对象图
  - 相关概念
  - 实例讲解



对象图-相关概念

- 对象图(Object Diagram) 是显示了一组对象和它们之间的关系。使用对象图来说明数据结构、类图中的类或组件等实例的快照。对象图和类图一样，反映了系统的静态过程，但它是以实际的或原型化为基础来表达对象间的关系。
- 对象图显示某时刻对象和对象之间的关系。一个对象图可看成一个类图的特殊实例，实例和类可在对象图中同时表示。



对象图-相关概念

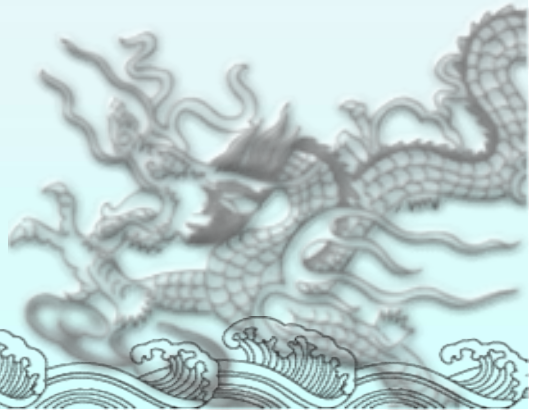
1．对象

对象是一件事、一个实体、一个名词，是客观存在的事物。一些对象是活的，一些对象不是。现实世界中的对象有汽车、人、房子、桌子、狗、支票簿或雨衣。

所有的对象都有属性，例如汽车有厂家、型号、颜色和价格。狗有种类、年龄、颜色和喜欢的玩具、对象还有行为（behavior）：汽车可以从一个地方移动到另一个地方，狗会叫。

2．对象的特点：

对象具有状态、行为和标识三个特点。

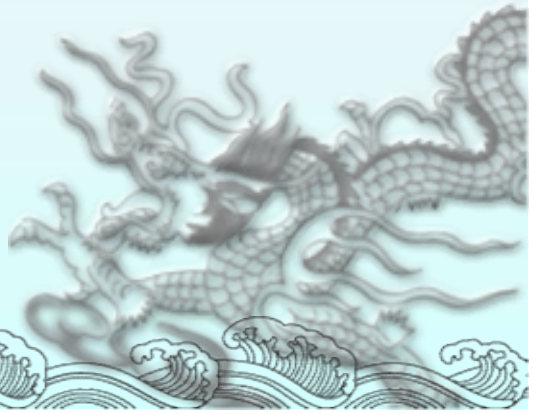


对象图-相关概念

状态：对象的状态指对象在某一时刻，对象所有属性值的集合。

行为：没有一个对象是孤立存在的，对象可以被操作，也可以操作别的对象。而行为就是一个对象根据它的状态改变和消息传送所采取的行动和所做出的反应 。

标识：为了将一个对象与其它所有对象区分开来，我们通常会给它起一个“标识”。

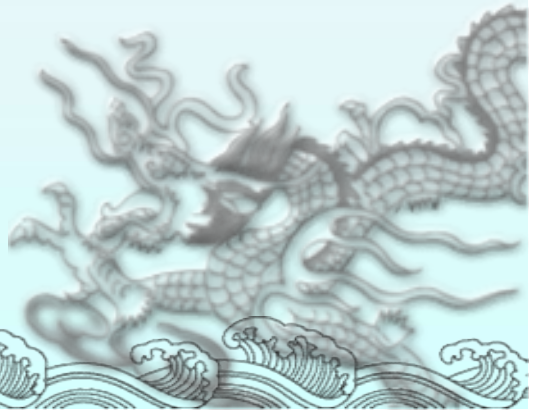


对象图-相关概念

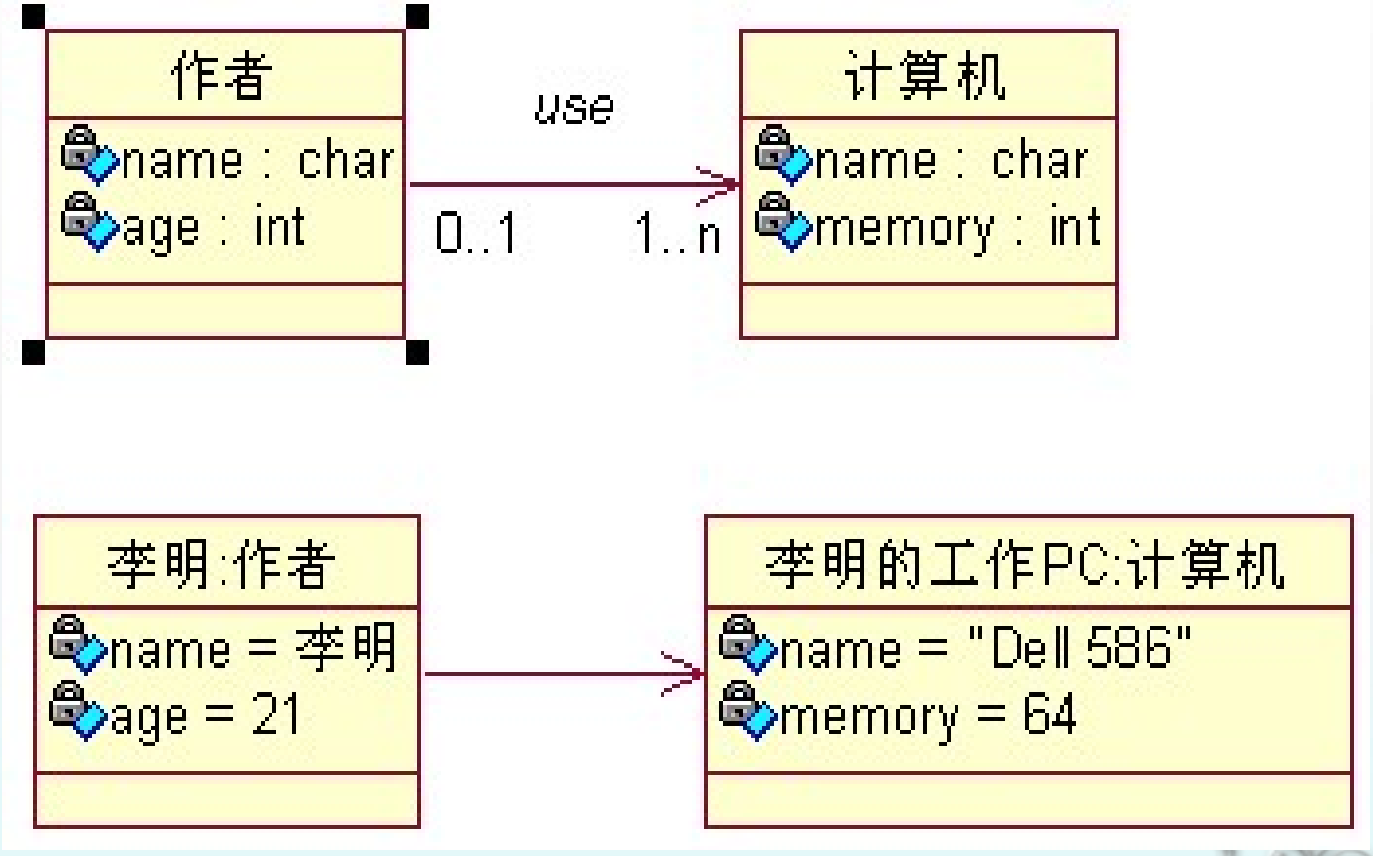
3．对象与类的区别

(1)对象是一个存在于时间和空间中的具体实体，而类是一个模型，该模型抽象出对象的“本质”：一组公共属性和一组公共方法。

(2)类是静态的，对象是动态的；类是一般化，对象是个性化；类是定义，对象是实例；类是抽象、对象是具体 。



对象图-相关概念


作者
name : char
age : int
use
0..1
1..n
计算机
name : char
memory : int
李明:作者
name = 李明
age = 21
李明的工作PC:计算机
name = "Dell 586"
memory = 64

对象图-相关概念

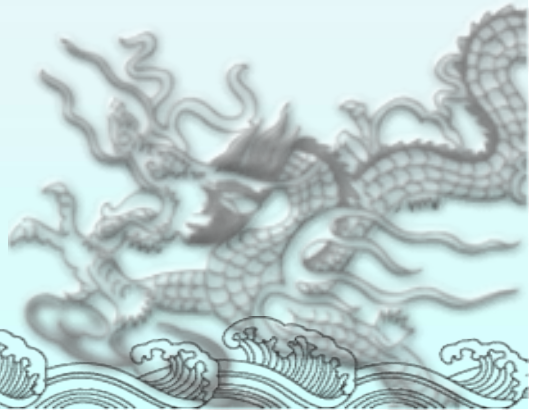
- 对象图
  - 对象图是描述对象及其关系的图。对象图可以看作类图在某一时刻的实例。
  - 几乎使用与类图完全相同的标识。
  - 他们的不同点在于，对象图显示类的多个对象实例，而不是实际的类。由于对象存在生命周期，因此对象图只能在系统某一时间段存在。

对象图-相关概念

对象图-相关概念

- 对象图的作用
  - 对象图常用来描述业务或软件系统在某一时刻，对象的组成、结构和关系。

- 对象图的组成元素
  - 组成对象图的元素有：对象、链、注释、约束。



对象的表示

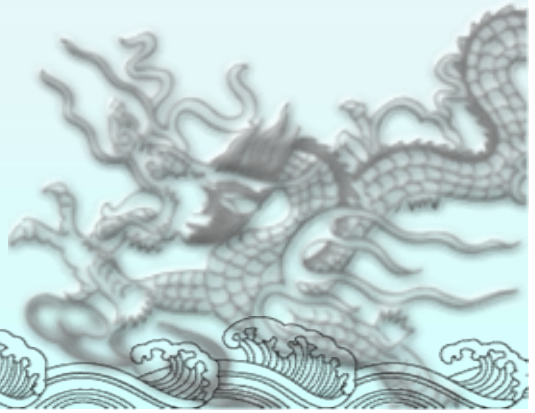
UML中，表示一个对象，主要是标识它的名称、属性。对象由一个矩形表示，它包含2栏，在第一栏写入对象名，在第二栏列出属性名及属性值，格式如：”属性名=属性值”

对象名有下面三种表示格式，不同点在于第一栏表示对象的格式不同：

（1）对象名：类名

对象名在前，类名在后，用冒号来连接。对象名和类名都加下划线。

李小平：Person
name = “李小平” **birthday = 21 October 1983**

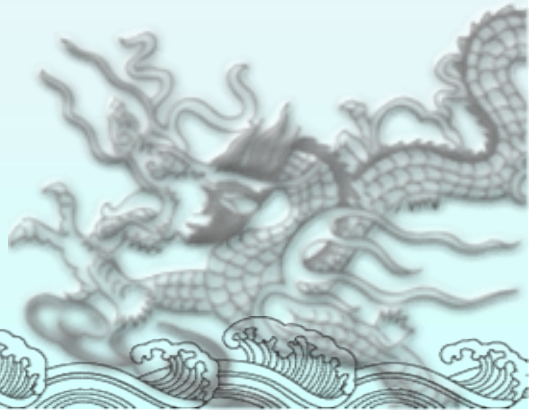


对象的表示

（2）：类名

这是对匿名对象的表示方法。这种格式用于尚未给对象取名的情况，前面的冒号不能省略。

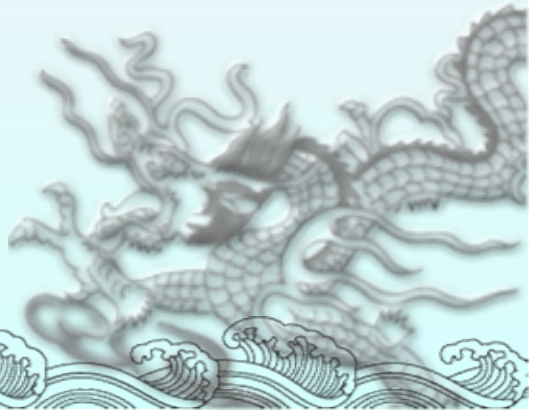
：Person
name = “ ” **birthday = 21 October 1983**



（3）对象名

省略格式，即省略掉类名。只有对象名，对象名必须加下划线。

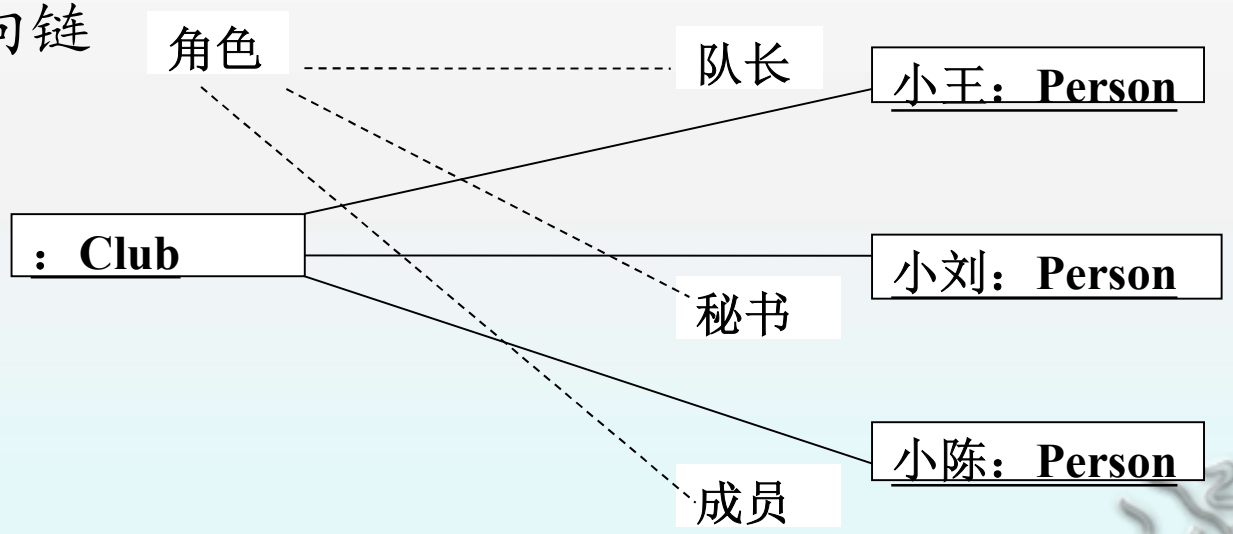
李小平
name = “李小平” **birthday = 21 October 1983**



链的表示

- 链是两个对象间的语义关系。关联是两个类间的关系。就象对象是类的实例一样，链是关联的实例。链分单向链和双向链。

1．双向链



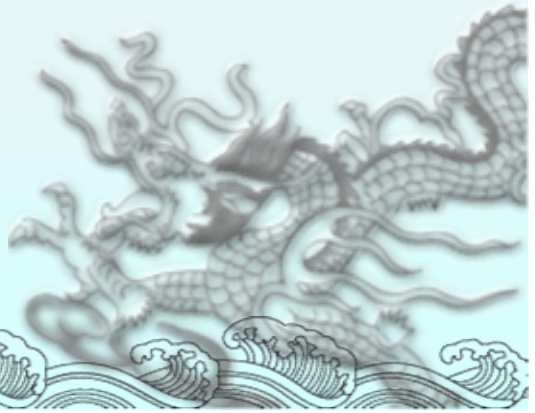
链的表示

2. 单向链

阅读对象图的方法

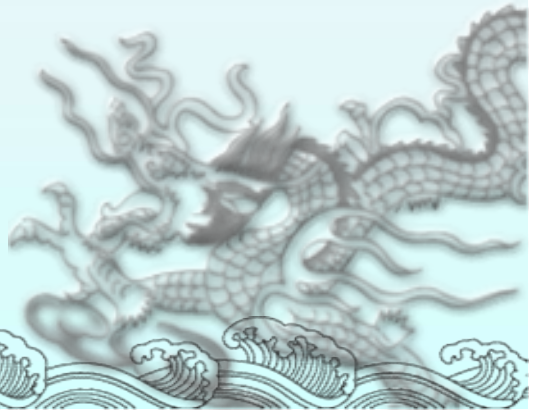
对象图中，对象间的关系称为链。阅读对象图的方法：

(1) 找出图中所有的类
(2) 了解每个对象的语义
(3) 了解对象之间连接含义

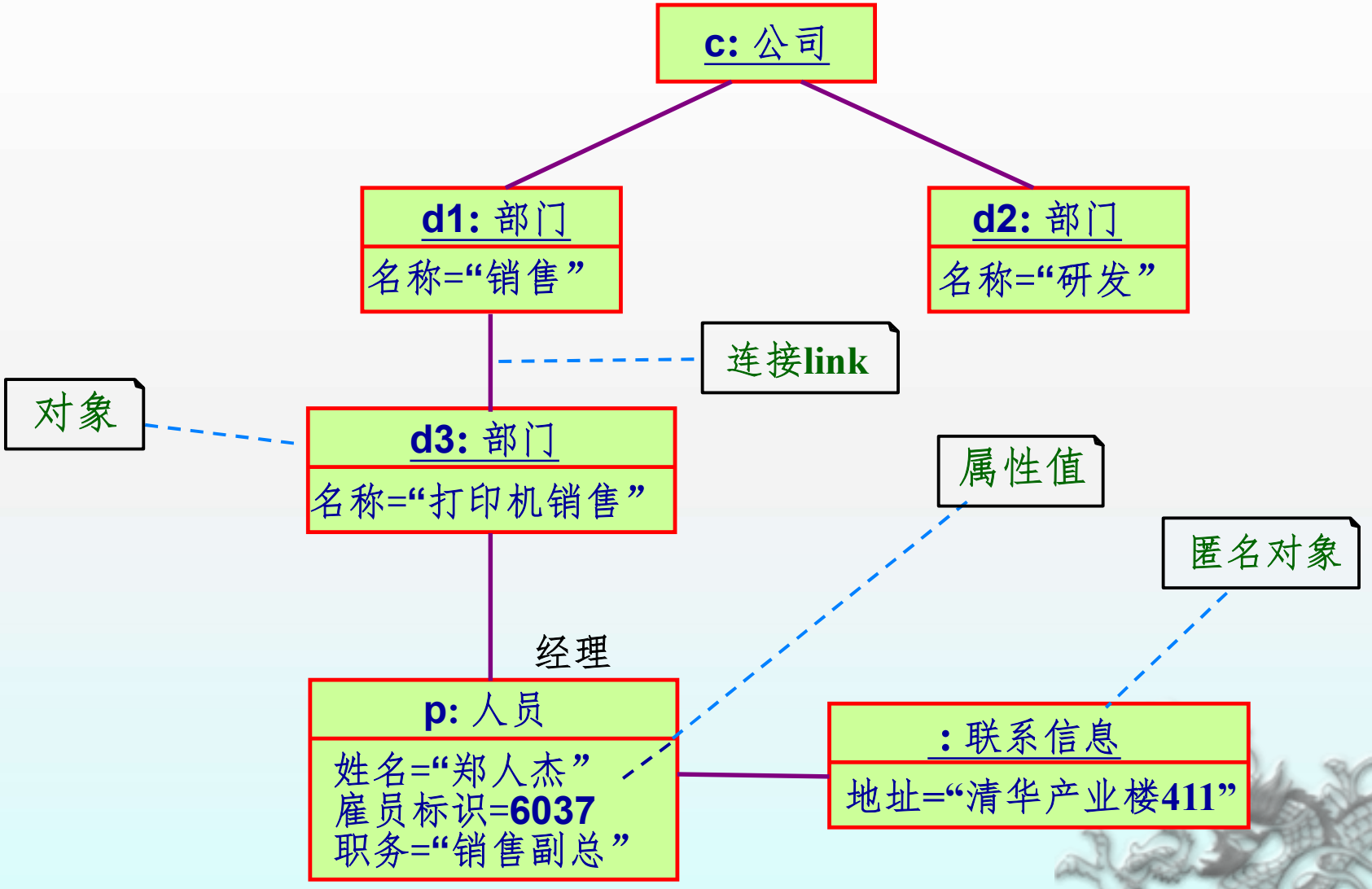


内容提纲

- 类图
  - 相关概念
  - 类间关系
  - 代码映射
  - 实例讲解
- 对象图
  - 相关概念
  - 实例讲解



对象图的实例


c: 公司
d1: 部门
名称="销售"
d2: 部门
名称="研发"
连接link
对象
d3: 部门
名称="打印机销售"
属性值
匿名对象
经理
p: 人员
姓名="郑人杰"
雇员标识=6037
职务="销售副总"
: 联系信息
地址="清华产业楼411"

习题

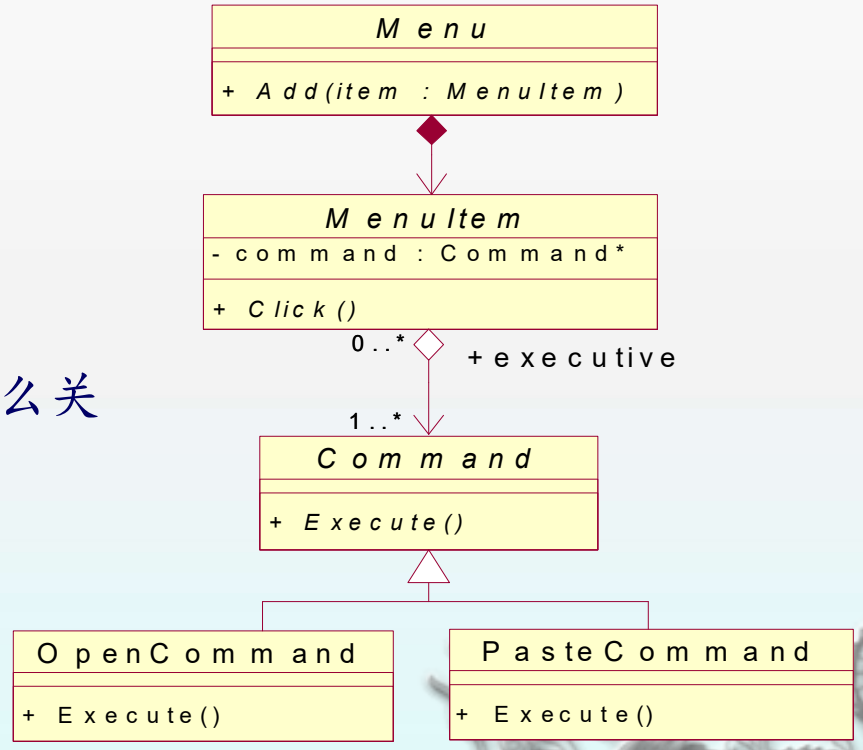
※ 右下图描述了菜单(Menu)、菜单项(MenuItem)、抽象命令类(Command)和具体命令类(OpenCommand, PasteCommand)之间的关系，完成1-4题

(1)哪两个类之间存在组合关系

① Menu、MenuItem
② MenuItem、Command
③ Command、OpenCommand
④ Command、PasteCommand

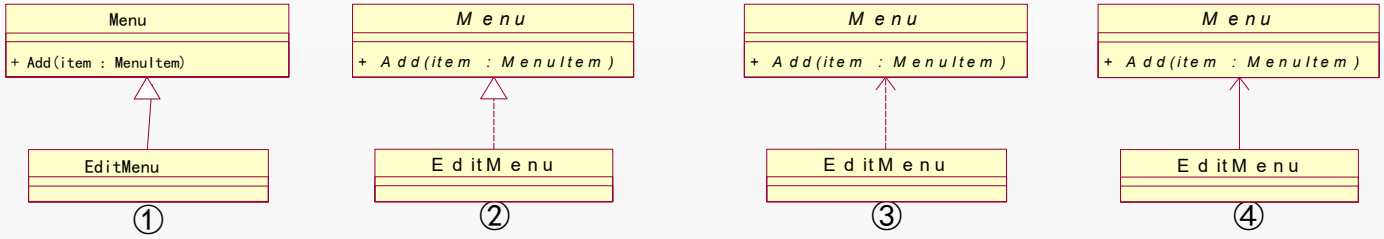
(2)OpenCommand和PasteCommand是什么关系

① 组合
② 泛化
③ 聚合
④ 没关系



习题

(3) 编辑菜单(EditMenu)是一种菜单，下面哪个图较好的描述了两者之间的关系



(4) 下面哪份代码(C++)最接近于图中对MenuItem的描述

```
class MenuItem
{
private:
  virtual void Click() =0;
public:
  Command* command;
};
```

①

```
class MenuItem
{
public:
  virtual void Click() = 0;
private:
  Command* command;
};
```

②

```
class MenuItem
{
private:
  virtual void Click() = 0;
  void undo();
public:
  Command* command;
};
```

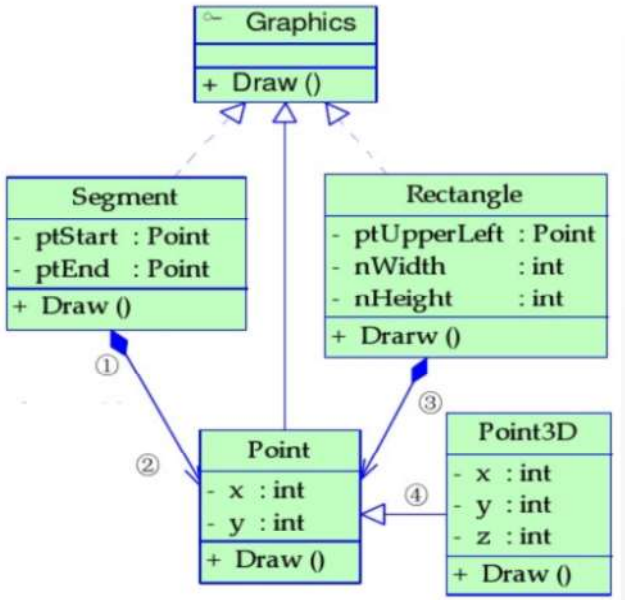
③

```
class menuitem
{
public:
  virtual void Click() =
0;0;
private:
  Command* command;
};
```

④

习题

※ 右图描述了图形接口(Graphics)、线段(Segment)、矩形(Rectangle)、点(Point)和三维点(Point3D)之间的关系， 完成5-7题



(5) 下面哪个关系没有在图中出现

①关联　　②泛化　　③实现　　④依赖

(6) 下面对图中①②③④四处的多重性的描述哪个不正确

① 0...*　　②1　　③0...*　　④1

习题

(7)下面哪份代码(Java)最接近于图中对Segment的描述

```
public class Segment implements
Graphics
{
privatc void Draw();
public Point ptStart;
public Point ptEnd;
}
```
①

```
public class Segment extends
Graphics
{
public void Draw();
private Point ptStart;
private Point ptEnd;
}
```
②

```
public class Segment implements
Graphics
{
private Point ptStart;
private Point ptEnd;
public void Draw();
}
```
③

```
public class segment implements
graphics
{
public void Draw();
private Point ptStart;
private Point ptEnd;
}
```
④

作业

1、简述类图中关联类的作用，并举例说明。

2、简述类与类之间的关系，并举例说明。