第7章：类图

本章内容

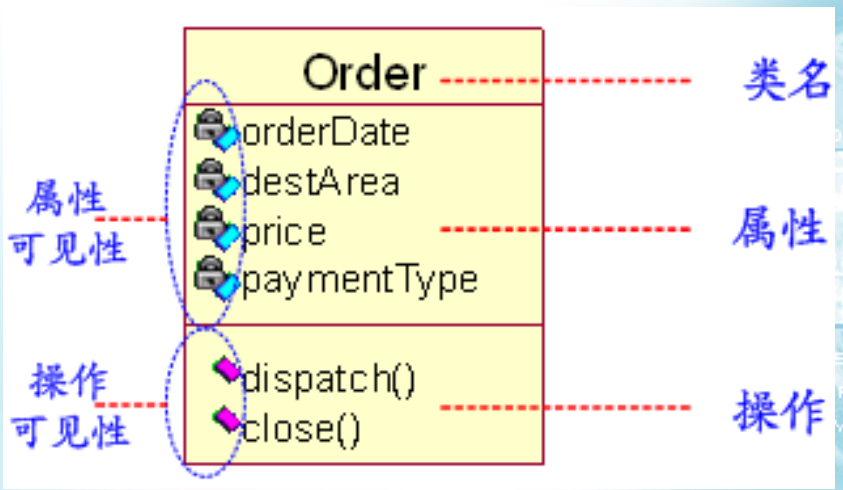
- 类的UML表示
- 如何阅读类图
- 类的其他高级概念
- 如何绘制类图
- 类图的应用

本章内容

- 类的UML表示
- 如何阅读类图
- 类的其他高级概念
- 如何绘制类图
- 类图的应用

如何用UML表示一个类

- 名称：每个类都有一个惟一的名称，通常采用CamelCase格式表示
- 属性：是已被命名的类的特性，它描述该类实例中包含的信息
- 操作：是类所提供的服务，它可以由类的任何对象请求以影响其行为
- 属性名和操作名也通常采用CamelCase格式表示，只不过首字母通常为小写。

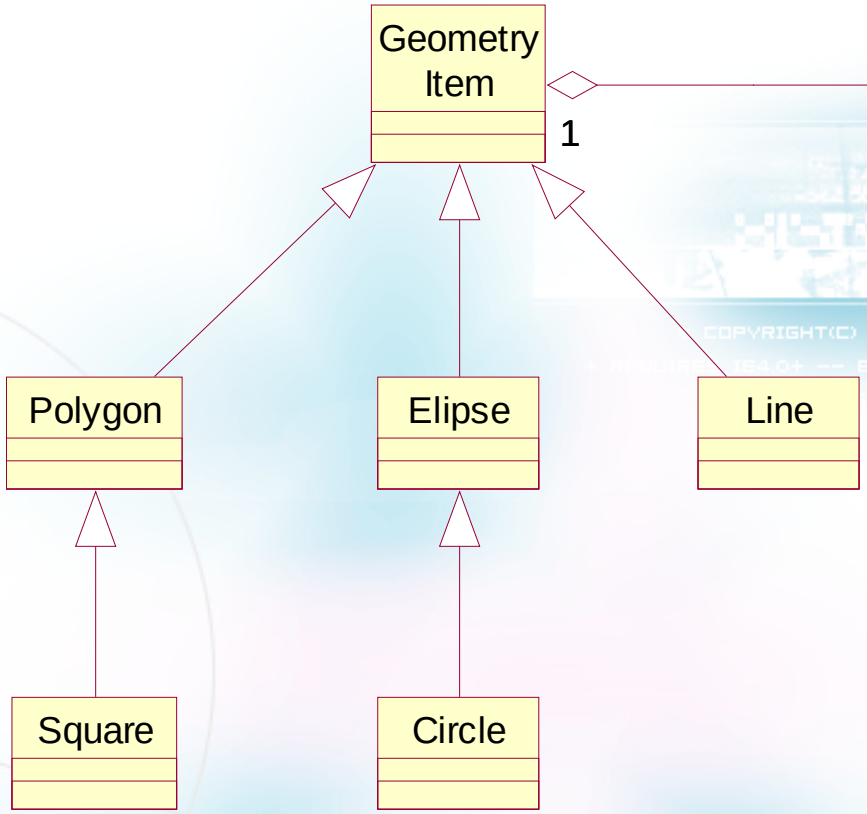


思考题:复习类之间的关系

- 用UML图形表示一组几何对象。组中的父类叫“几何单项”（GeometryItem），由此衍生出线（Line）、圆（Circle）、正方形（Square）、椭圆（Elipse）和多边形（Polygon）子类。椭圆有两个焦点，它们重合时就成了圆。正方形是多边形的一种。所有的GeometryItem都是由点（Point）组成的。

思考题

- **Answer:**



在设计模式中应用UML展现模式结构

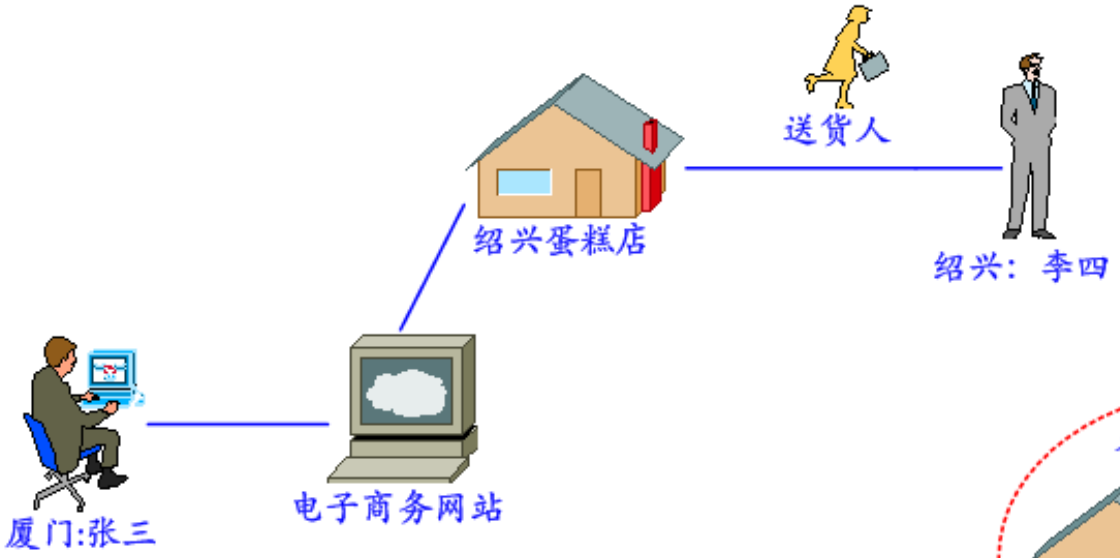
- 单例模式
- 工厂模式
- 组合模式

……

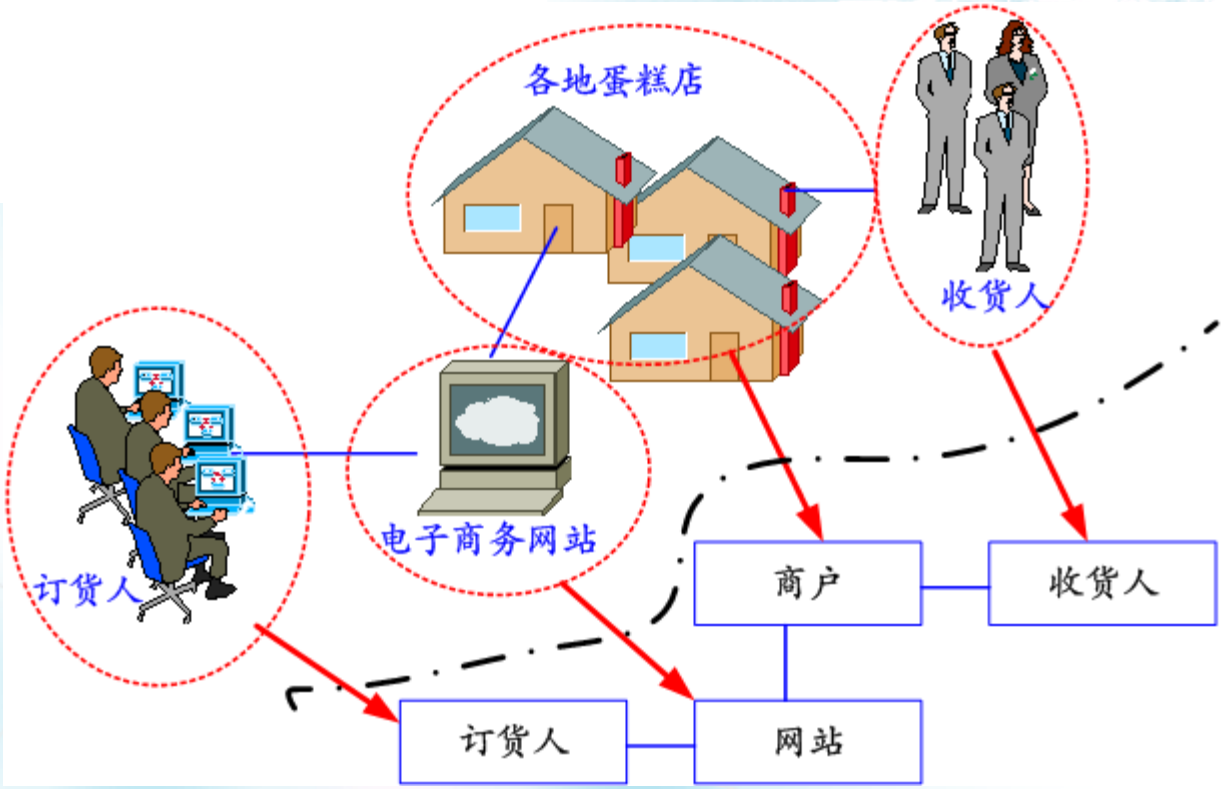
本章内容

- 类的UML表示
- 如何阅读类图
- 类的其他高级概念
- 如何绘制类图
- 类图的应用

面向对象思想



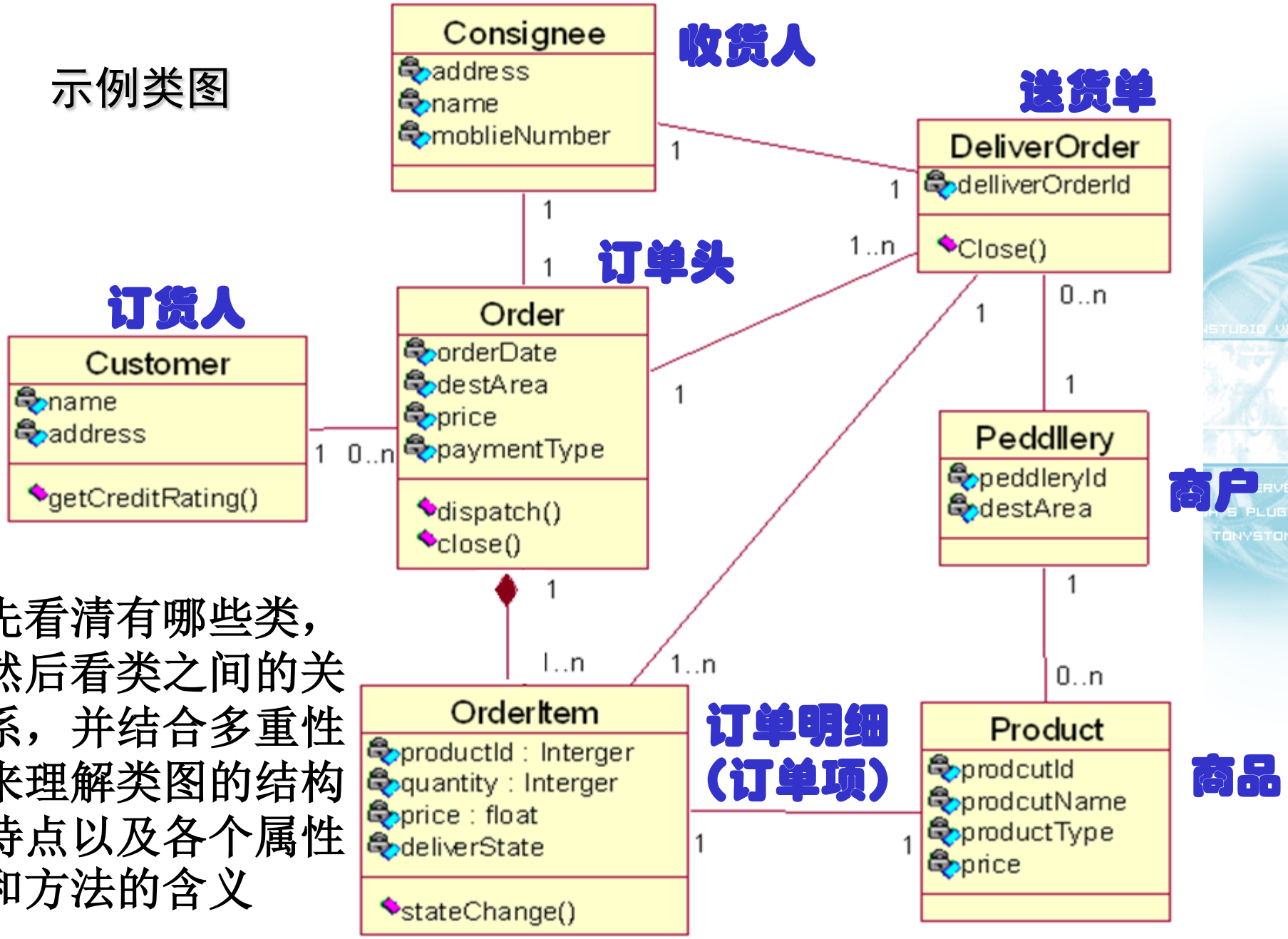
张三通过电子商务网站给李四送一个蛋糕



面向对象思想

- 每个对象都扮演了一个角色，并为其它成员提供特定的服务或执行特定的行为。
- 在面向对象世界中，行为的启动是通过将“消息”传递给对此行为负责的对象来完成的；同时还将伴随着执行要求附上相关的信息（参数）；而收到该消息的对象则会执行相应的“方法”来实现需求
- 用类和对象表示现实世界，用消息和方法来模拟现实世界的核心思想

示例类图



先看清有哪些类，然后看类之间的关系，并结合多重性来理解类图的结构特点以及各个属性和方法的含义

读图过程

- 读出类：图中共有7个类，Order、OrderItem、Customer、Consignee、DeliverOrder、Peddlery、Prodcut

- 读出关系：从图中关系最复杂（也就是线最密集）的类开始阅读，本图中最复杂的就是Order类。

  1）OrderItem和Order之间是组合关系，根据箭头的方向可知Order包含了OrderItem。

  2）Order类和Customer、Consignee、DeliverOrder是关联关系。也就是说，一个订单和客户、收货人、送货单是相关的。

读图过程

- 多重性：用来说明关联的两个类之间的数量关系

源类及多重性	目标类及多重性	分析
Customer(1)	Order(0…n)	订单是属于某个客户的，网站的客户可以有0个或多个订单
Order(1)	Consignee(1)	每个订单只能够有一个收货人
Order(1)	OrderItem(1…n)	订单是由订单项组成的，至少要有一个订单项，最多可以有n个
Order(1)	DeliverOrder(1…n)	一个订单有一个或多个送货单
DeliverOrder(1)	OrderItem(1…n)	一张送货单对应订单中的一到多个订单项
DeliverOrder(1)	Consignee(1)	每张送货单都对应着一个收货人
Peddlery(1)	DeliverOrder(0…n)	每个商户可以有相关的0个或多个送货单
OrderItem(1)	Product(1)	每个订单项中都包含着唯一的一个产品
Peddlery(1)	Prodcut(0…n)	产品是属于某个商户的，可以注册0到多个产品

说明：系统根据订单项的产品所属的商户，将其分发给商户，拆成了多个送货单！

读图过程

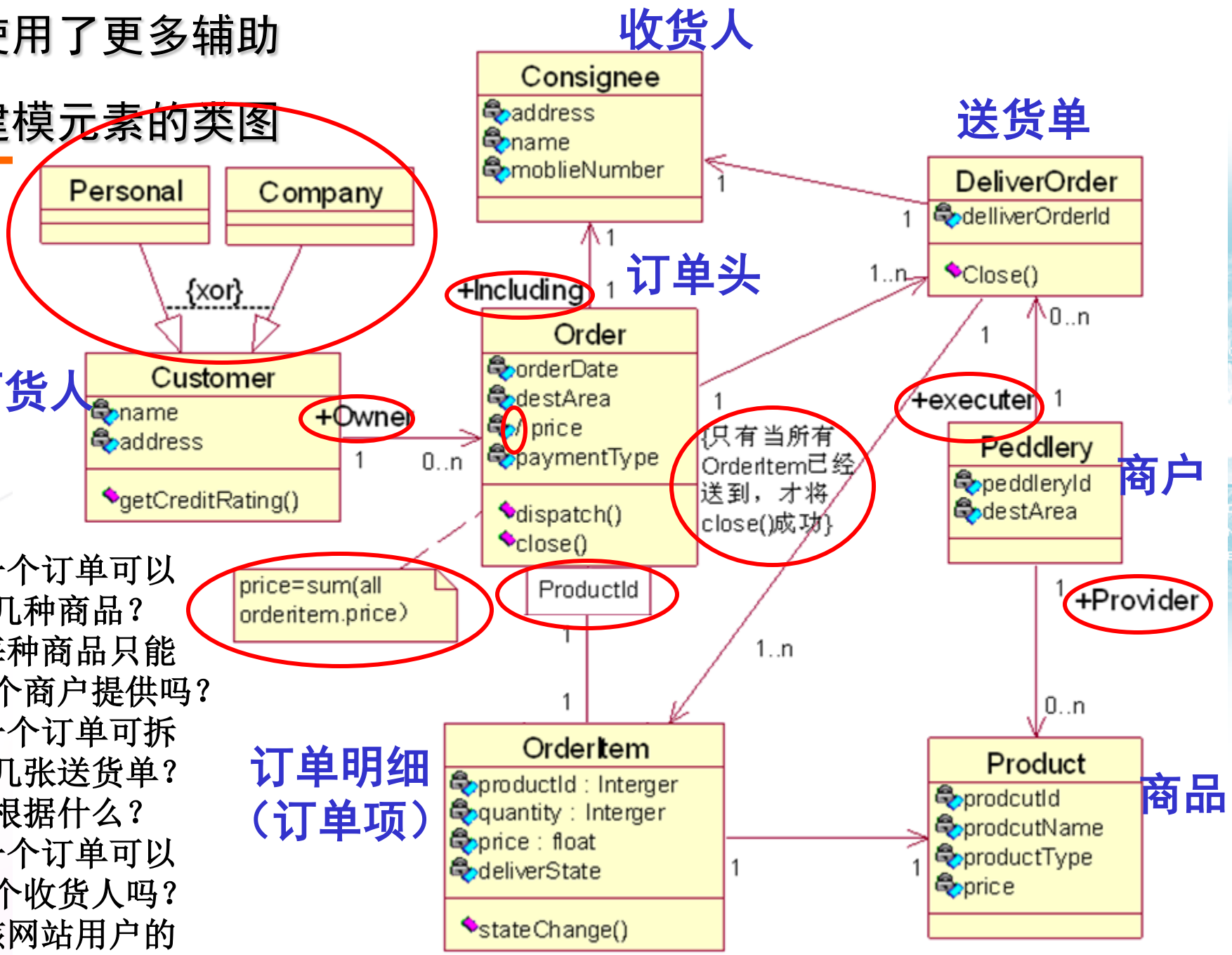
- 理解方法和属性
  - Order类的两个方法：dispatch()和close()，分别实现“分拆订单生成送货单”和“完成订单”。
  - DeliveOrder类的Close()方法，表示“完成送货”。
  - OrderItem类的stateChange()方法和deliverState属性，是用来改变其“是否交给收货人”标志位的。

读图过程

- 过程分析

  先调用Order的dispatch()方法，它将根据其包含的OrderItem中产品信息，来按供应商户分拆成若干个DeliverOrder。商户登录系统后就可以获取其DeliverOrder，并在送货完后调用close()方法。这时，就将调用OrderItem的stateChange()方法来改变其状态。同时再调用Order的close()方法，判断该Order的所有的OrderItem是否都已经送到了，如果是就将其真正close()掉！

使用了更多辅助建模元素的类图



(1)一个订单可以订购几种商品？
(2)每种商品只能由一个商户提供吗？
(3)一个订单可拆分为几张送货单？拆分根据什么？
(4)一个订单可以有多个收货人吗？
(5)该网站用户的登录身份有几种？

增强的辅助建模元素

- 导航箭号：类的实例之间只能沿着导航箭头的方向传递 。例如：在Order中可以获取其相应的Consignee，而从Consignee中是无法了解与其相关的Order的。

- 角色名称：Customer端有一个“+Owner”字符串，这表示Customer扮演的角色是Owner，也能对关联进行命名

- 导出属性：是指可以根据其他值计算出来的特性，这种属性应在其名称前加上一个“/”符号。

增强的辅助建模元素

- 限定符：在Order和OrderItem之间的组合关系中，Order这端多了一个方框，里面写着“ProductId”。它在UML中称为限定符，存在限定符的关联称为限定关联。它用来表示某种限定关系。在本例中，它的用途是说明：对于一张订单，每一种产品只能用一个订单项。

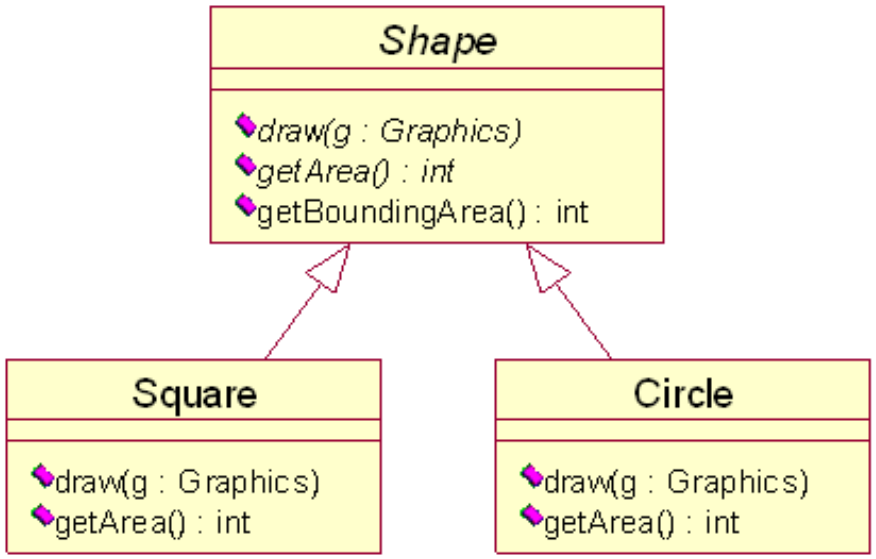
- 约束：用来说明规则，如{xor}

本章内容

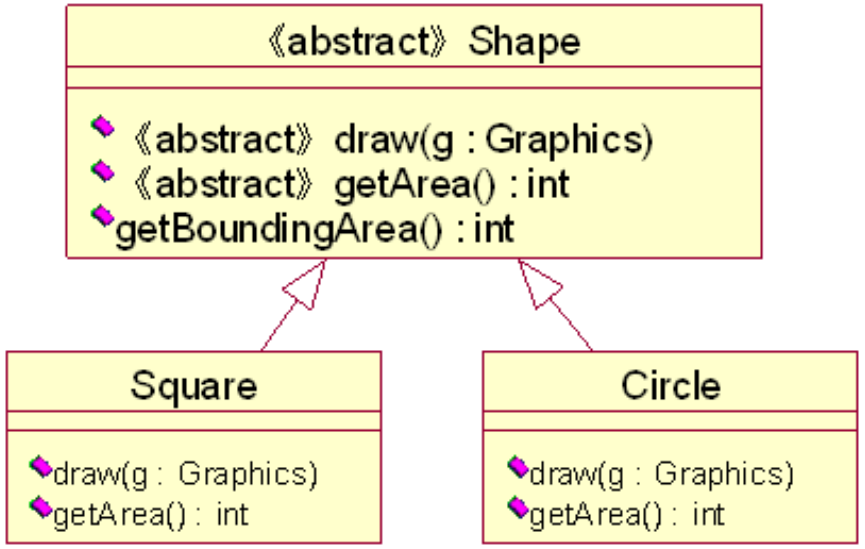
- 类的UML表示
- 如何阅读类图
- **类的其他高级概念**
- 如何绘制类图
- 类图的应用

抽象类

- 抽象类是一种不能够被直接实例化的类，也就是说不能够创建一个属于抽象类的对象



UML标准表示法（适用于蓝图）



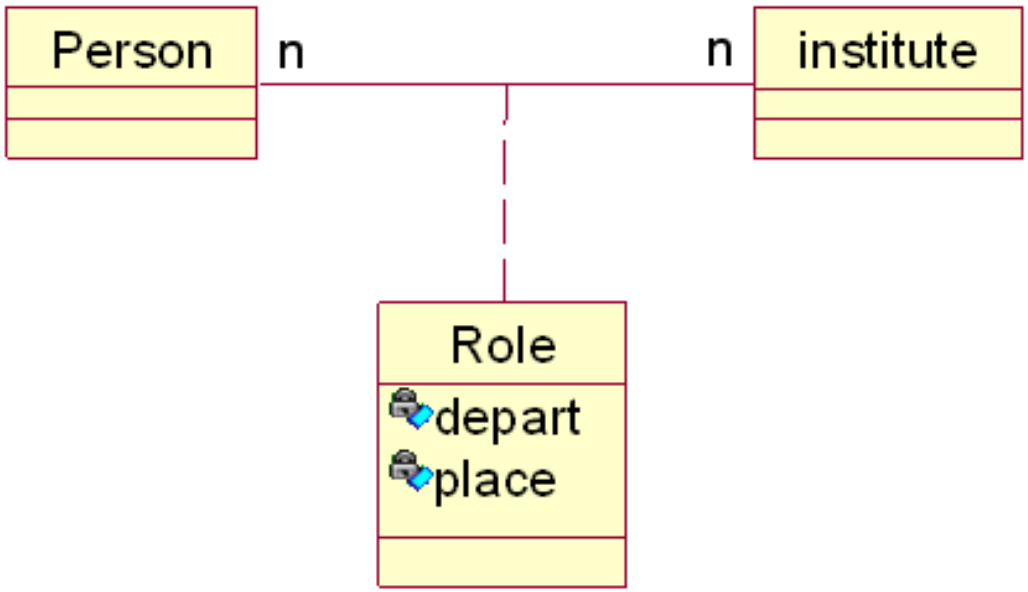
构造型表示法（适用于草图）

接口

- 接口则是一种类似于抽象类的机制，它是一个没有具体实现的类

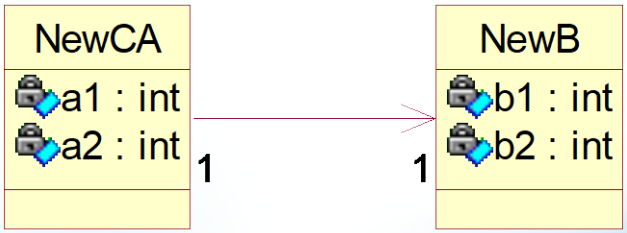
关联类

- 关联类即是关联也是类，它不仅像关联那样连接两个类，而且还可以定义一组属于关系本身的特性 。
- 如果在具有关联关系的类中，存在着一个属性放哪个类都不合适的情况，就要考虑使用关联类。



UML中关联关系与Java程序实现

- 一对一的单向关联



NewCA.java

```
//Source file: F:\\j\\NewCA.java
public class NewCA
{
    private int a2;
    private int a1;
    public NewB theNewB;

    /**
    @roseuid 4608DE99038A
     */
    public NewCA()
    {

    }
}
```

NewCA.java | NewB.java

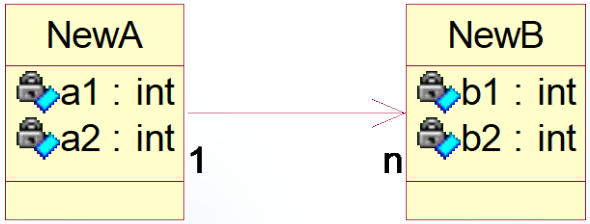
NewB.java

```
//Source file: F:\\j\\NewB.java
public class NewB
{
    private int b2;
    private int b1;

    /**
    @roseuid 4608DE9903B9
     */
    public NewB()
    {

    }
}
```

NewCA.java | NewB.java

UML中关联关系与Java程序实现

- 一对多的单向关联



NewA.java

```
//Source file: F:\\j\\NewA.java
public class NewA
{
   private int a2;
   private int a1;
   public NewB theNewB[];

   /**
   @roseuid 4608E0E4037A
    */
   public NewA()
   {

   }
}
```

NewA.java NewB.java

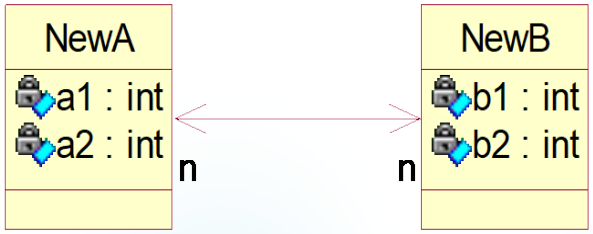
NewB.java

```
//Source file: F:\\j\\NewB.java
public class NewB
{
   private int b2;
   private int b1;

   /**
   @roseuid 4608E0E403B9
    */
   public NewB()
   {

   }
}
```

NewA.java NewB.java

UML中关联关系与Java程序实现

- 双向关联



NewA.java

```
//Source file: F:\\NewA.java
public class NewA
{
   private int a1;
   private int a2;
   public NewB theNewB[];
   /**
   @roseuid 4608FDF10000
    */
   public NewA()
   {

   }
}
```

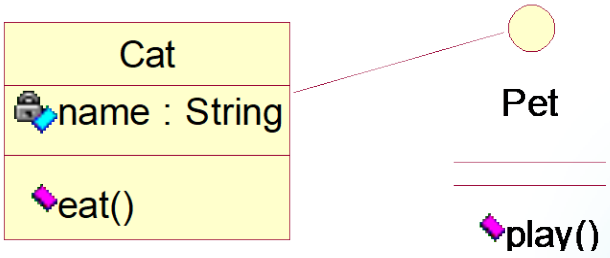
NewB.java | NewA.java

NewB.java

```
//Source file: F:\\NewB.java
public class NewB
{
   private int b1;
   private int b2;
   public NewA theNewA[];
   /**
   @roseuid 4608FDF003A9
    */
   public NewB()
   {

   }
}
```

NewB.java | NewA.java

UML中实现关系与Java程序实现



Pet.java

```
//Source file: F:\\Pet.java
public interface Pet
{
   /**
   @roseuid 460905240128
    */
   public void play();
}
```

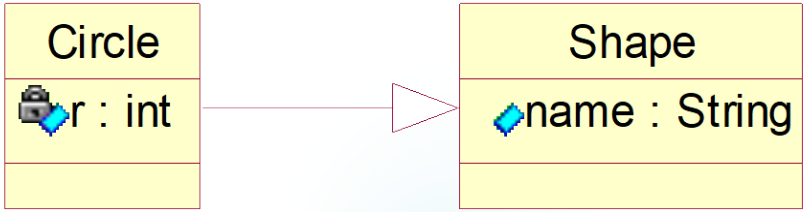
Cat.java | Pet.java

Cat.java

```
//Source file: F:\\Cat.java
public class Cat implements Pet
{
   private String name;
   /**
   @roseuid 46090546002E
    */
   public Cat()
   {

   }
   /**
   @roseuid 460903E0000F
    */
   public void eat()
   {

   }
   /**
   @roseuid 46090546003E
    */
   public void play()
   {

   }
```

Cat.java | Pet.java

UML中泛化关系与Java程序实现



Shape.java

```
//Source file: F:\\Shape.java
public class Shape
{
   public String name;

   /**
   @roseuid 46090339036B
    */
   public Shape()
   {

   }
}
```

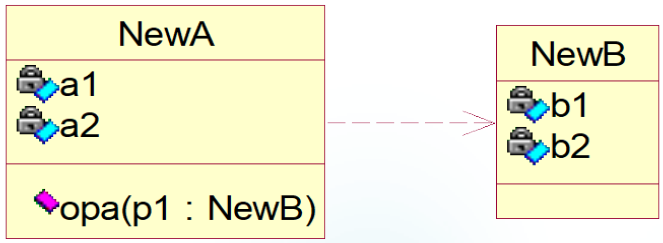
Shape.java | Circle.java

Circle.java

```
//Source file: F:\\Circle.java
public class Circle extends Shape
{
   private int r;

   /**
   @roseuid 460903390399
    */
   public Circle()
   {

   }
}
```

Shape.java | Circle.java

UML中依赖关系与Java程序实现



NewA.java

```
//Source file: F:\\NewA.java
public class NewA
{
   private int a1;
   private int a2;
   /**
   @roseuid 460901FE009C
    */
   public NewA()
   {

   }
   /**
   @param p1
   @roseuid 460901640232
    */
   public void opa(NewB p1)
   {

   }
}
```

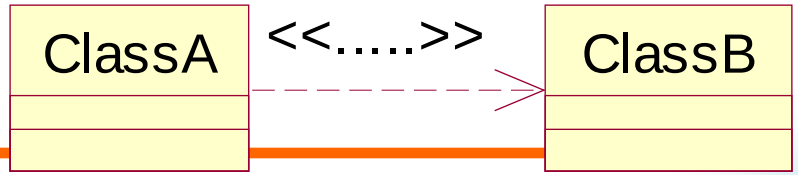
NewA.java NewB.java

NewB.java

```
//Source file: F:\\NewB.java
public class NewB
{
   private int b1;
   private int b2;

   /**
   @roseuid 460901FE00DA
    */
   public NewB()
   {

   }
}
```

NewA.java NewB.java

UML中依赖关系与Java程序实现



依赖构造型	含义	例子程序

《create》	表明目标对象是由源对象创建的，目标对象创建后将传递给系统其他部分。	public class ClassA{
		public ClassB createB() {
		return new ClassB();
		}
		}

《local》或《call》	源类对象创建目标类对象实例，并将该实例包含在一个局部变量中。例如右边的例子中，将赋给一个名为test的变量	public class ClassA{
		public void testMethod() {
		ClassB test=new ClassB();
		}
		}

《parameter》	源类对象通过它的某个成员函数的参数得以访问目标类对象实例。它的意思是指：类ClassA的操作需要类ClassB的实例作为参数，或返回类ClassB的实例。	public class ClassA{
		public void testMethod(ClassB test) {
		// use b;
		}
		}

《delegate》	源类对象把一个对于成员函数的调用传递给目标类对象。这是现代编程语言和设计模式中很常用的一种机制，但这不属于UML的标准关系。	public class ClassA{
		private ClassB objectB;
		public void testMethod() {
		objectB.testMethod();
		}
		}

对象约束语言OCL

- 环境与约束

每个OCL表达式都必须是针对某个元素的，因此在OCL表达式前必须说明它针对元素（这就称为环境），其表示方法有2种：

1）“context Object inv：”，其中Object是OCL表达式针对的建模元素名称；

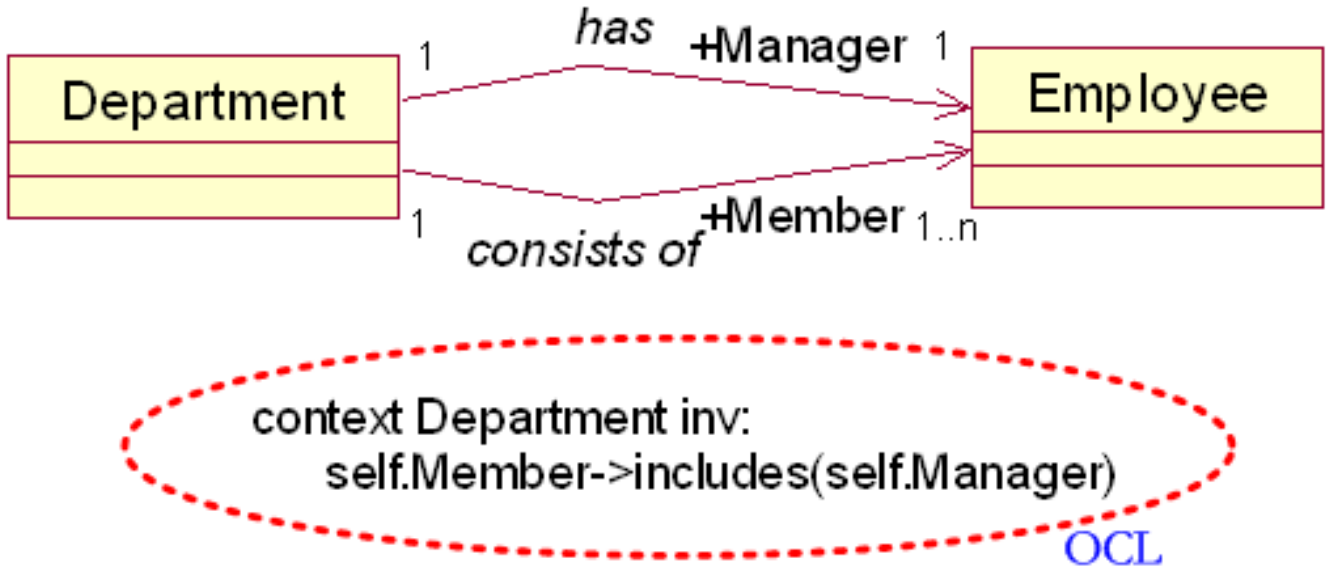
2）“ Object ”，其中Object是OCL表达式针对的建模元素名称。

当声明了环境之后，就可以用self来引用它的变量

对象约束语言

- 子集约束：

  若要表示“经理一定是部门成员之一”



对象约束语言

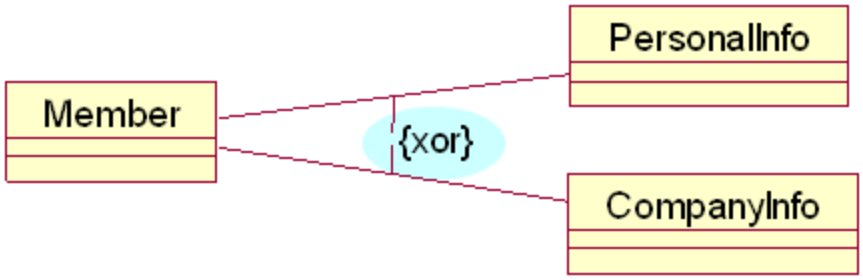
- 一致性：

  假如有Customer（客户）、Contract（合同）和Invoice（发票）三个类，要表示“一个客户拥有零个或多个合同，发票是基于某个合同的，而一个客户将收到零张或多张发票”

  Invoice：**self.contract.customer=self.customer**

对象约束语言

- 异或关系：




```
context Member inv:
    PersonalInfo->isEmpty xor
    CompanyInfo->isEmpty
```

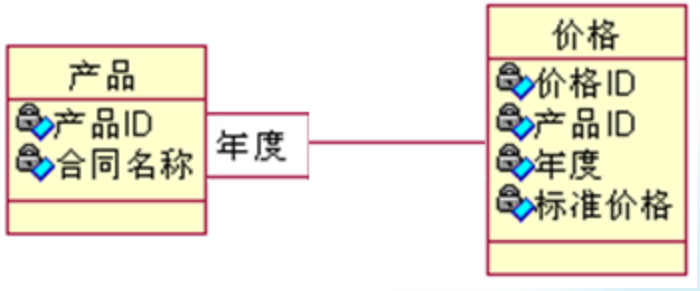
- 规定的取值范围：

  Rectangle：length>0 and width>0

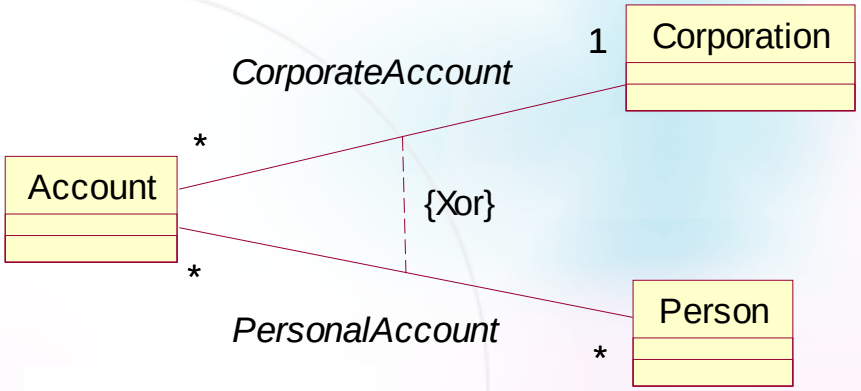
思考题

- 识别下列类之间关联的精化特征

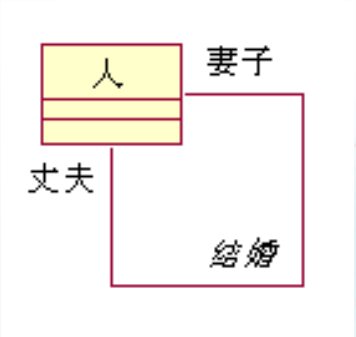
（1）



（2）



（3）



本章内容

- 类的UML表示
- 如何阅读类图
- 类的其他高级概念
- 如何绘制类图
- 类图的应用

需求描述

- 小王是一个爱书之人，家里各类书籍已过千册，而平时又时常有朋友外借，因此需要一个个人图书管理系统。该系统应该能够将书籍的基本信息按计算机类、非计算机类分别建档，实现按书名、作者、类别、出版社等关键字的组合查询功能。在使用该系统录入新书籍时系统会自动按规则生成书号，可以修改信息，但一经创建就不允许删除。该系统还应该能够对书籍的外借情况进行记录，可对外借情况列表打印。另外，还希望能够对书籍的购买金额、册数按特定时间周期进行统计 。

发现类

- 小王是一个爱书之人，家里各类书籍已过千册，而平时又时常有朋友外借，因此需要一个个人图书管理系统。该系统应该能够将书籍的基本信息按计算机类、非计算机类分别建档，实现按书名、作者、类别、出版社等关键字的组合查询功能。在使用该系统录入新书籍时系统会自动按规则生成书号，可以修改信息，但一经创建就不允许删除。该系统还应该能够对书籍的外借情况进行记录，可对外借情况列表打印。另外，还希望能够对书籍的购买金额、册数按特定时间周期进行统计

筛选备选类

- “小王”、“人”、“家里”很明显是系统外的概念，无须对其建模；
- 而“个人图书管理系统”、“系统”指的就是将要开发的系统，即系统本身，也无须对其进行建模；
- 很明显“书籍”是一个很重要的类，而“书名”、“作者”、“类别”、“出版社”、“书号”则都是用来描述书籍的基本信息的，因此应该作为“书籍”类的属性处理，而“规则”是指书号的生成规则，而书号则是书籍的一个属性，因此“规则”可以作为编写“书籍”类构造函数的指南。
- “基本信息”则是书名、作者、类别等描述书籍的基本信息统称，“关键字”则是代表其中之一，因此无需对其建模；
- “功能”、“新书籍”、“信息”、“记录”都是在描述需求时使用到的一些相关词语，并不是问题域的本质，因此先可以将其淘汰掉；

筛选备选类

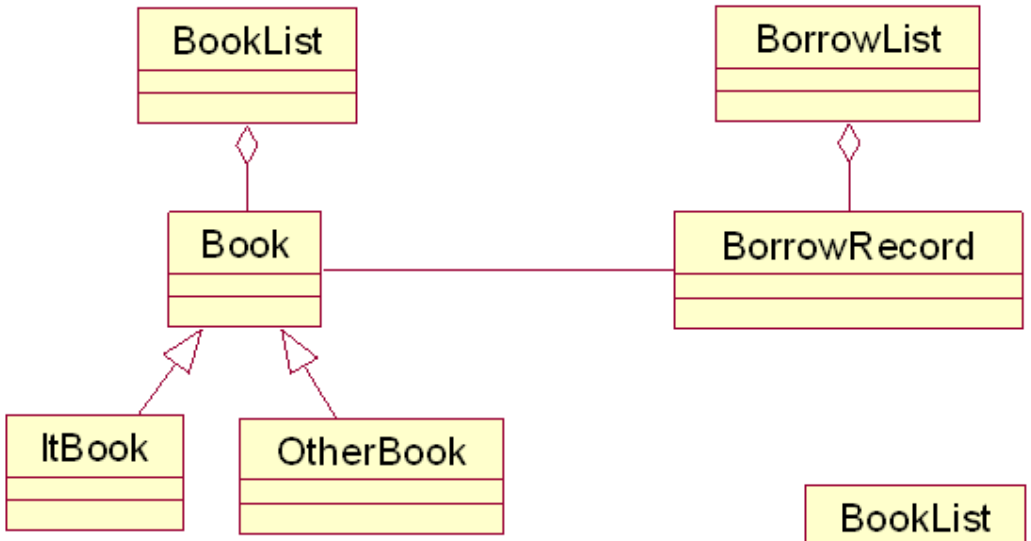
- “计算机类”、“非计算机类”是该系统中图书的两大分类，因此应该对其建模，并改名为“计算机类书籍”和“非计算机类书籍”，以减少歧义；
- “外借情况”则是用来表示一次借阅行为，应该成为一个候选类，多个外借情况将组成“外借情况列表”，而外借情况中一个很重要的角色是“朋友”—借阅主体。虽然到本系统中并不需要建立“朋友”的资料库，但考虑到可能会需要列出某个朋友的借阅情况，因此还是将其列为候选类。为了能够更好地表述，将“外借情况”改名为“借阅记录”，而将“外借情况列表”改名为“借阅记录列表”；
- “购买金额”、“册数”都是统计的结果，都是一个数字，因此不用将其建模，而“特定时限”则是统计的范围，也无需将其建模；不过从这里的分析中，我们可以发现，在该需求描述中隐藏着一个关键类—书籍列表，也就是执行统计的主体。

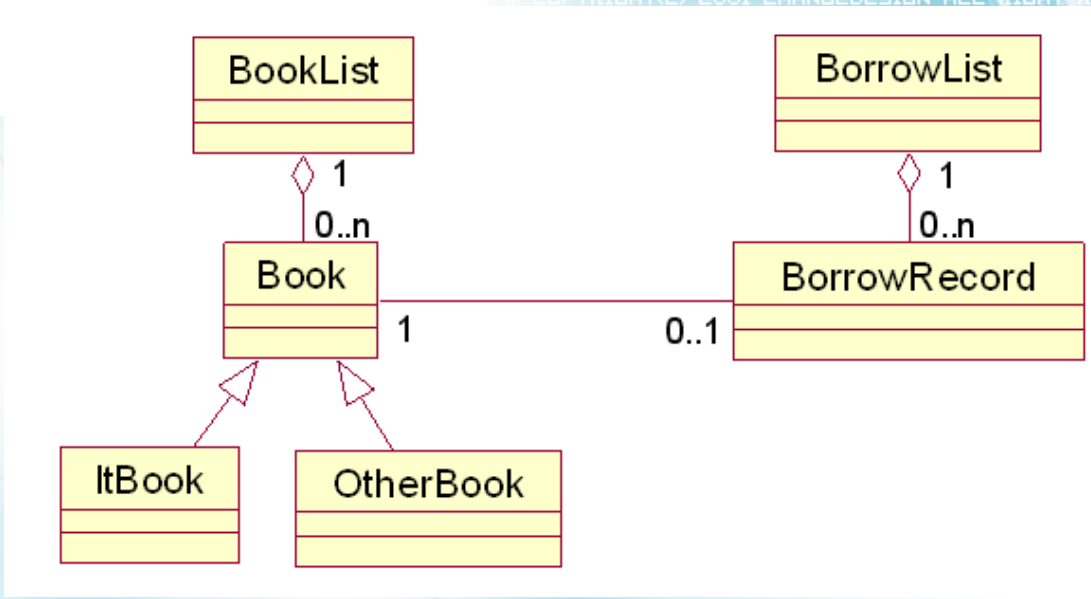
得到候选类

书籍	计算机类书籍	非计算机类书籍
借阅记录	借阅记录列表	书籍列表

- 在使用“名词动词法”寻找类的时候，很多团队会在此耗费大量的时间，特别是对于中大型项目，这样很容易迷失方向。其实在此主要的目的是对问题领域建立概要的了解，无需太过咬文嚼字

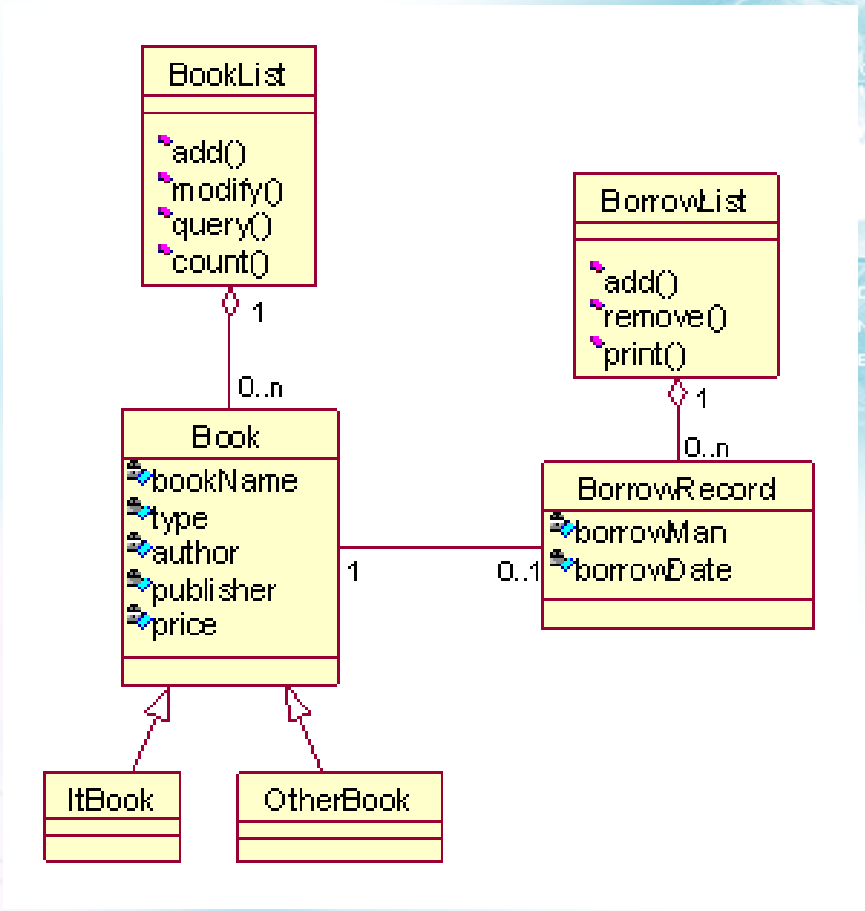
关联分析，建模，多重性分析，再建模





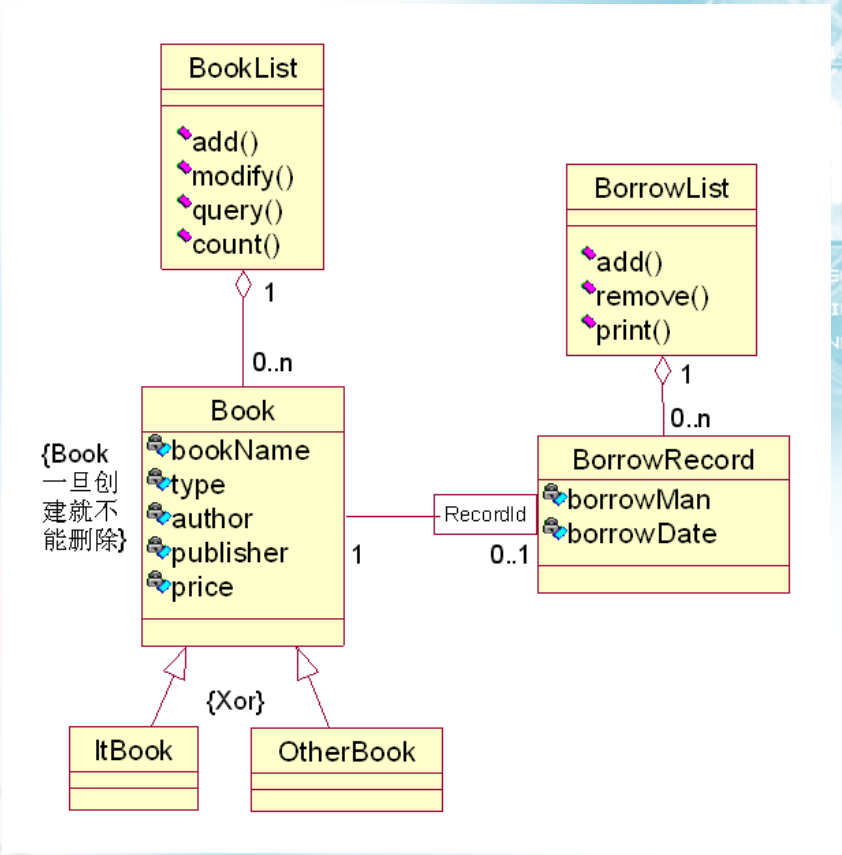
职责分析

- 书籍类：从需求描述中，可找到书名、类别、作者、出版社；同时从统计的需要中，可得知“定价”也是一个关键的成员变量。
- 书籍列表类：书籍列表就是全部的藏书列表，其主要的成员方法是新增、修改、查询（按关键字查询）、统计（按特定时限统计册数与金额）。
- 借阅记录类：借阅人（朋友）、借阅时间。
- 借阅记录列表类：主要职责就是添加记录（借出）、删除记录（归还）以及打印借阅记录



限定与修改

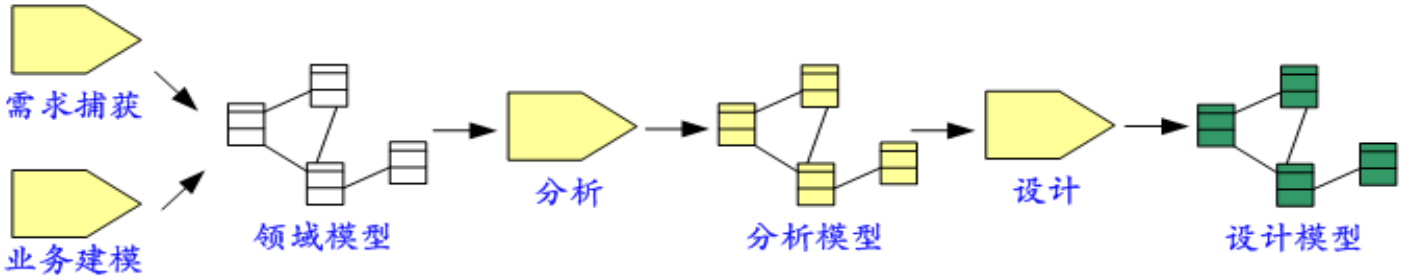
- 导航性分析：Book与BookList之间、BorrowRecord和BorrowList之间是组合关系均无需添加方向描述，而Book与BorrowRecord之间则是双方关联，也无需添加
- 约束：Book对象创建后就不能够被删除只能被修改，因此在Book类边上加上用自由文本写的约束；一本书要么属于计算机类，要么属于非计算机类，因此在ItBook和OtherBook间加了“{Xor}”约束
- 限定符：一本书只有一册，因此只能够被借一次，因此对于一本Book而言只能有一个RecordId与其对应



本章内容

- 类的UML表示
- 如何阅读类图
- 类的其他高级概念
- 如何绘制类图
- 类图的应用

软件系统模型



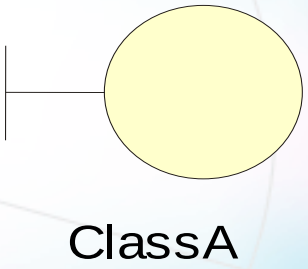
- 领域模型是从面向对象的视角看待现实世界的结果，也就是通过类图来描述现实世界中各种事物的关系。
- 分析模型和领域模型是很相近的，分析模型主要是针对软件系统的分析，领域模型则更多是偏重对业务领域的分析
- 设计模型则是在分析模型的基础上添加设计元素的结果。与分析模型相比，设计模型中的类的属性集更趋完善；

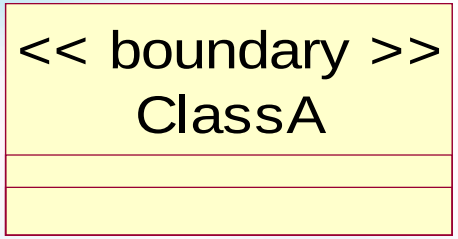
三种分析类的表示法

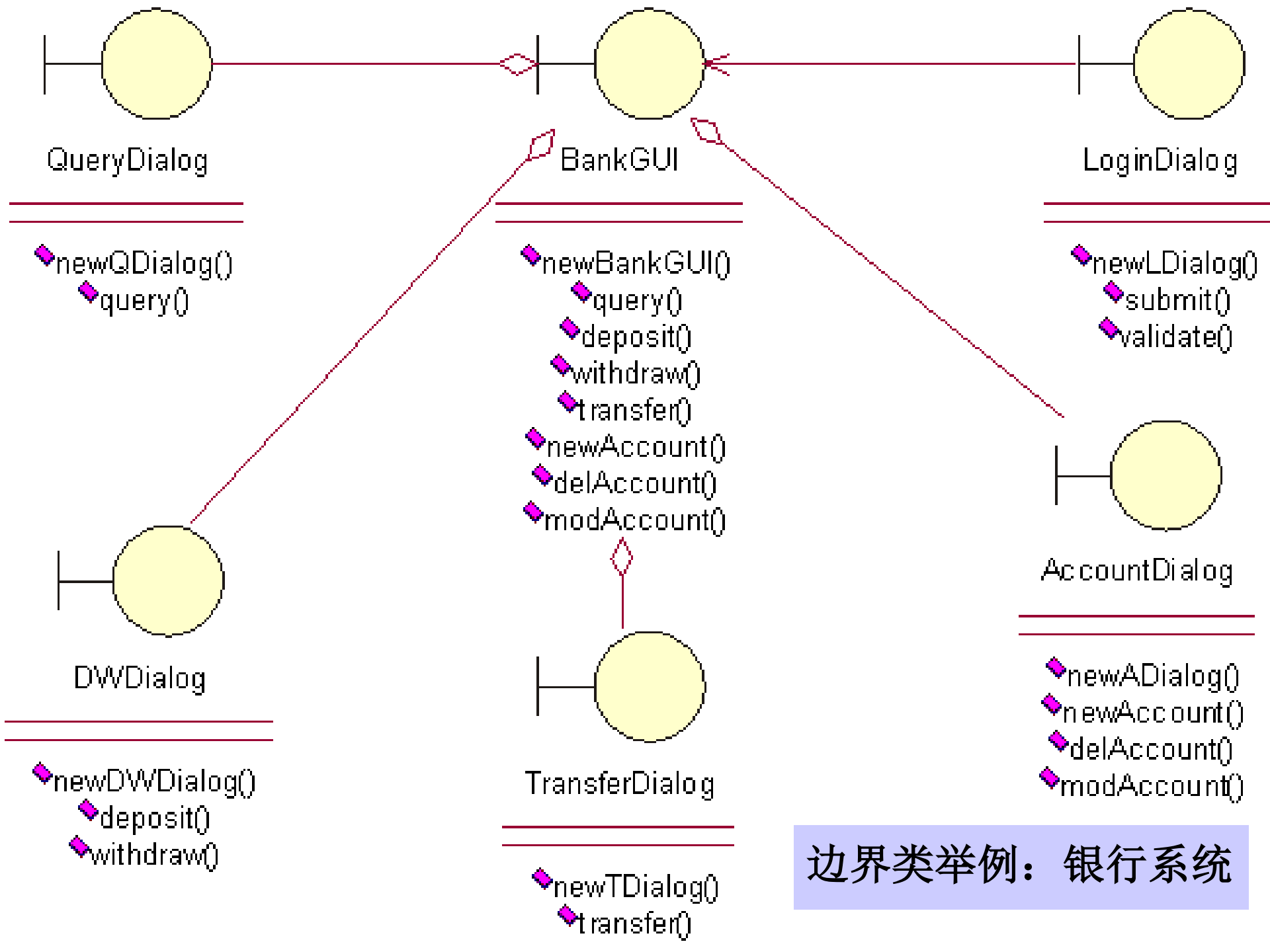
- 分析模型有3种十分有用的构造型：
  - 边界类 **boundary**
  - 控制类 **control**
  - 实体类 **entity**

三种分析类——边界类

- 边界类boundary
  - 完成参与者与系统的交互
  - 位于系统与外界的交界处，例如窗体、对话框、报表，与外部设备或系统交互的类
  - 边界类可以通过用例确定，因为参与者必须通过边界类参与用例



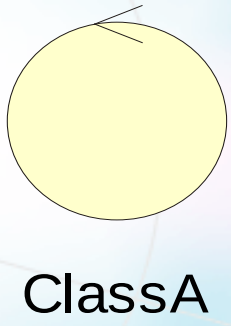


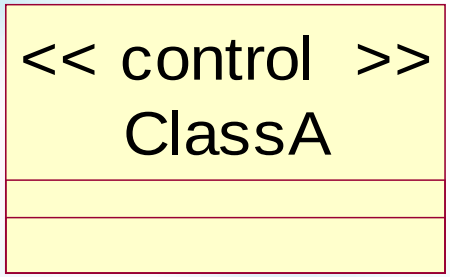

QueryDialog
newQDialog()
query()
BankGUI
newBankGUI()
query()
deposit()
withdraw()
transfer()
newAccount()
delAccount()
modAccount()
LoginDialog
newLDialog()
submit()
validate()
DWDialog
newDWDialog()
deposit()
withdraw()
TransferDialog
newTDialog()
transfer()
AccountDialog
newADialog()
newAccount()
delAccount()
modAccount()

边界类举例：银行系统

三种分析类——控制类

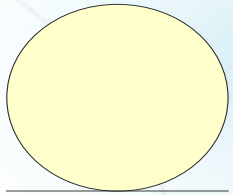
- 控制类control
  - 体现应用程序的执行逻辑，协调其他类工作和控制总体流程
  - 一般每个用例有一个控制类
  - 控制类会向其他类发送消息



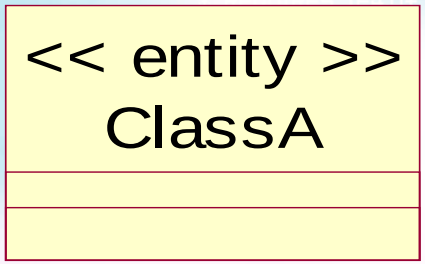


三种分析类——实体类

- 实体类entity
  - 保存永久信息，最终可能映射到数据库中的表格和文件中



ClassA



本章内容回顾

- UML中类的一些高级概念及其在UML中的表示方法
- UML的关系与Java程序实现
- 阅读类图的方法、技巧和相关的知识
- 类图的绘制方法
- 类图的作用