
第5章：UML世界的构成

* CHANGEDESIGNSTUDIO V1.0

* COPYRIGHT(C) 2001 CHANGEDESIGN ALL RIGHT RESERVED

* REQUIRES IE4.0+ -- 800*600+ -- MICROMEDIA FLASH/5 PLUGIN

* SPECIAL THANKS TO MY SITE JURGES FOR SOPHOTO AND TONYSTONE

本章内容

- UML的组成
- UML的概念模型
- UML建模实例（一）
- UML的双向工程
- UML建模实例（二）



* COPYRIGHT(C) 2001 CHANGEDESIGN ALL RIGHT RESERVED
* RESOURCES 164.0+ -- 800+600+ -- MICROMEDIA FLASH/5 PLUGIN
* VISIT OUR WEBSITE JARGES FOR SOPHOTO AND TONYSTONE



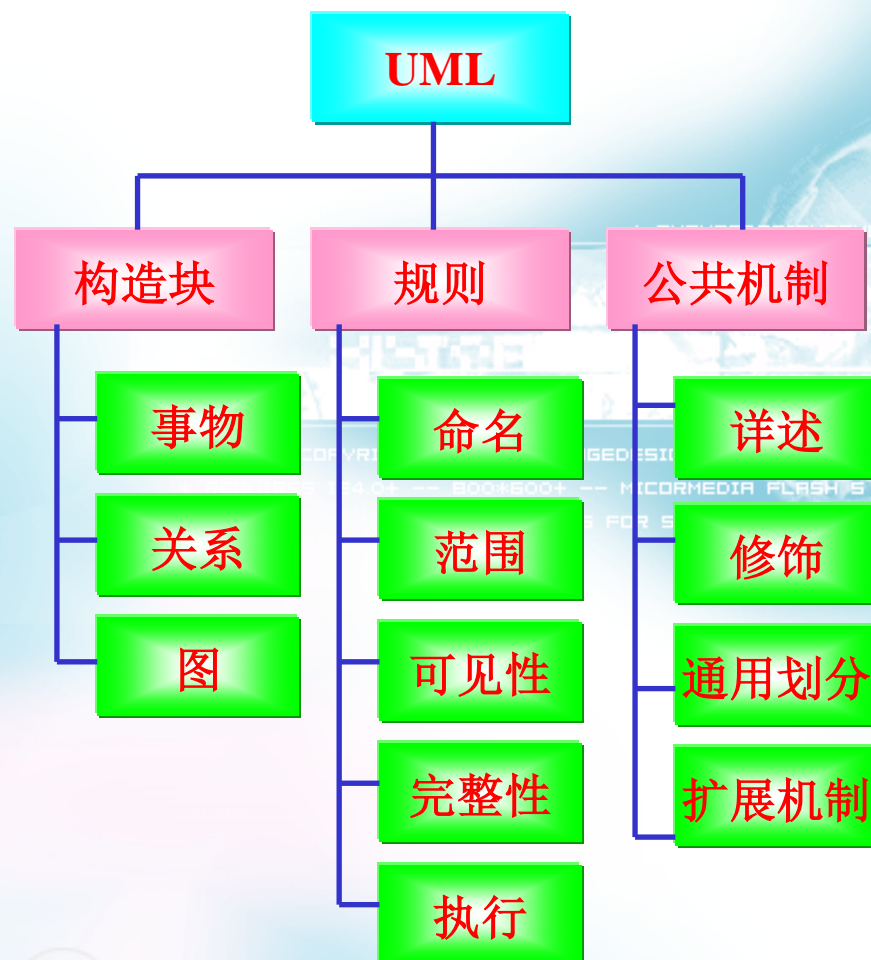
本章内容

- UML的组成
- UML的概念模型
- UML建模实例（一）
- UML的双向工程
- UML建模实例（二）



UML2.0的组成

- 基本构造块：建模元素
- UML规则：支配基本构造块如何放在一起的规则
- 公共机制：运用于整个UML模型中的公共机制、扩展机制



UML2.4组成

- **UML基础结构（Infrastructure）**——定义一个可复用的元语言核心，用来定义各种元模型，包括UML、MOF（UML 1）和CWm等元模型。
- **UML上层结构（Superstructure）**——提供可直接用来构造用户系统的各种模型元素，以及从不同的视角对系统进行建模的各种模型图。
- **UML 对象约束语言（OCL）**——即是UML的一部分，又可作独立语言。
- **UML图交换（Diagram Interchange）**——给出在不同的建模工具之间实现模型交换的规范。

本章内容

- UML的组成
- UML的概念模型
- UML建模实例（一）
- UML的双向工程
- UML建模实例（二）



UML的概念模型

- UML构造块：事物、关系、图
- UML规则
- UML公共机制
- UML体系结构与图



* COPYRIGHT(C) 2001 CHANGEDESIGN ALL RIGHT RESERVED
* REQUIRES IE4.0+ -- 800*600+ -- MICROMEDIA FLASH/5 PLUGIN
* WEBSITE PURCHASES FOR SOPHOTO AND TONYSTONE



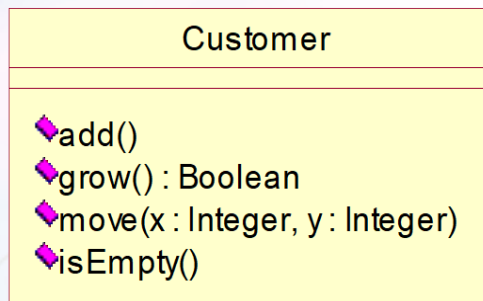
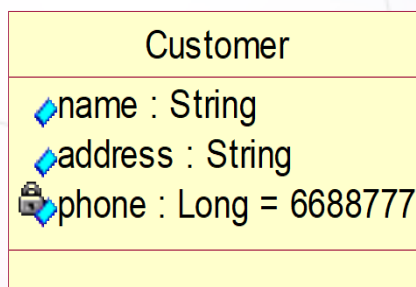
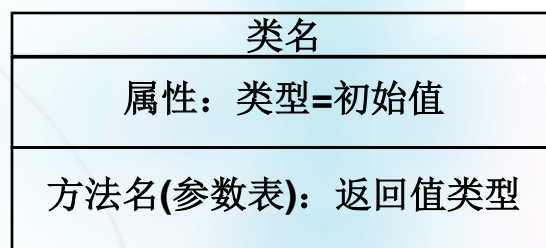
UML构造块——事物thing

事物构造块是对模型中最具有代表性的成分的抽象

- **结构事物**：UML中的名词，它是模型的静态部分，描述概念或物理元素。
- **行为事物**：UML中的动词，它是模型中的动态部分。
- **分组事物**：UML中的容器，用来组织模型，使模型更加的结构化。
- **注释事物**：UML中的解释部分，和代码中的注释语句一样，是用来描述模型的。

结构事物——类

- 类是对一组具有相同属性、相同操作、相同关系和相同语义的对象的抽象
- UML中类是用一个矩形表示的，它包含三个区域，最上面是类名、中间是类的属性、最下面是类的方法



结构事物——对象

- 对象是类的一个实例

Point
x: int y: int
distance():int

类

<u>p1:Point</u>
x=1 y=5

Point对象
被命名为p1

<u>:Point</u>
x=2 y=3

匿名
Point对象

* COPYRIGHT(C) 2001 CHANGEDESIGN ALL RIGHT RESERVED
104.0+ -- 800+600+ -- MICROMEDIA FLASH/5 PLUGIN
WWW.CHANGEDESIGN.COM SITE IMAGES FOR SOPHOTO AND TONYSTONE

结构事物——接口

- 接口是描述某个类或构件的一个服务操作集

接口

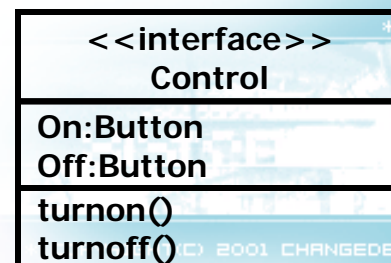
方法

提供汉堡(套餐名, 钱) 它实际完成“提供汉堡”功能



其它类

想要汉堡



Control

UML1.0规范



供给接口



需求接口

UML2.0规范

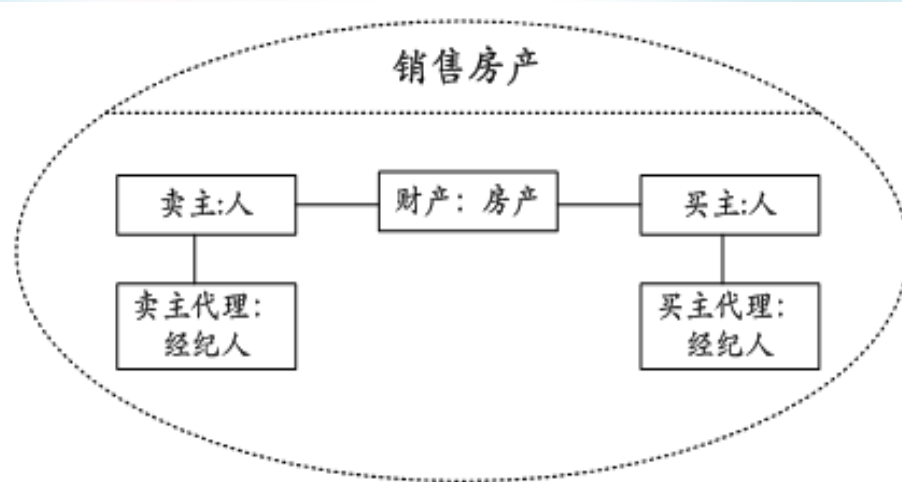
结构事物——用例与协作

- 用例是Ivar Jacobson首先提出的，现已成为了面向对象软件开发中一个需求分析的最常用工具
- 用例实例是在系统中执行的一系列动作，这些动作将生成特定执行者可见的价值结果。



维护课程表

- 协作定义了一个交互，它是由一组共同工作以提供某协作行为的角色和其他元素构成的一个群体。
- 对于某个用例的实现就可以表示为一个协作

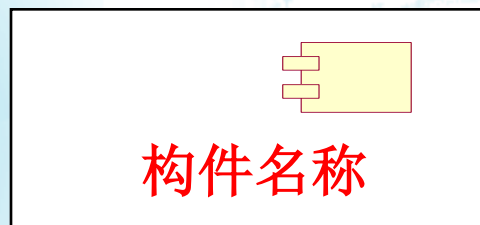


结构事物——构件

- 在实际的软件系统中，有许多要比“类”更大的实体，例如一个COM组件、一个DLL文件、一个JavaBeans、一个执行文件等等。为了更好地对在UML模型中对它们进行表示，就引入了构件（也译为组件）



1.0 习惯用法



2.0 习惯用法

- 构件是系统设计的一个模块化部分，它隐藏了内部的实现，对外提供了一组外部接口。在系统中满足相同接口的组件可以自由地替换

结构事物——节点

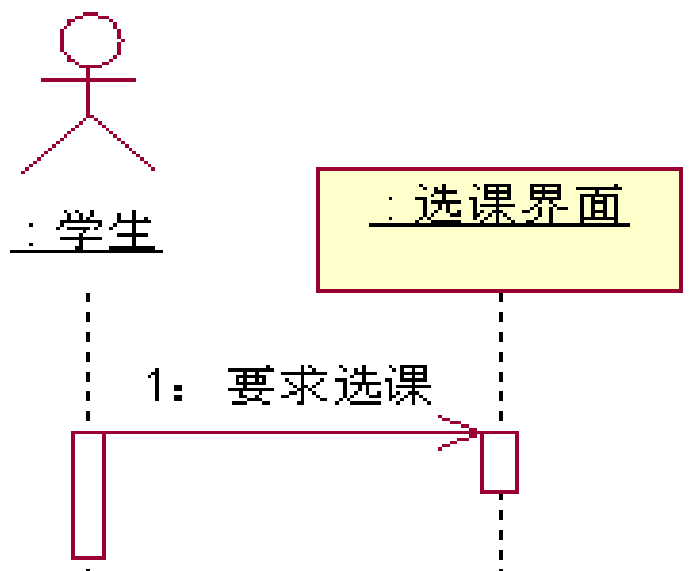
- 为了能够有效地对部署的结构进行建模，UML引入了节点这一概念，它可以用来描述实际的PC机、服务器等软件运行的基础硬件



- 节点是运行时存在的物理元素，它表示了一种可计算的资源，通常至少有存储空间和处理能力

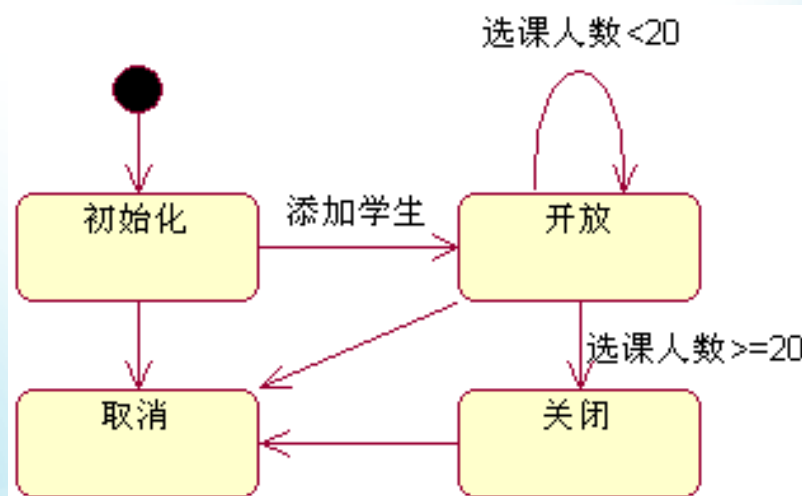
行为事物——交互

- 交互（interaction）：在特定语境中，共同完成某个任务的一组对象之间交换的信息集合。
- 表示法是一条有向直线，并在上面标有操作名

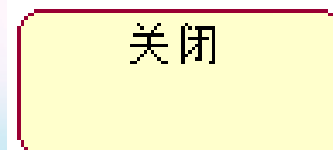


行为事物——状态和状态机

- 状态机（state machine）：
是一个对象或交互在生命
周期内响应事件所经历的
状态序列

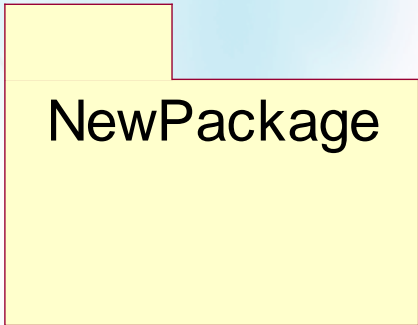


- 状态：在UML模型中将状态画为一个圆角矩形，并在矩形内写出状态名称及其子状态



分组事物——包

- 对于一个中大型的软件系统而言，通常会包含大量的类，因此也就会存在大量的结构事物、行为事物，为了能够更加有效地对其进行整合，生成或简或繁、或宏观或微观的模型，就需要对其进行分组。在UML中，提供了“包（Package）”来完成这一目标



A UML Package Diagram showing a package named 'NewPackage'. The package is represented by a yellow rectangle with a red border. The name 'NewPackage' is written in black text inside the rectangle.

NewPackage

注释事物——注释

- 结构事物是模型的主要构造块，行为事物则是补充了模型中的动态部分，分组事物而是用来更好地组织模型，似乎已经很完整了。而注释事物则是用来锦上添花的，它是用来在UML模型上添加适当的解释部分。



Note

思考题

- 在UML图中，对象和类如何区分？

- (a) 对象的标签显式为斜体
- (b) 类的标签放在一个方框中
- (c) 对象的标签加了下划线



UML构造块——关系relationship

种类	变种	表示法	关键字或符号	种类	变种	表示法	关键字或符号
抽象	派生	依赖关系	《derive》	导入	私有	依赖关系	《access》
	显现		《manifest》		公有		《import》
	实现	实现关系	虚线加空心三角	信息流			《flow》
	精化	依赖关系	《refine》	包含并			《merge》
	跟踪		《trace》	许可			《permit》
关联		关联关系	实线	协议符合			未指定
绑定		依赖关系	《bind》(参数表)	替换		依赖关系	《substitute》
部署			《deploy》	使用	调用		《call》
扩展	Extend		《extend》(扩展点)		创建		《create》
扩展	extension	扩展关系	实线加实心三角		实例化		《instantiate》
泛化		泛化关系	实线加空心三角		职责		《responsibility》
包含		依赖关系	《include》		发送		《send》

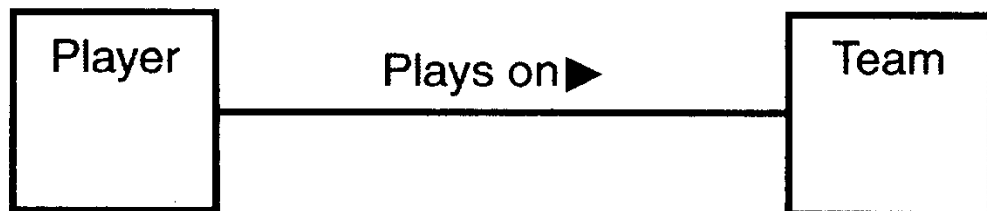
关系——关联关系

- 关联（Association）表示两个类之间存在某种语义上的联系。关联关系提供了通信的路径，它是所有关系中最通用、语义最弱的。
- 在UML中，使用一条实线来表示关联关系

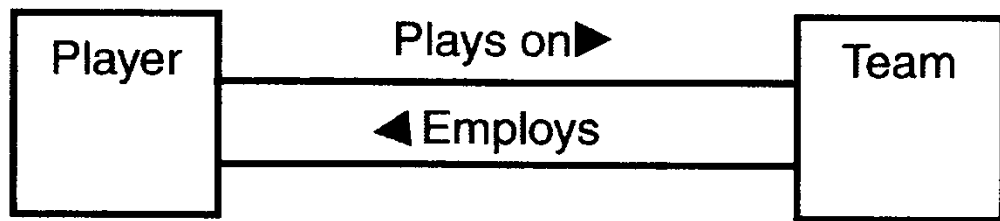


关系——关联关系

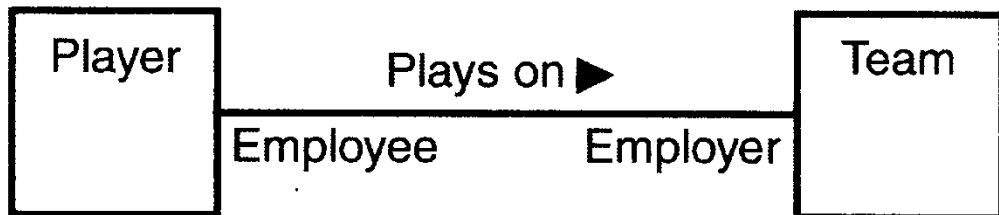
- 关联关系举例



队员和球队之间的关联



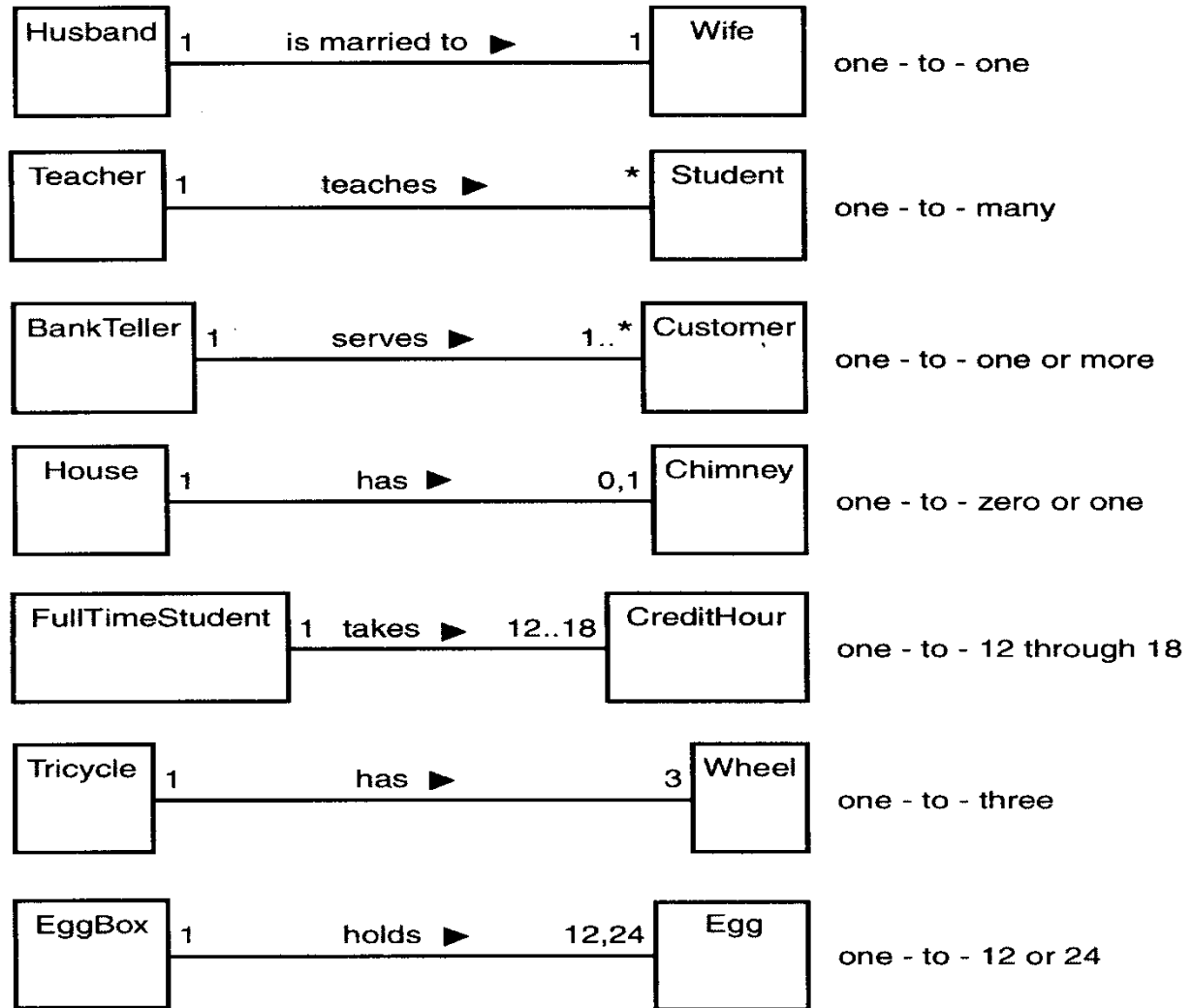
两个类之间的不同关联
可以表示在一个图中



参与关联的每个类
通常都扮演某种角色

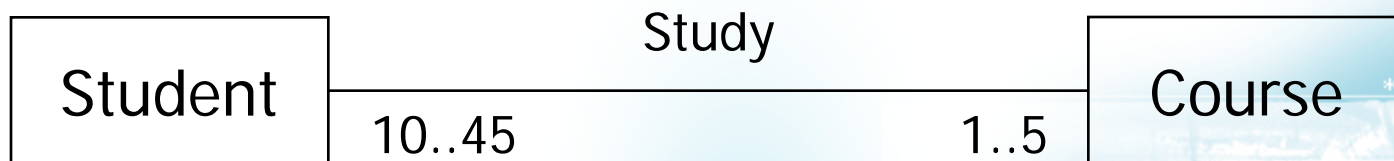
关系——关联关系

关联的多重性举例

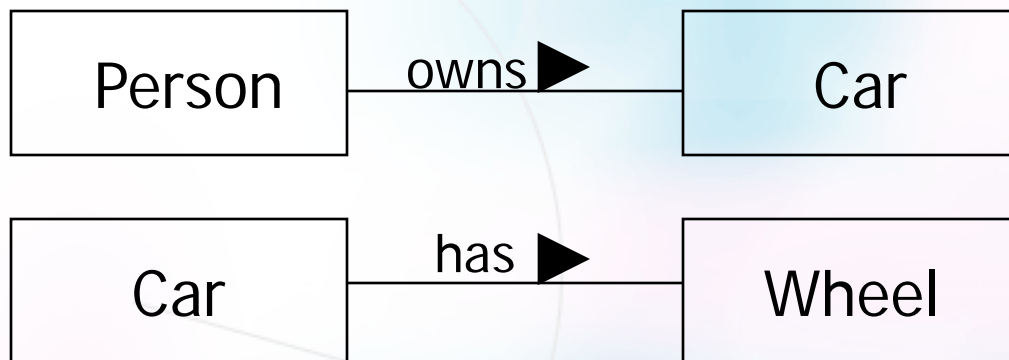


思考题

- 描述下列关联关系？



- 标注下列关联的多重性？



关系——关联关系

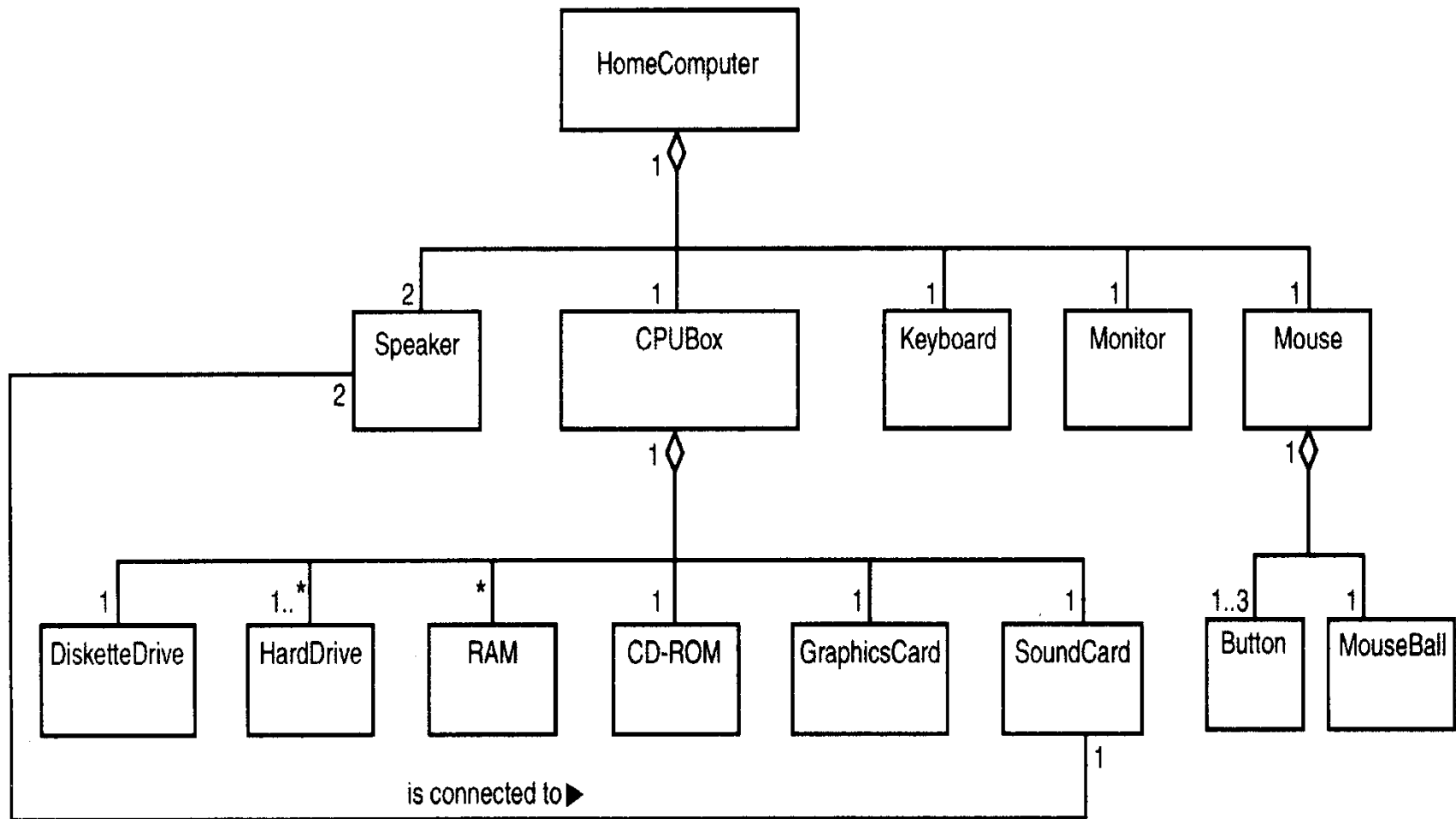
- 在关联关系中，有两种比较特殊的关系：聚合和组合



- 聚合关系：聚合（Aggregation）是一种特殊形式的关联。聚合表示类之间的关系是整体与部分的关系
- 如果发现“部分”类的存在，是完全依赖于“整体”类的，那么就应该使用“组合”关系（Composition）来描述

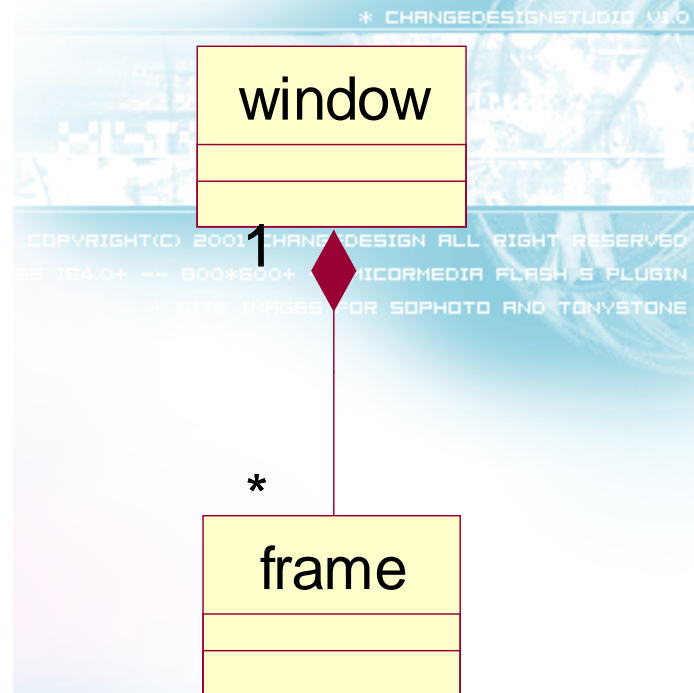
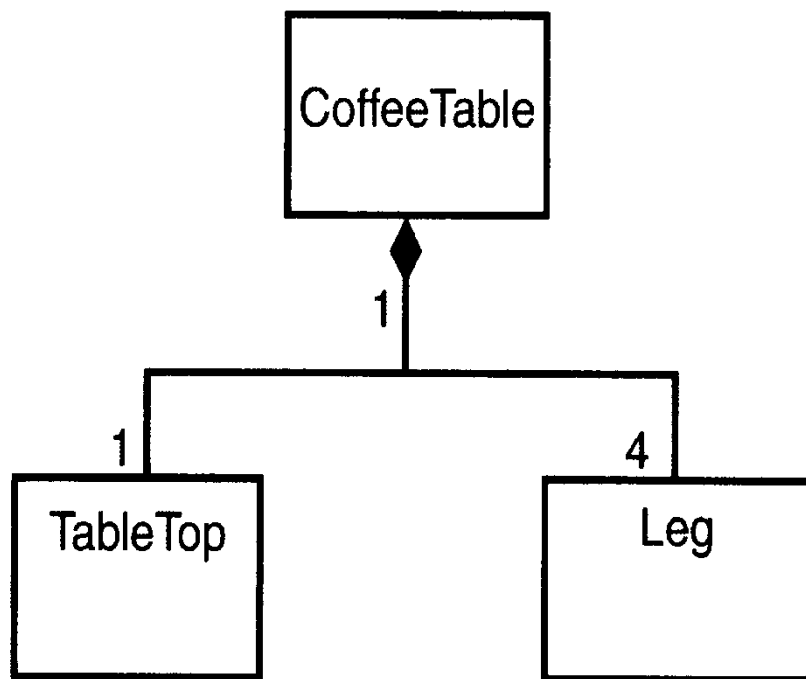
关系——关联关系

● 聚合举例



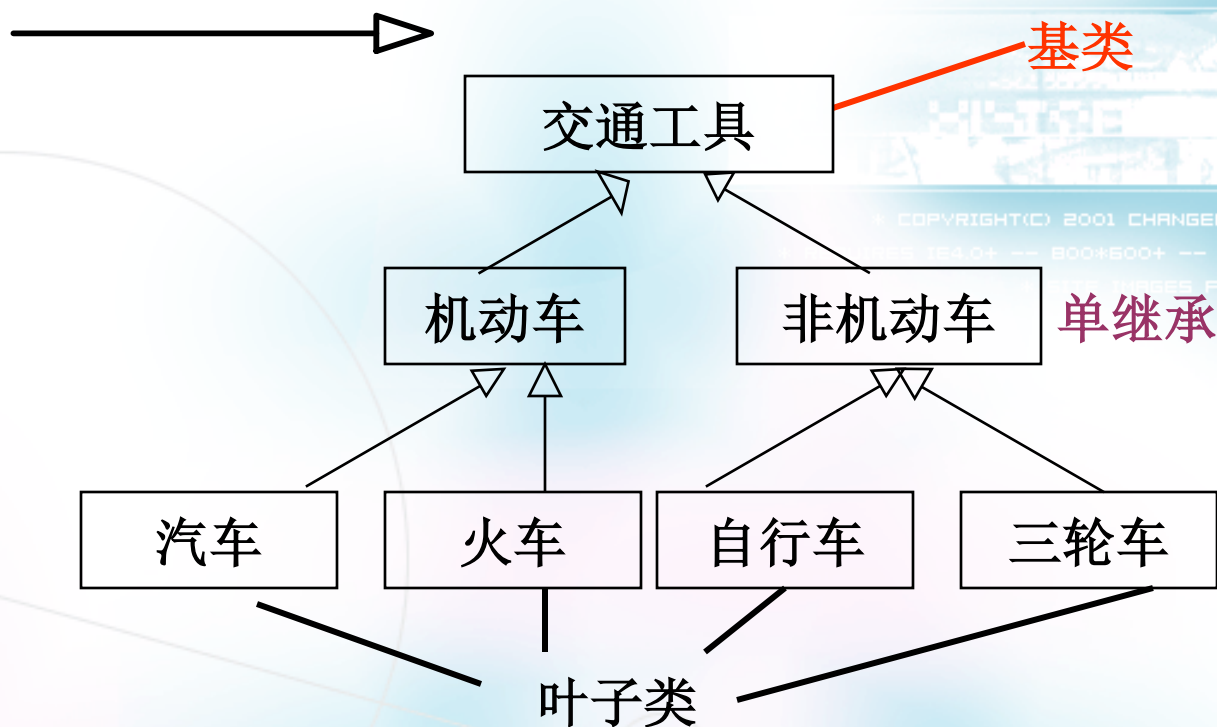
关系——关联关系

- 组合举例



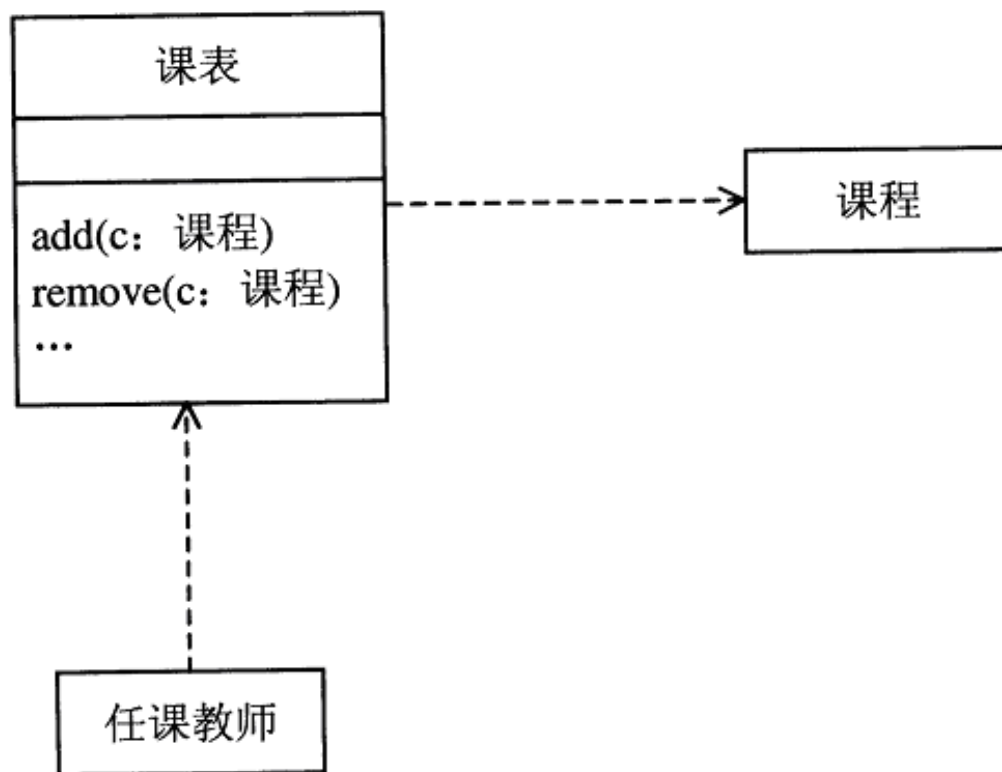
关系——泛化关系

- 泛化（Generalization）关系描述了一般事物与该事物中的特殊种类之间的关系，也就是父类与子类之间的关系。



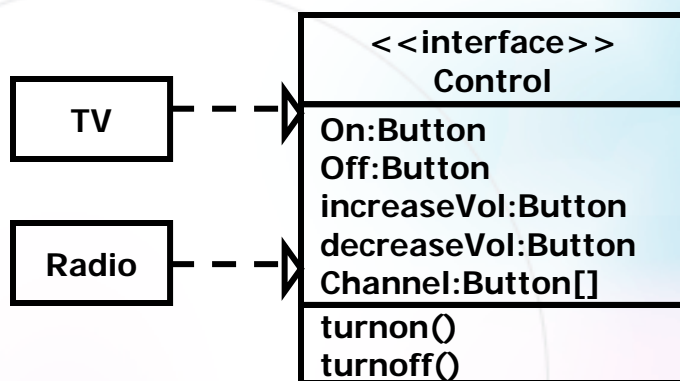
关系——依赖关系

- 有两个元素X、Y，如果修改元素X的定义可能会引起对另一个元素Y的定义的修改，则称元素Y依赖（Dependency）于元素X。



关系——实现关系

- 实现（**Realization**）关系是用来规定接口和实现接口的类或组件之间的关系。接口是操作的集合，这些操作用于规定类或组件的服务。



思考题

- 分析下面的关系，哪些是普通的关联，哪些是聚合？

(1) 街道上的房子

(2) 书中的书页

(3) 交响乐中的音符



思考题

- 以UML图形表示下列关系(标注多重性), 并指出下列关系是哪种关系?

- (1) 一个国家有一个首都
- (2) 一位进餐的哲学家正在使用一把叉子
- (3) 一条线由一组有序的点组成
- (4) 一个运动员在一个时期内只能效力于一个运动队



UML的概念模型







- UML构造块
- UML规则
- UML公共机制
- UML体系结构与图



UML规则

- 命名：为事物、关系和图起名字。由字符、数字、下划线组成，名字必须是唯一的。

- 可见性：

可见性	规则	标准表示法	Rose 属性	Rose 方法
public	任一元素，若能访问包容器，就可以访问它	+		
protected	只有包容器中的元素或包容器的后代才能够看到它	#		
private	只有包容器中的元素才能够看得到它	-		
package	只有声明在同一个包中的元素才能够看到该元素	~		

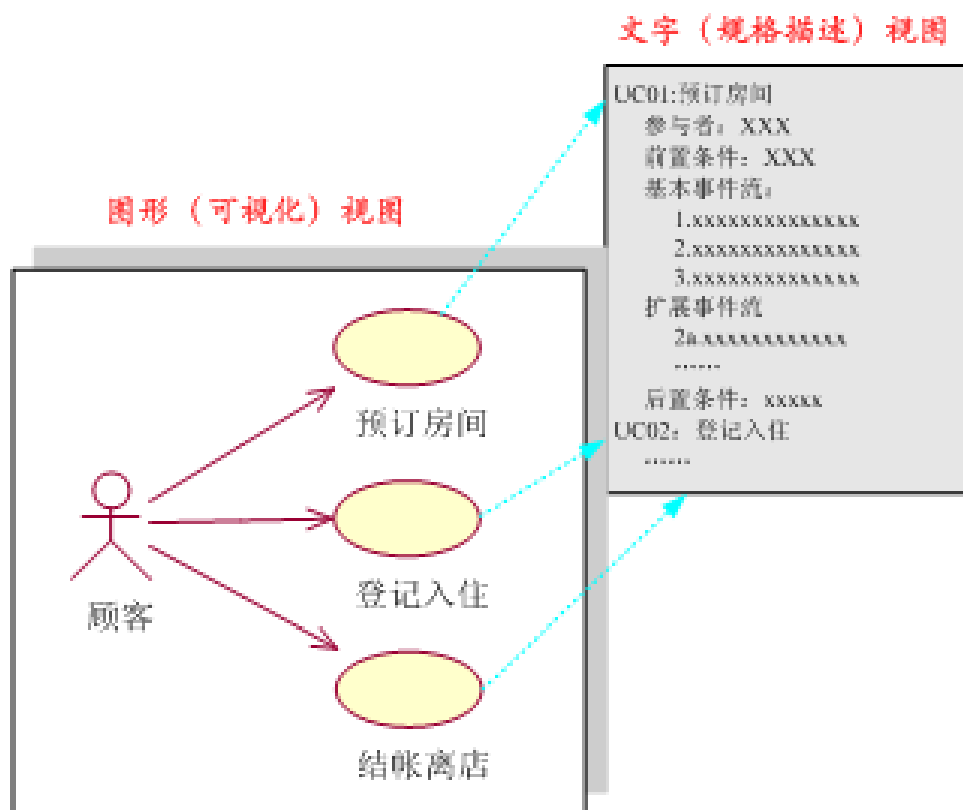
UML的概念模型

- UML构造块
- UML规则
- UML公共机制
- UML体系结构与图



UML公共机制——规格描述

- 在图形表示法的每个部分后面都有一个规格描述（也称为详述），它用来对构造块的语法和语义进行文字叙述。这种构思，也就使可视化视图和文字视图的分离：

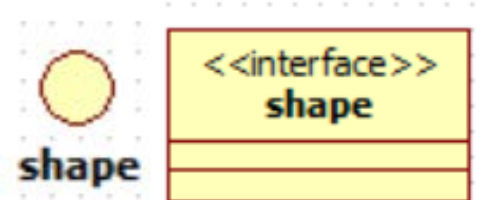
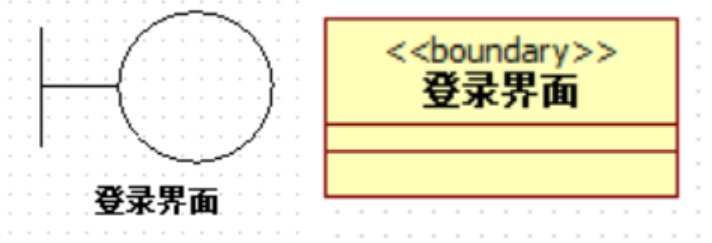


UML公共机制——UML修饰与通用划分

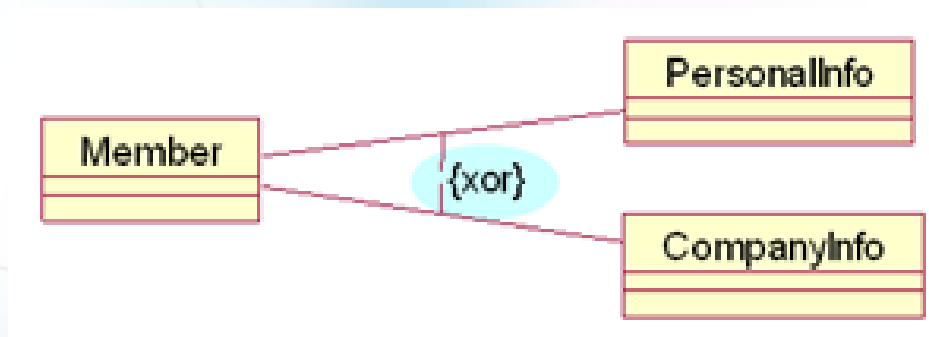
- 在为了更好的表示这些细节，UML中还提供了一些修饰符号，例如不同可视性的符号、用斜体字表示抽象类
- UML通用划分：
 - 1) 类与对象的划分：类是一种抽象，对象是一个具体的实例
 - 2) 接口与实现的分离：接口是一种声明、是一个契约，也是服务的入口；实现则是负责实施接口提供的契约

UML扩展机制

- **构造型(stereotype)**: 在实际的建模过程中, 可能会需要定义一些特定于某个领域或某个系统的构造块



- **约束**是用来增加新的语义或改变已存在规则的一种机制（自由文本和OCL两种表示法）。使用花括号括起来的串来表示, 放在相关的元素附近



UML的概念模型

- UML构造块
- UML规则
- UML公共机制
- UML体系结构与图



* COPYRIGHT(C) 2001 CHANGEDESIGN ALL RIGHT RESERVED
* REQUIRES IE4.0+ -- 800*600+ -- MICROMEDIA FLASH/5 PLUGIN
* WEBSITE: WWW.CHANGEDESIGN.COM * WEBSITE: WWW.SOPHOTO.COM AND WWW.TONYSTONE.COM



UML定义的图

图名	功能	备注
类图	描述类、类的特性以及类之间的关系	UML 1原有
对象图	描述一个时间点上系统中各个对象的一个快照	UML 1非正式图
复合结构图	描述类的运行时刻的分解	UML 2.0新增
构件图	描述构件的结构与连接	UML 1原有
部署图	描述在各个节点上的部署	UML 1原有
包图	描述编译时的层次结构	UML中非正式图
用例图	描述用户与系统如何交互	UML 1原有
活动图	描述过程行为与并行行为	UML 1原有
状态机图	描述事件如何改变对象生命周期	UML 1原有
顺序图	描述对象之间的交互，重点在强调顺序	UML 1原有
通信图	描述对象之间的交互，重点在于连接	UML 1中的协作图
定时图	描述对象之间的交互，重点在于定时	UML 2.0 新增
交互概览图	是一种顺序图与活动图的混合	UML 2.0新增

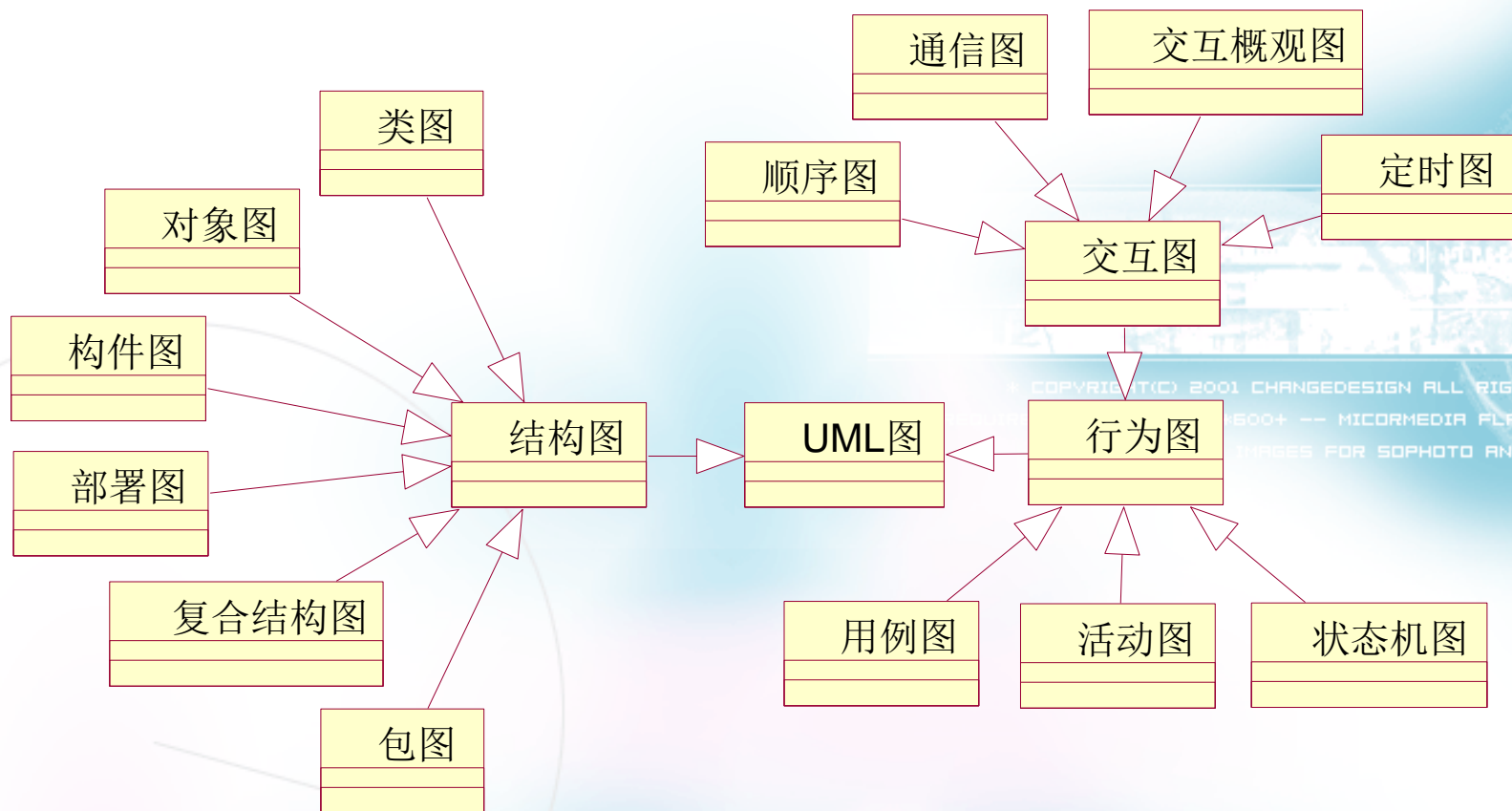
UML2.4中的几种图

- 结构图（6种）：类图、构件图、对象图、部署图（上四种UML 1都有）、组合结构图、包图。
- 行为图（3种）：活动图、用例图、状态机图
- 交互图（4种）：顺序图、通信图[协作图]（以上2种UML 1都有）、交互概览图、定时图

UML视图和图

主要领域	视图	图
结构	静态视图	类图
	设计视图	复合结构图、协作图、构件图
	用例视图	用例图
动态	状态视图	状态机图
	活动视图	活动图
	交互视图	顺序图、通信图
物理	部署视图	部署图
模型管理	模型管理视图	包图
	特性描述	包图

UML图形分类



本章内容

- UML的组成
- UML的概念模型
- UML建模实例（一）
- UML的双向工程
- UML建模实例（二）



UML建模实例——选课系统

实例——XX大学想用软件实现选课过程

- 教务处课程管理人员昭示本学期可以选择的课程
 - 每门课可能又多个不同的班在教授
- 学生选择4门必修课和2门选修课
- 当学生选课完毕，学校的独立计费系统将会记录学生应缴纳的学费金额
- 在选定课程后的一段时间内，学生可以删除或添加别的课程
- 老师可以通过该系统得到选修自己课程的学生名单
- 学生、老师登陆时需要进行验证

UML建模实例——选课系统

● 识别参与者（ Actor ）

- 课程管理人员
- 学生
- 老师
- 独立的学校计费系统

● 识别用例（Use case）

- 课程管理人员：建立和发布本学期课程表
- 学生：维护自己的课程表
- 老师：取得人员名单
- 独立的学校计费系统：得到计费信息

UML建模实例——选课系统

- 使用活动图 (Activity diagram) 分析特定use case
- 根据用例和文档抽象出类(Class)
- 根据类和用例的特定场景制作序列图 (Sequence diagram)
- 根据序列图完善类图的方法
- 用状态图 (Statechart diagram) 说明关键类的状态
- 用构件图 (Component diagram) 说明程序的逻辑组织
- 用部署图 (Deployment diagram) 说明系统的部署

本章内容

- UML的组成
- UML的概念模型
- UML建模实例（一）
- UML的双向工程
- UML建模实例（二）



* COPYRIGHT(C) 2001 CHANGEDESIGN ALL RIGHT RESERVED
* REQUIRES IE4.0+ -- 800*600+ -- MICROMEDIA FLASH/5 PLUGIN
* VISIT OUR WEBSITE: WWW.CHANGEDESIGN.COM SITE JURGES FOR SOPHOTO AND TONYSTONE



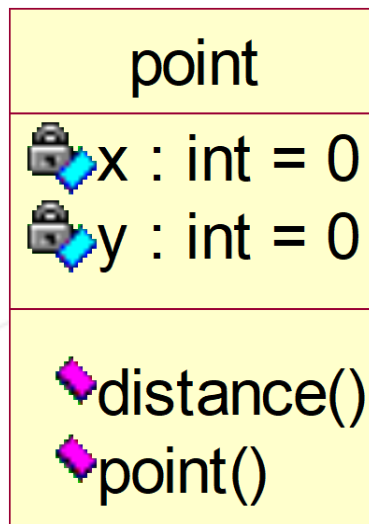
UML的正向工程

● 由模型生成代码

举例：Rose生成Java代码步骤

- (1) 建立模型
- (2) 选择[Tools]->[Check Model]检查模型的规则是否有错
- (3) 选择[Tools]->[Java]->[Syntax Check]检查Java语法
- (4) 选择[Tools]->[Java]->[Generate Java]生成代码
- (5) 选择[Tools]->[Java]->[Edit Code]编辑生成的代码

UML的正向工程



```
point
//Source file: f:\point.java
public class point
{
    private int x = 0;
    private int y = 0;
    /**
     * @roseuid 46088E4D03D8
     */
    public point()
    {
    }
    /**
     * @return int
     * @roseuid 46088DB80196
     */
    public int distance()
    {
        return 0;
    }
}
```

UML的逆向工程

● 由代码生成模型

举例：由Java代码生成Rose模型步骤

- (1) 装入相应Java应用框架
- (2) 选择[Tools]->[Java]->[Reverse Engineer Java]
- (3) 选择逆向工程的Java文件，然后单击[Reverse]

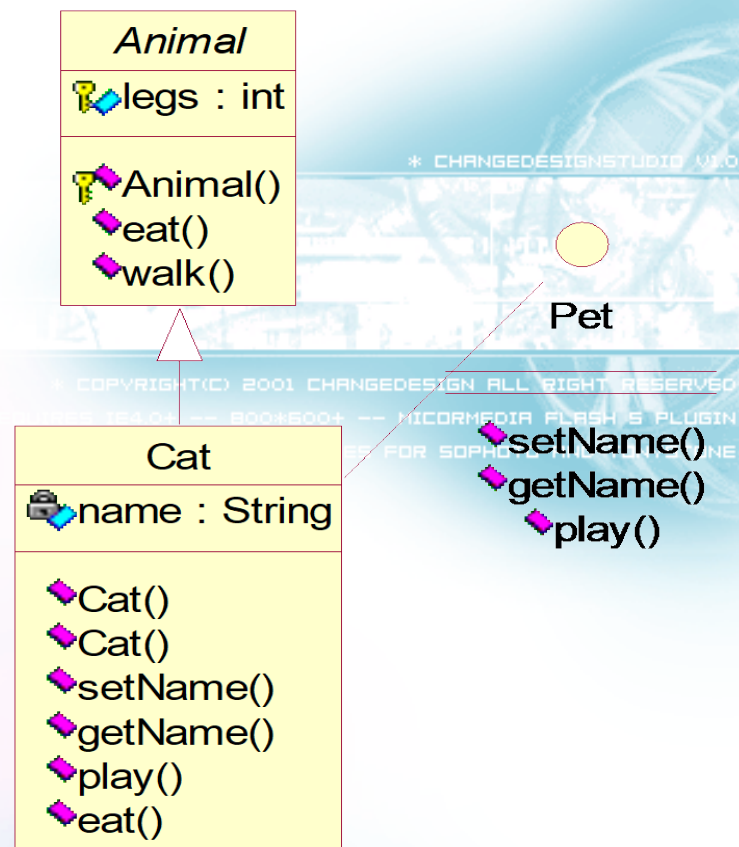
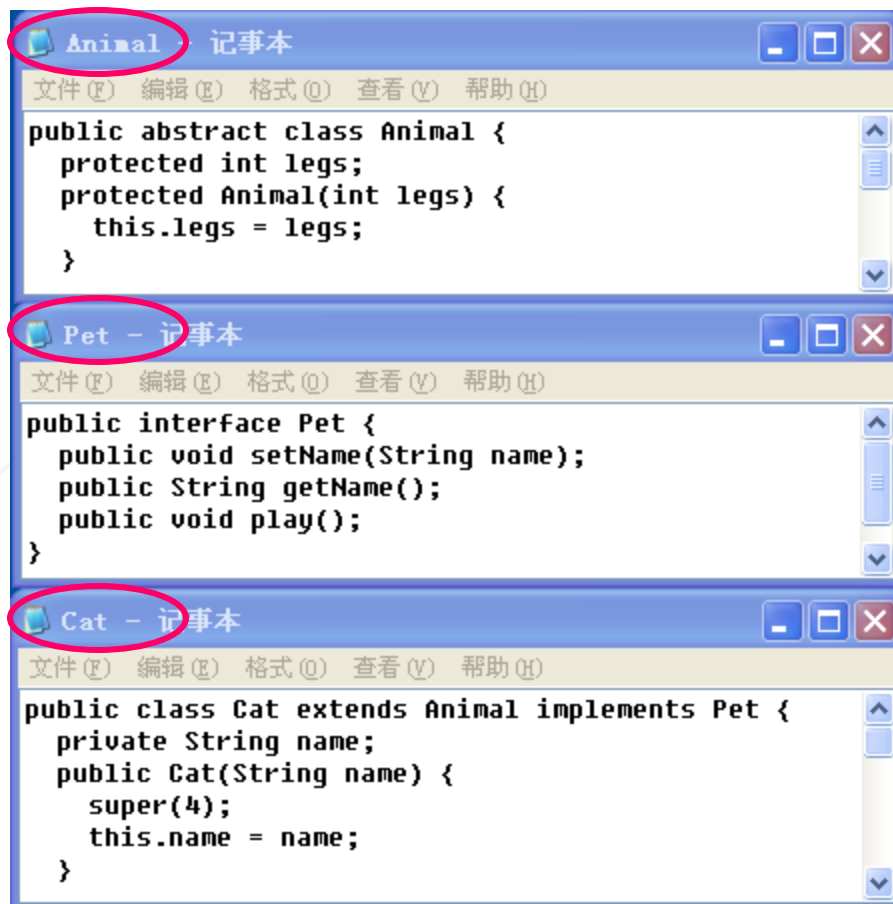
* CHANGEDESIGNSTUDIO V1.0

© COPYRIGHT(C) 2001 CHANGEDESIGN ALL RIGHT RESERVED

FOR WINDOWS 95/98/0+ -- 800*600+ -- MICROMEDIA FLASH/5 PLUGIN

FOR MORE INFORMATION VISIT OUR WEBSITE JURGES FOR SOPHOTO AND TONYSTONE

UML的逆向工程



本章内容

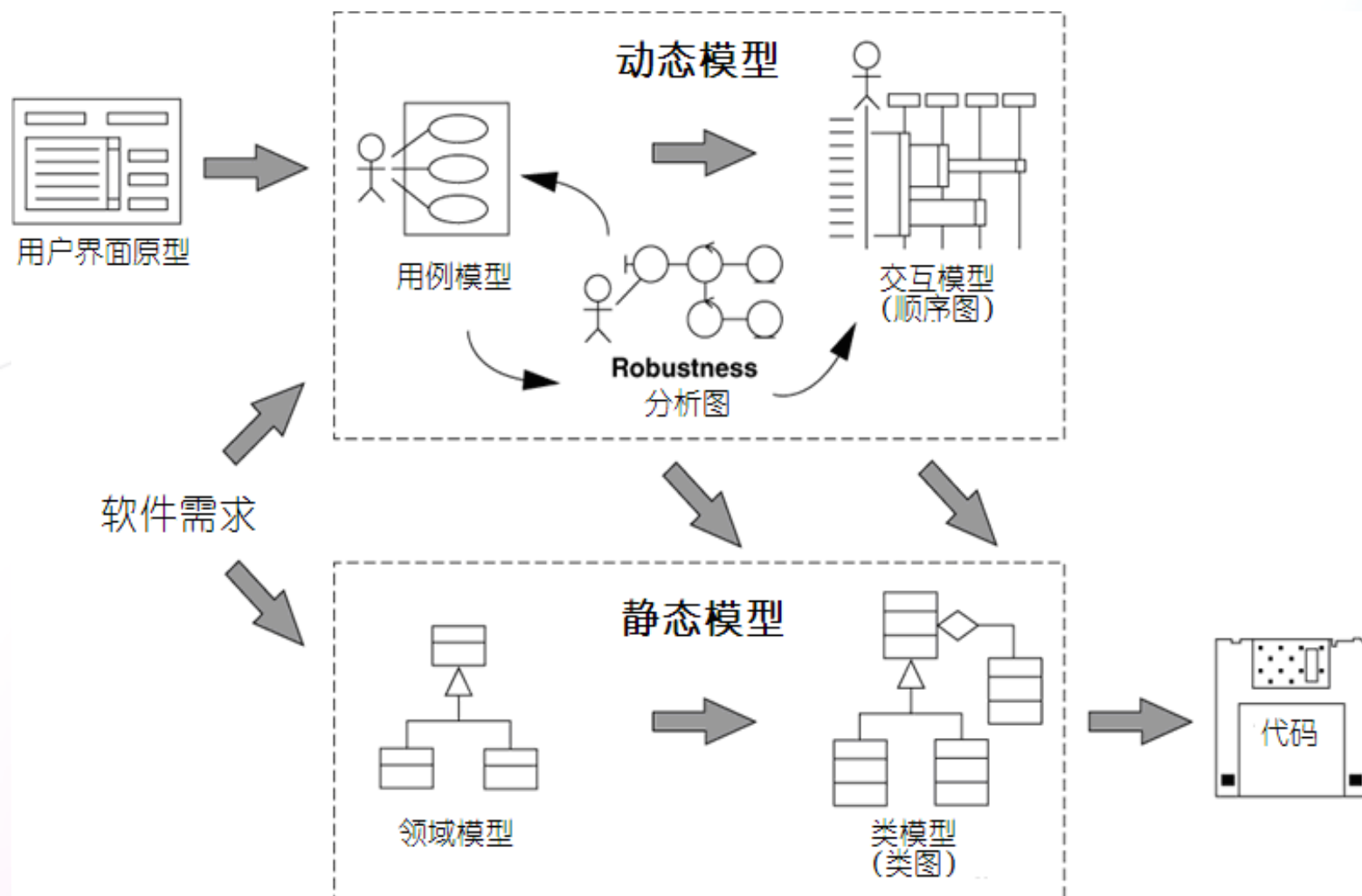
- UML的组成
- UML的概念模型
- UML建模实例（一）
- UML的双向工程
- UML建模实例（二）



* COPYRIGHT(C) 2001 CHANGEDESIGN ALL RIGHT RESERVED
* REQUIRES IE4.0+ -- 800*600+ -- MICROMEDIA FLASH/5 PLUGIN
* VISIT OUR WEBSITE: WWW.CHANGEDESIGN.COM FOR 50PHOTO AND TONYSTONE



基于UML的需求分析过程



UML建模实例：小游戏

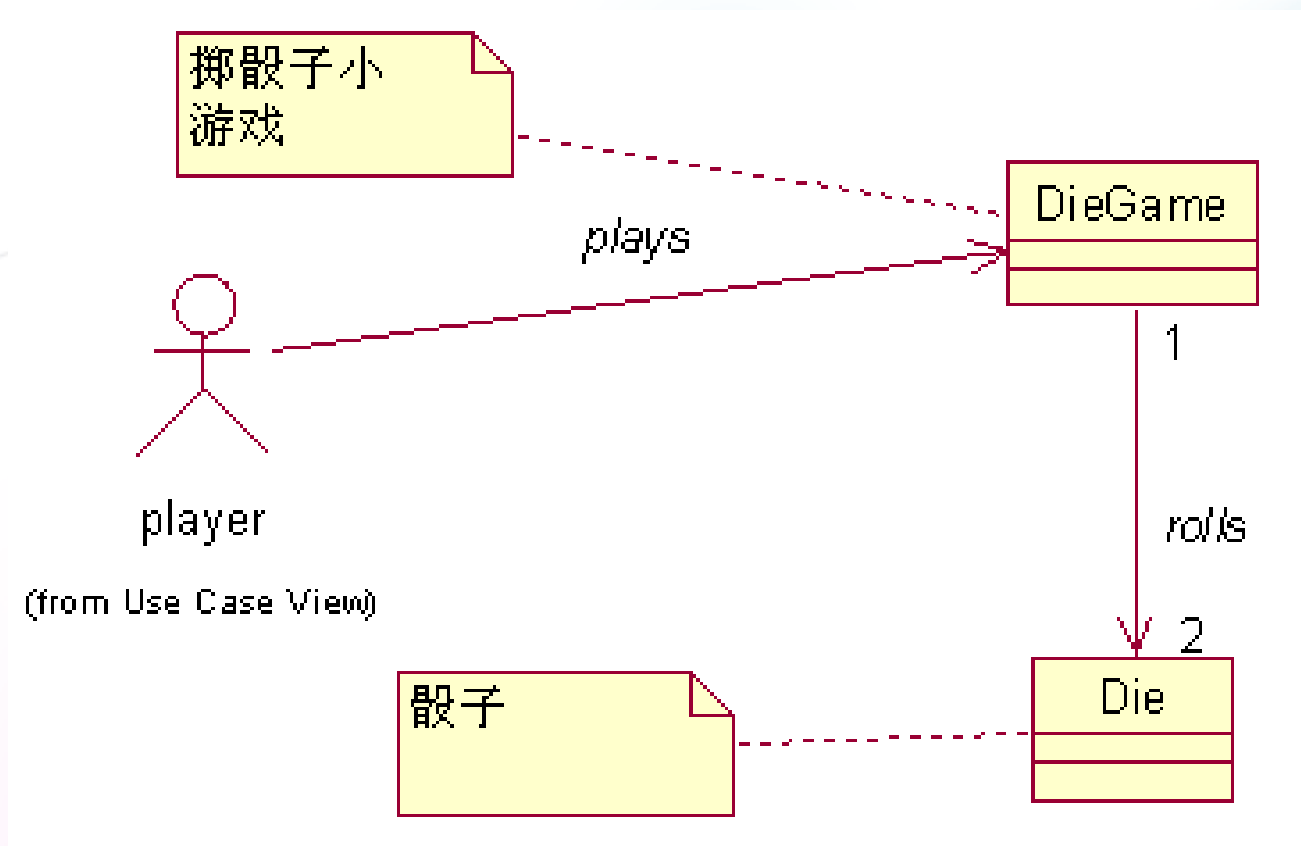
1、定义用例

- 用例：Play a Game(进行游戏)
- 参与者：Player（游戏者）
- 描述：这个用例始于游戏者拾起骰子并投掷骰子。如果2次的总点数是7，游戏者赢，否则输。



UML建模实例：小游戏

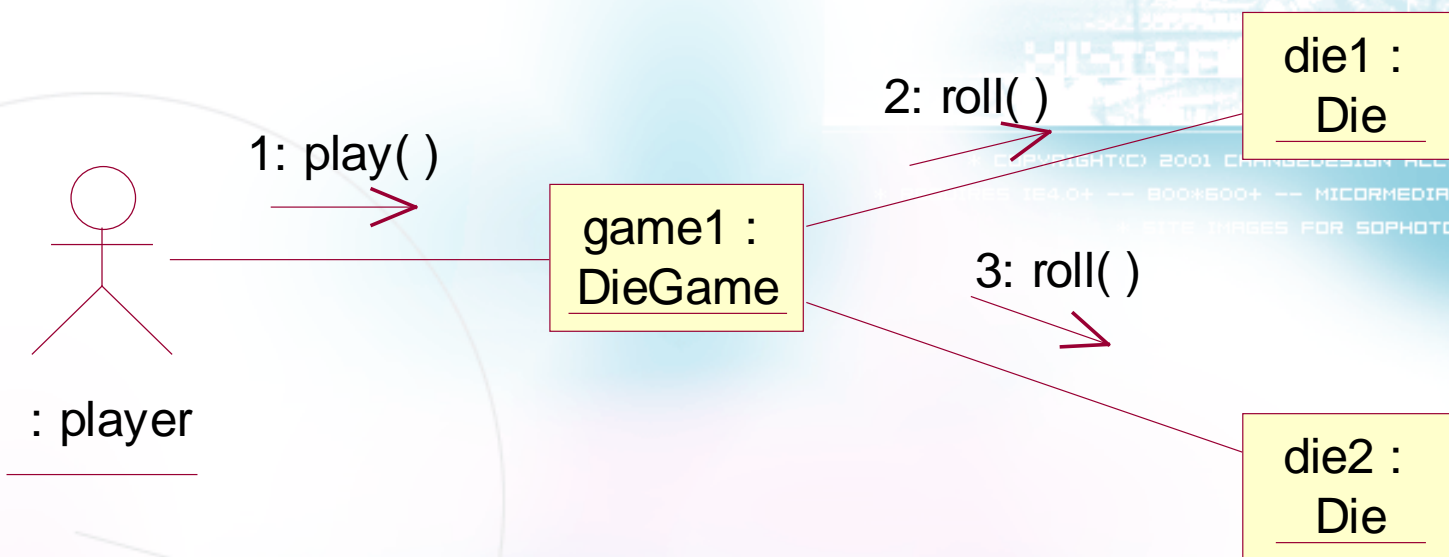
2、初步分析设计类图



UML建模实例：小游戏

3、定义协作图

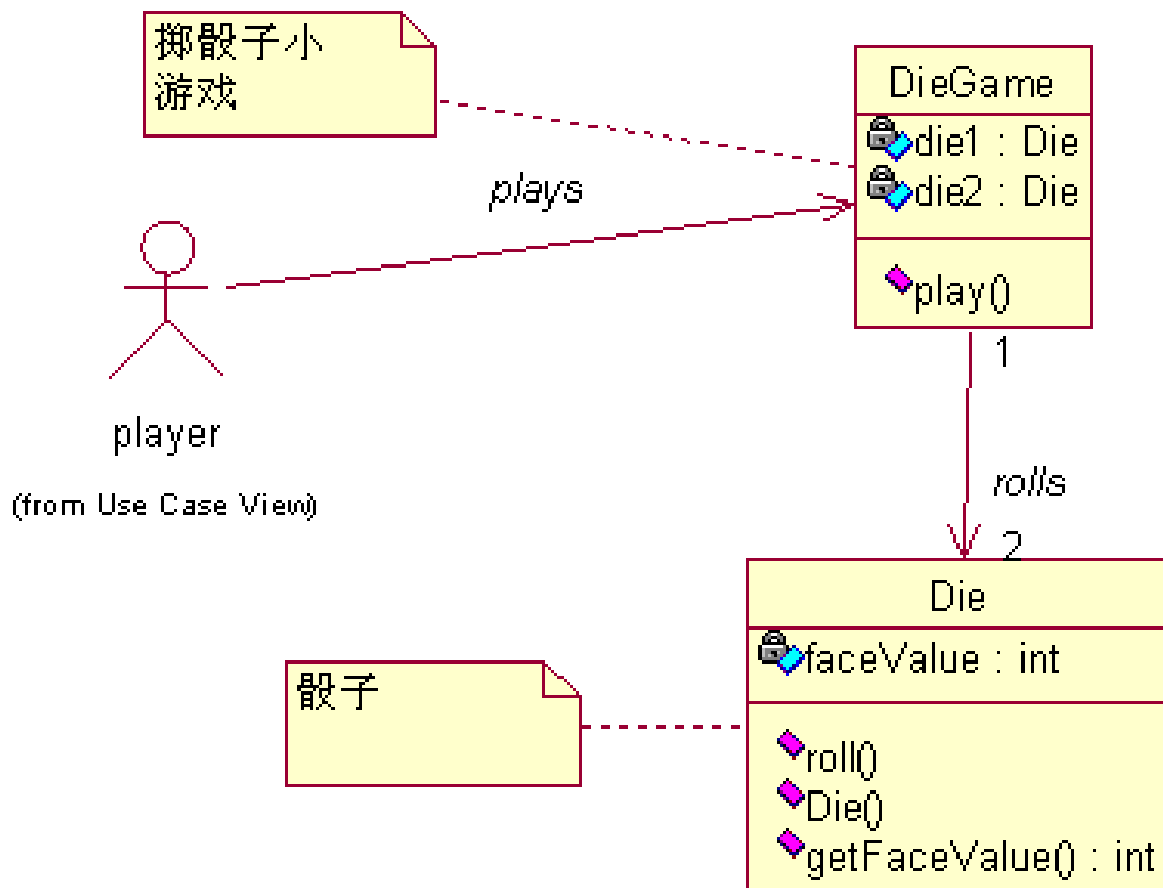
为对象分配职责以及展示对象间如何通过消息进行交互。



UML建模实例：小游戏


4、进一步细化类图


因为一个play消息被发送到DieGame实例，所以DieGame需要一个play()方法，而Die需要一个roll()方法。





UML建模实例：小游戏

Die

 faceValue : int

 roll()

 Die()

 getFaceValue() : int

```
public class Die
```

```
{ private int faceValue;
```


```
public void roll() { }
```


```
public Die() { }
```


```
public int getFaceValue() { }
```

```
}
```

DieGame

 die1 : Die

 die2 : Die

 play()

```
public class DiceGame {
```

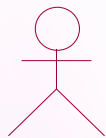
```
private Die die1=new Die();
```

```
private Die die2=new Die();
```

```
public DiceGame() { }
```

```
public boolean play(){ }
```

```
}
```



player

(from Use Case View)

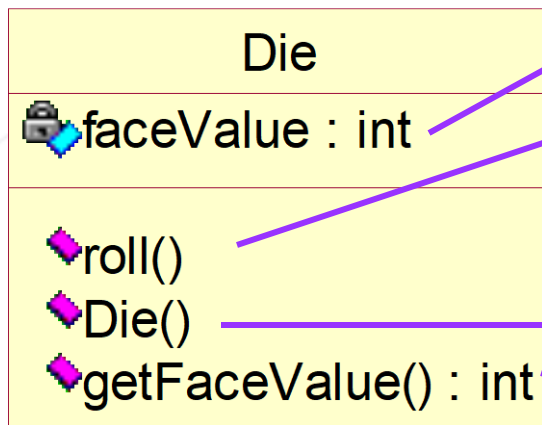
```
public class Player {
```

```
public static void main(String[] args) { }
```

```
}
```


UML建模实例：小游戏

5.1、Die类的实现



```
public class Die
{
    private int faceValue;
    public void roll() {
        faceValue=1+(int)(Math.random() * 6);
    }
    public Die() { }
    public int getFaceValue() {
        return faceValue;
    }
}
```

UML建模实例：小游戏

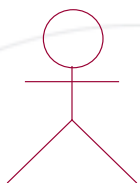
5.2、DieGame类的实现



```
public class DiceGame {
    private Die die1=new Die();
    private Die die2=new Die();
    public DiceGame() { }
    public boolean play(){
        die1.roll();
        int fv1=die1.getFaceValue();
        die2.roll();
        int fv2=die2.getFaceValue();
        System.out.println("Your total score is:
        "+(fv1+fv2));
        boolean win=false;
        if(((fv1+fv2)==7)    win=true;
        return win;
    }
}
```

UML建模实例：小游戏

5.3、Player类的实现



player

(from Use Case View)

```
public class Player {  
    public static void main(String[] args) {  
        DiceGame game1=new DiceGame();  
        if(game1.play()==true){  
            System.out.println("You win the game!");  
        }else{  
            System.out.println("You fail!try again!");  
        }  
    }  
}
```

本章内容回顾

- UML的组成和常用元素
- UML中的关系
- UML 的图以及不同图的划分和类别
- UML的双向工程
- 两个UML建模实例

* CHANGEDESIGNSTUDIO V1.0

* COPYRIGHT(C) 2001 CHANGEDESIGN ALL RIGHT RESERVED
* RESOURCES 164.0+ -- 800*600+ -- MICROMEDIA FLASH/5 PLUGIN
* WEBSITE: WWW.CHANGEDESIGN.COM SITE JURGES FOR SOPHOTO AND TONYSTONE