Chapter 6

- **Requirements Modeling: Scenarios, Information, and Analysis Classes**

Slide Set to accompany

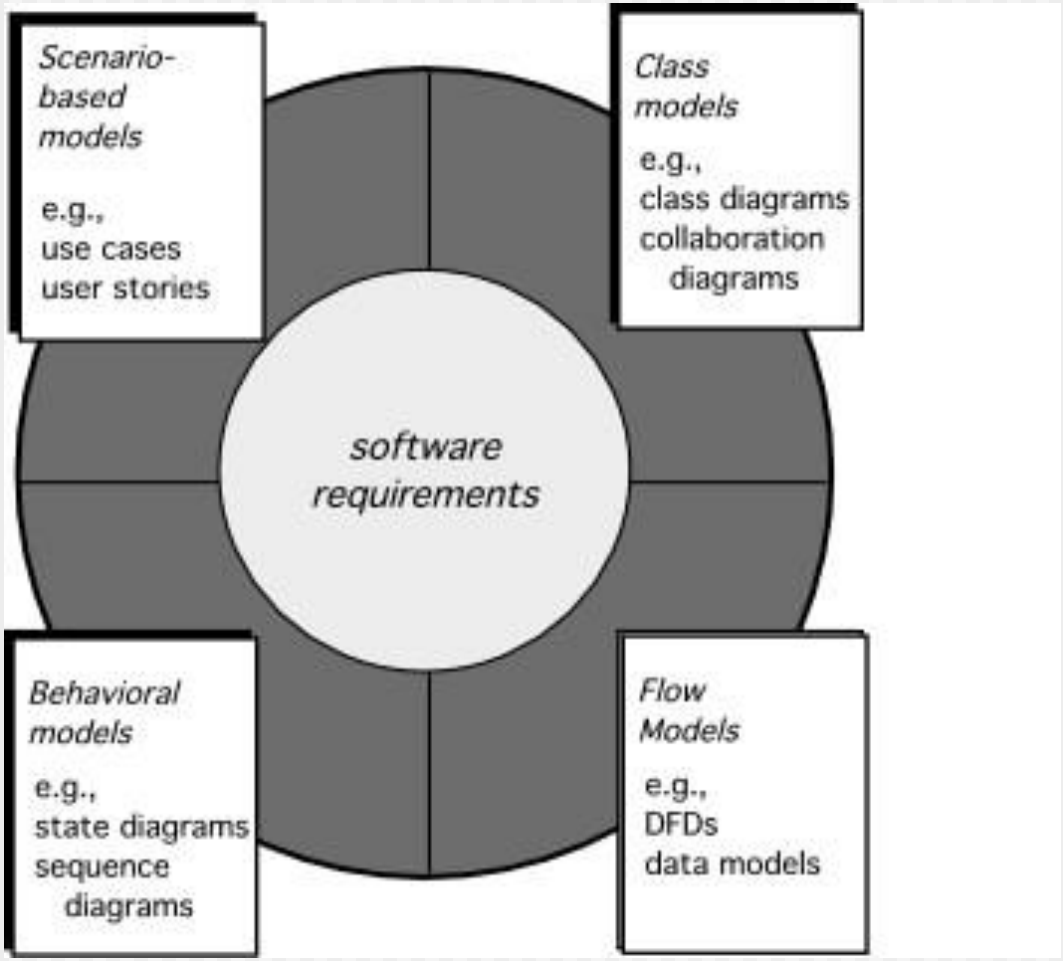
Software Engineering: A Practitioner's Approach, 7/e

by Roger S. Pressman

Requirements Analysis

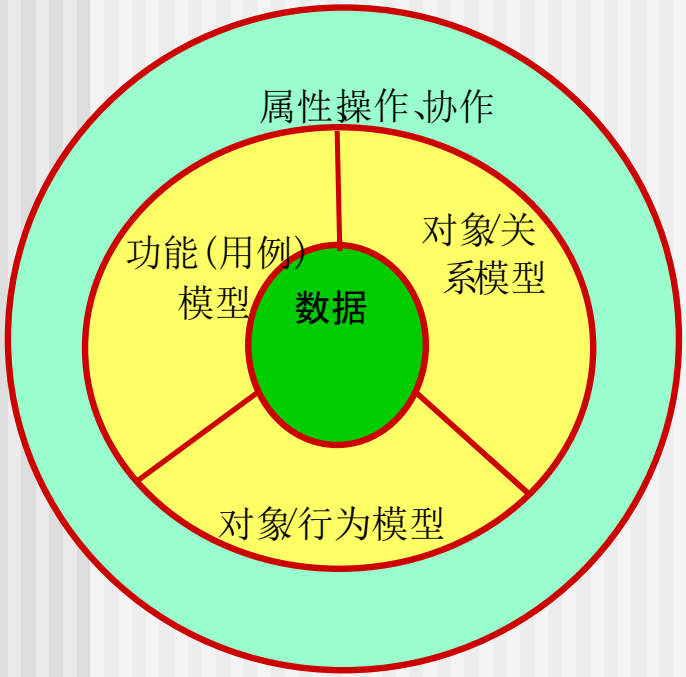
- 需求分析的本质是：通过建立概念化的分析模型，对收集的需求进行提炼和审查。
- 需求分析模型包括:
  - Scenario-based models of requirements from the point of view of various system “actors”
  - Flow-oriented models that represent the functional elements of the system and how they transform data as it moves through the system
  - Data models that depict the information domain for the problem
  - Class-oriented models that represent object-oriented classes and the manner in which classes collaborate to achieve system requirements
  - Behavioral models that depict how the software behaves as a consequence of external “events”

Elements of Requirements Analysis

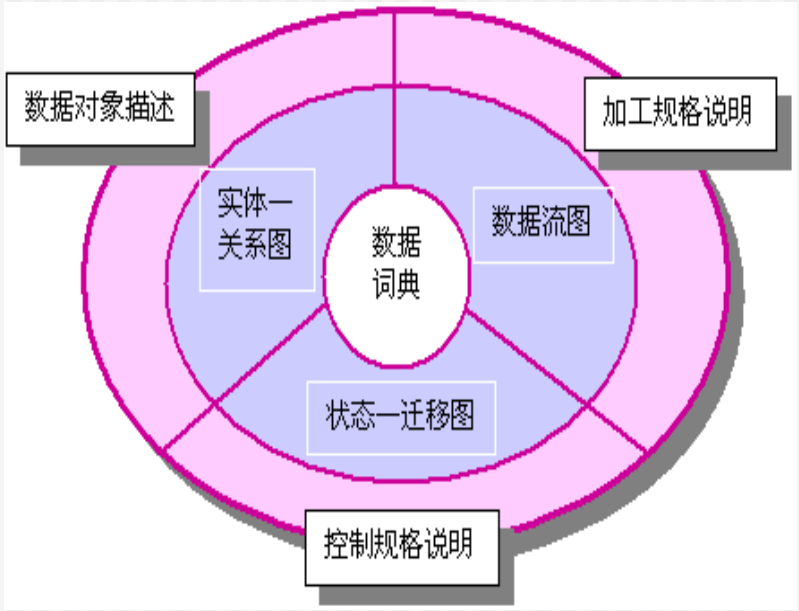


Two Analysis Approaches

- Structured analysis (SA)　结构化分析方法
- Object –oriented analysis (OOA) 面向对象分析方法



OOA



SA

OOA & UML

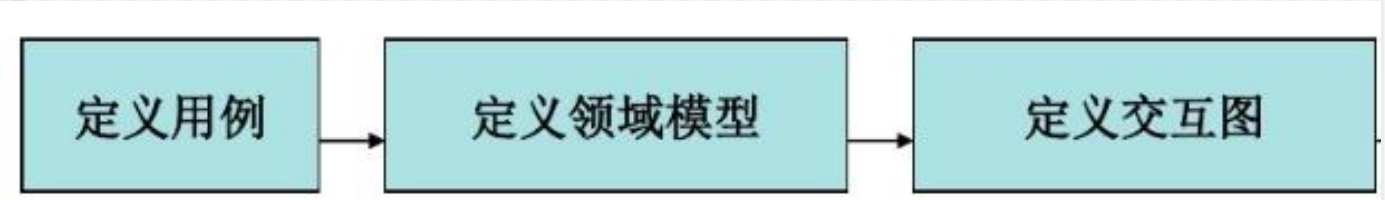
- 现实世界由实体及其相互关系构成，可将其理解为问题空间（现实世界）中的对象。
- OOA方法通过将其映射为解空间（软件系统）中的对象及其关系来寻求问题的求解。

- 一般使用UML （Unified Modeling Language）完成需求分析模型的定义 。
- UML是一种标准图形表示法。

Object Oriented Analysis

- OOA分析模型包括三类：用例模型、（静态）领域模型和（动态）行为模型。

- OOA分析过程



Object Oriented Analysis

- 第一步：建立用例模型
  - 从不同用户的角度获取并描述功能需求，核心是用例。
- 第二步：建立领域模型
  - 发现和描述用例中涉及的对象和概念，核心是类图。
- 第三步：建立动态(交互)模型
  - 分析用例实现中，对象之间的交互行为，核心是时序图、状态图等

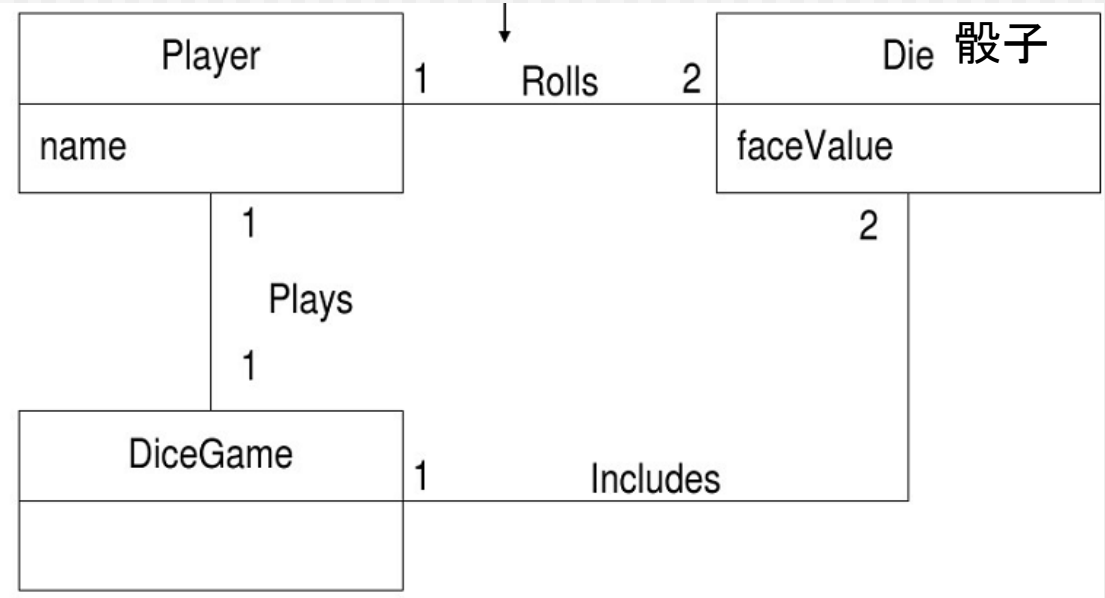
OOA分析实例(1)

- 如开发一小游戏: 游戏者掷两个骰子，如果总点数为7，则游戏玩家赢得比赛，否则输掉比赛。
- 分析步骤1：用例及用例描述

用例: Play Game（进行游戏）
参与者： Player （游戏者）
描述：
游戏者请求掷骰子；
系统展示骰子结果；
判断胜负：如果骰子总点数是7，则游戏者赢，否则游戏者输。

OOA分析实例(2)

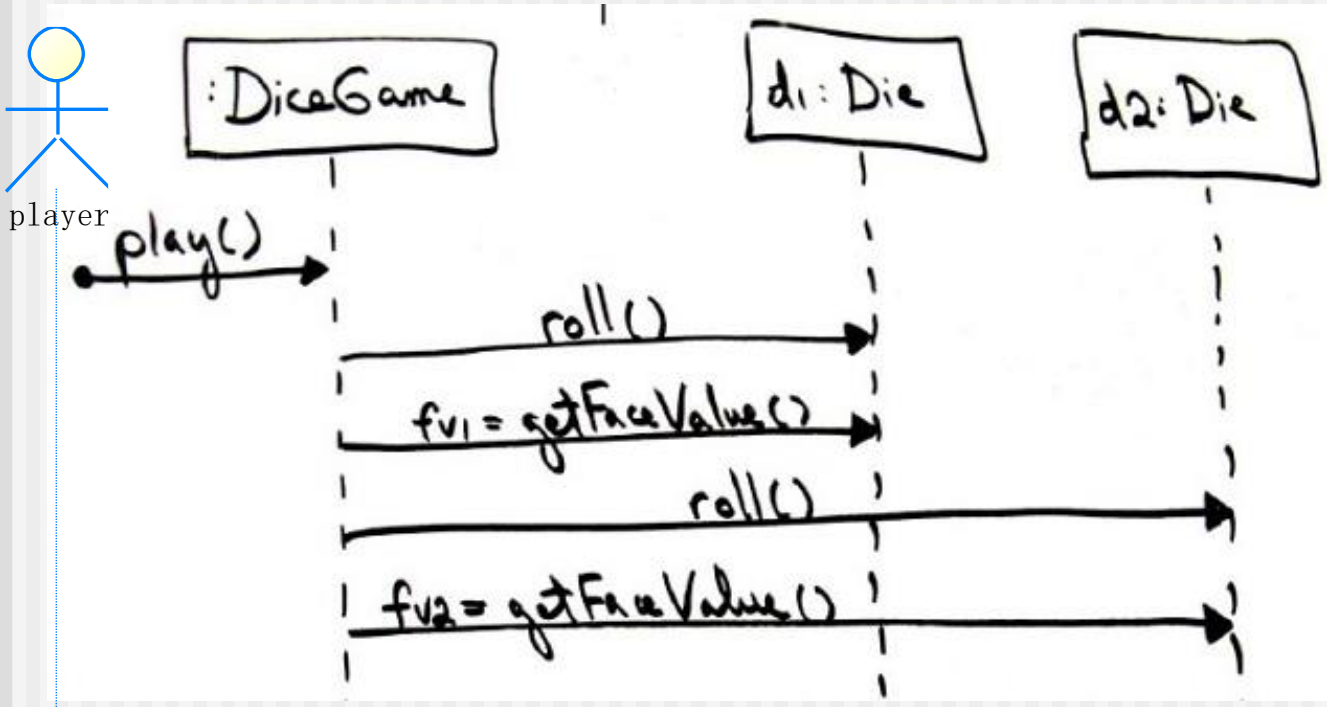
- 分析步骤2：根据用例描述建立领域模型
  -有几个对象？分别是什么？



OOA分析实例(3)

- 分析步骤3：根据用例描述建立交互模型

  -利用时序图，找出参与对象在该用例中的职责或行为



第一步：用例建模

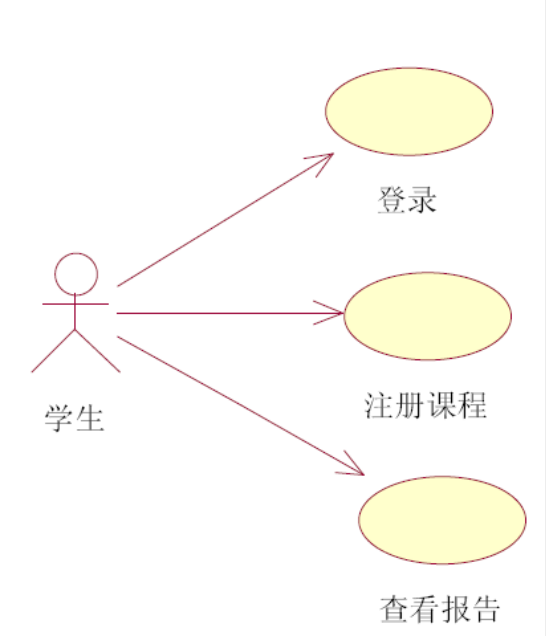
- 用例表示从执行者的角度观察到系统的功能和外部行为。
- 用例图主要描述用户需求，强调谁在使用系统，系统可以实现哪些功能目标。
- 用例特征

  用例都是动宾结构

  用例是相互独立的

  用例由参与者启动

  有可观测的执行结果



如何识别用例

- To begin developing a set of use cases, list the functions or activities performed by a specific actor.

识别用例最好的方法就是从分析系统的参与者（可以是角色、外部系统、设备甚至时钟信号等）开始，考虑每一个参与者是如何使用系统的。

练习—道路修复系统

需求陈述：

市民可以登录网站报告坑洼的位置和严重程度。每个被报告的坑洼，将登记到“市政部门的修复系统”中，并被赋予一个标识号，按街道地址、大小、位置、城区和修复的优先级（坑洼严重程度）存储。

市政人员确认坑洼情况属实后，填写派工单并对派工执行情况进行登记。内容包括：位置和大小、修理队的人数、使用的修理装备、修复所用的时间、坑洼状况（正在工作、已被修理、临时修理、未修理）、使用填料的数量和修理的开销（由使用的时间、人数、使用的材料的装备计算得到）。

坑洼经修复后，可输出坑洼的损害报告，包括市民的姓名、地址、电话号码、损害类型和修复总费用。

练习—道路修复系统

此问题中，系统外部参与者包括：

市民---------- 登记报告坑洼

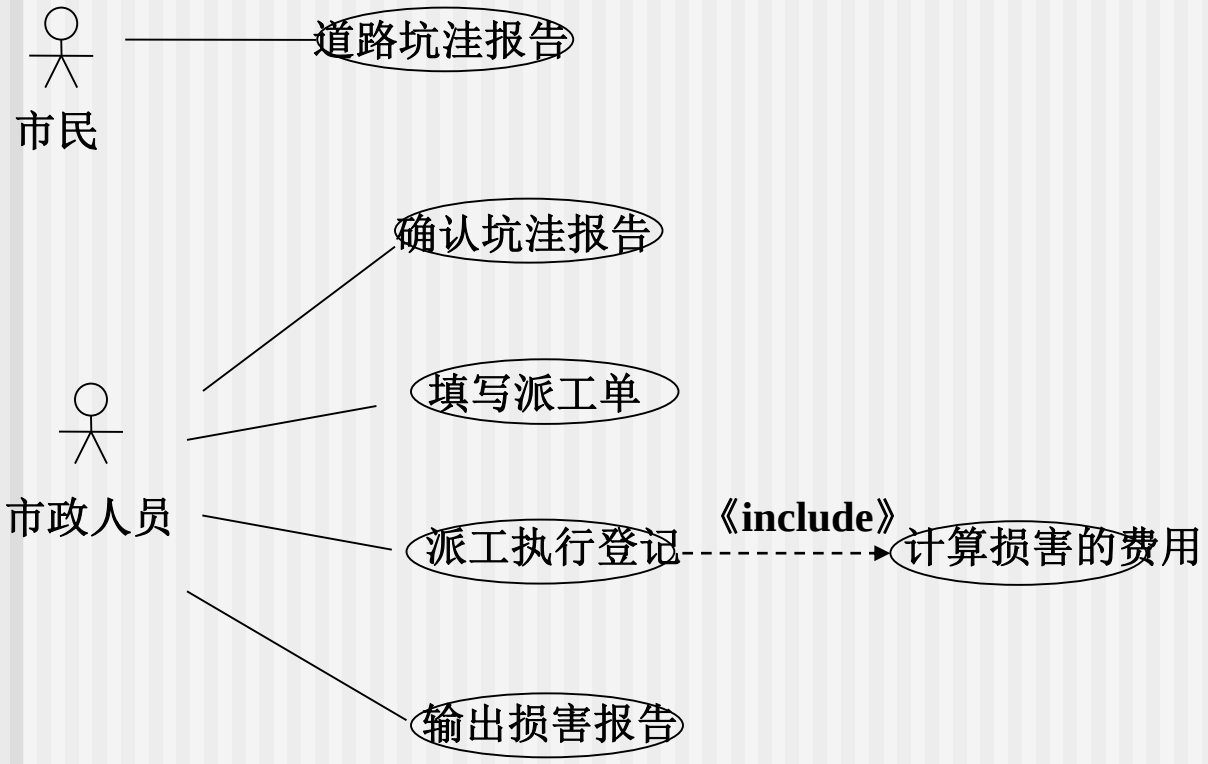
市政工作人员------确认坑洼报告

填写派工单

派工执行登记

输出损害报告

练习—道路修复系统



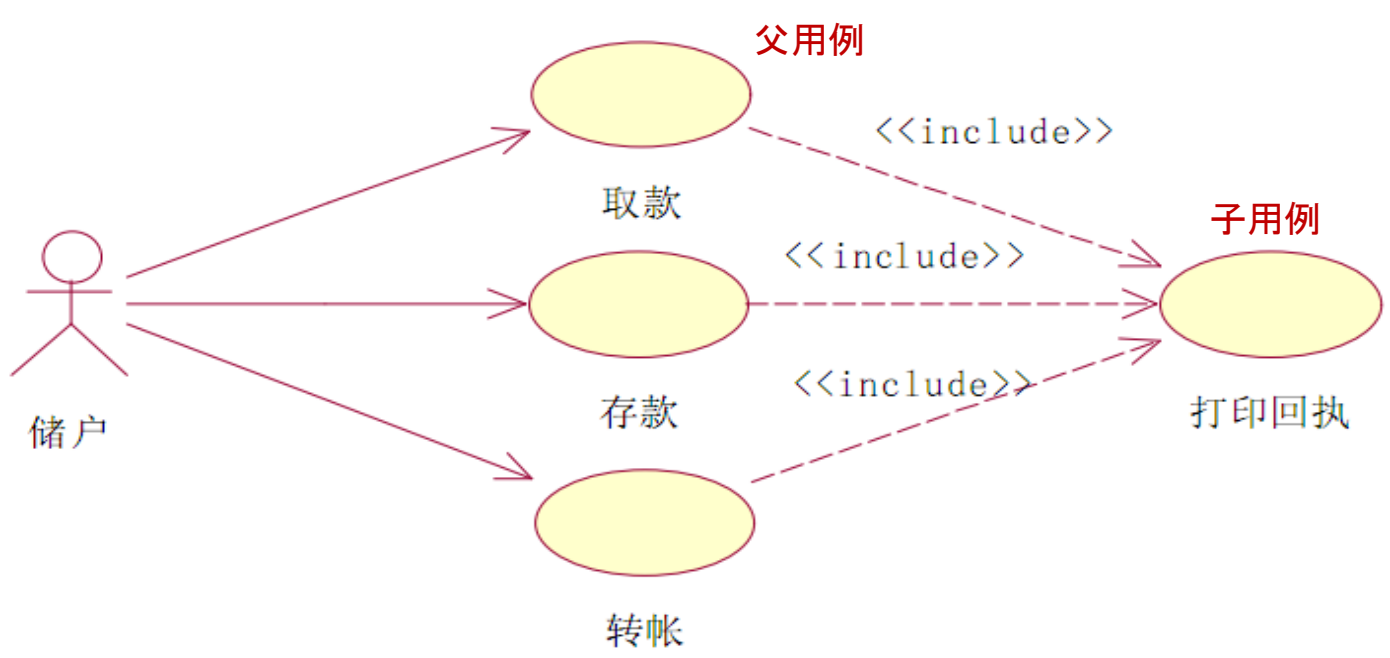
用例模型主要包含4种元素：执行者、用例，执行者与用例间关联、用例间关系。

用例图(1)

- 《include》关系

  父用例执行过程中会用到子用例(无条件执行)。

  子用例一般表示为公共功能。

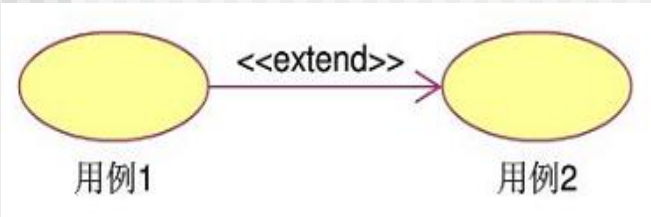


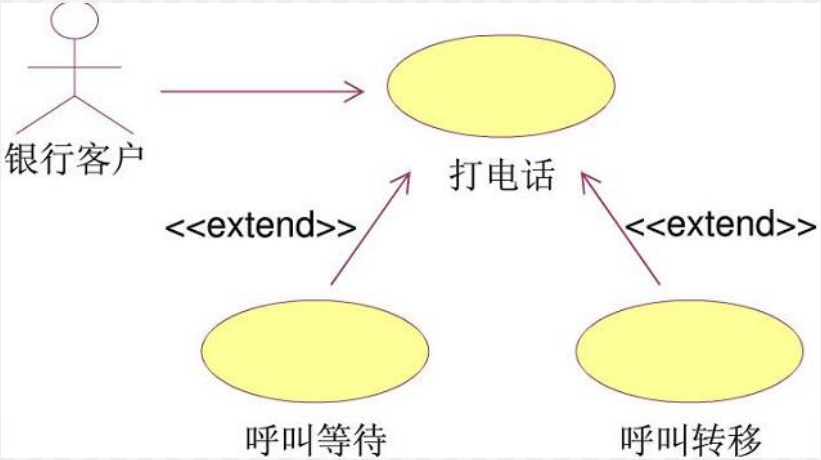
用例图(2)

- 《extend》关系

父用例在特定情况下（称为扩展点）会用到子用例(有条件执行)

子用例在一般表示为异常功能。



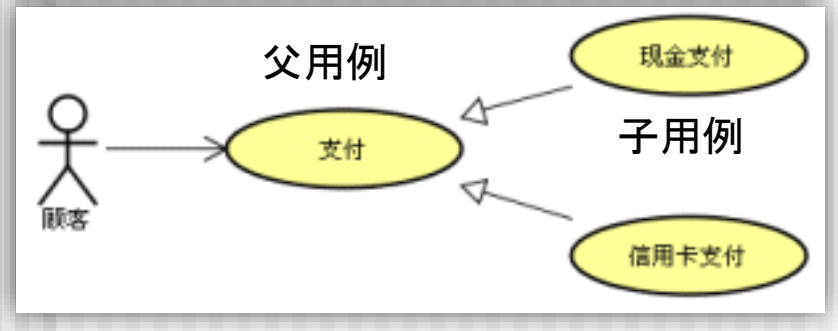


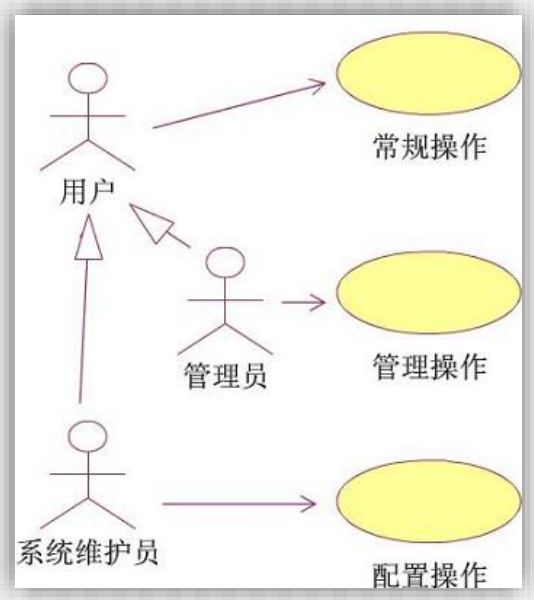
用例图(3)

- 泛化(generalization)关系

当多个用例共同拥有一种类似的行为时，可以将它们的共性抽象成为父用例，其他用例作为子用例。

该关系也可存在于参与者之间。





用例描述

- 单纯的用例图并不能描述完整的信息，需要用文字描述不能反映在图形上的信息。
- 用例描述是将用例发生的各种场景描述出来，表示参与者与系统交互时双方的行为，即参与者做什么，系统做什么反应。
- 场景能真实反映参与者在用例执行中可能遇到的各种情形。
- 可以使用用例模版或活动图描述用例执行中的各种场景。

用例描述—使用用例模版

- 用例描述模版的内容大致包括：用例名、参与者，前置条件，后置条件，事件流等。
- 用例的事件流
  - 说明参与者与系统之间的交互过程；
  - 说明用例在不同条件下可以选择执行的多种方案；
- 分为基本流和备选流两类
  - 基本流：描述该用例正常执行的一种场景，系统执行一系列活动步骤来响应参与者提出的服务请求；
  - 备选流：描述用例执行过程中异常的或偶尔发生的一些情况。

用例描述—基于用例模版

用例名称：	ATM取款
描述：	客户持银行卡（本行或其他行）从ATM提取现金
actors:	客户和银行主机
前置条件：	无
基本流：	1. 客户插入银行卡。 2. ATM从银行卡读入卡号（含银行标识和账号），验证卡的有效性。 3. 客户输入密码。 4. ATM验证帐号和密码。 5. ATM显示包括取款在内的服务功能，客户选择“取款”。 6. 客户输入取款额。 7. ATM向银行主机发出卡号、密码、账号和取款额。 8. 银行主机核实账户余额足够否，足够则执行扣款，并向ATM机返回取款成功信息。 9. ATM打印并吐出凭条。 10. ATM清点并吐出现金。
备选流：	4a. 累计3次密码错误： ATM吞卡，[用例失败] 4b. 无此帐号： ATM吞卡，[用例失败] 5a. ATM无现金： ATM不显示“取款”功能，客户可选择其他服务， [用例失败]
扩展点：	无
非功能需求：	ATM响应客户时间不超过15秒

用例描述—基于活动图

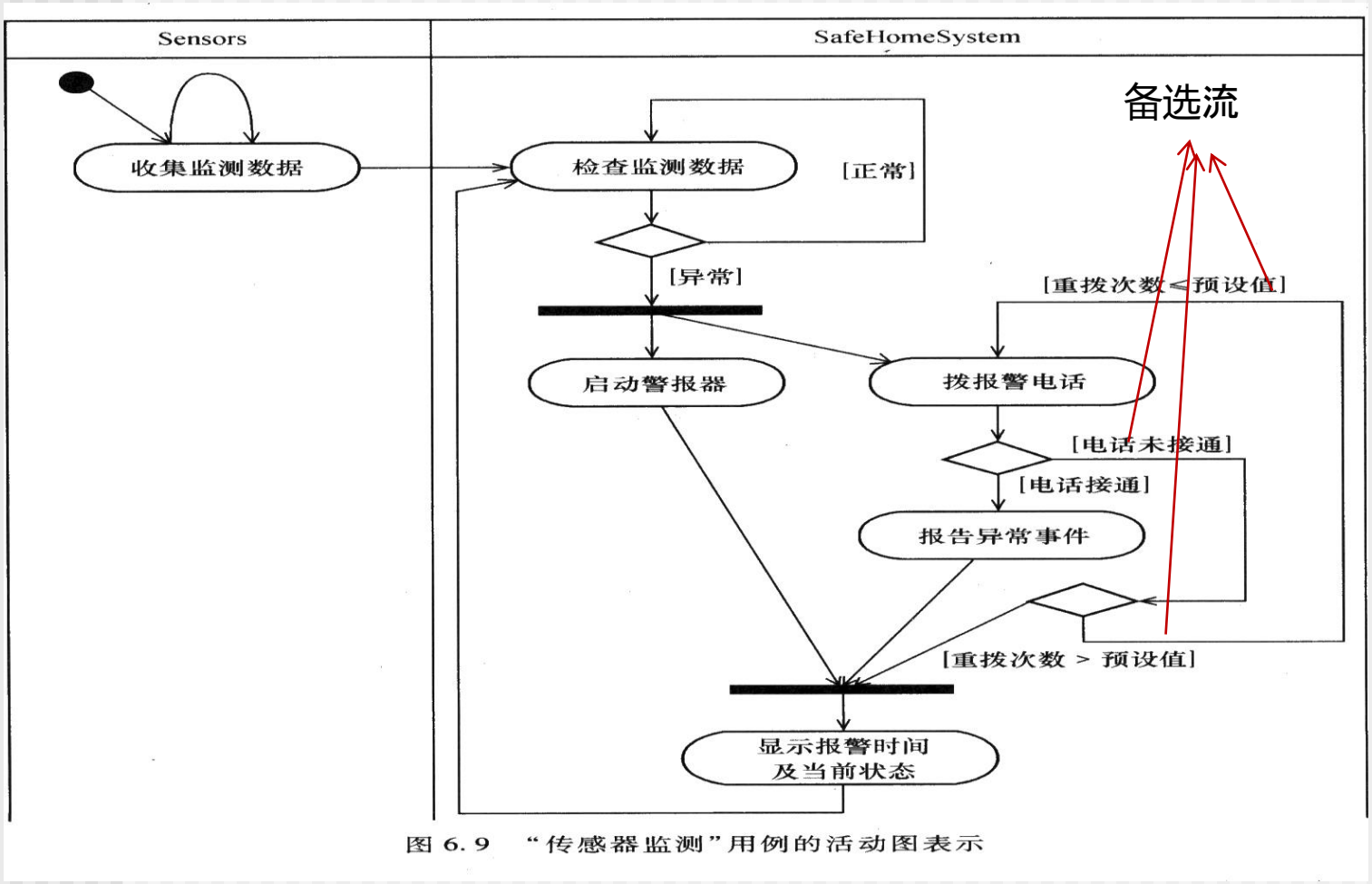


图 6.9 “传感器监测”用例的活动图表示

第二步：数据建模

- focuses attention on the data domain
- creates a model at the customer's level of abstraction
- indicates how data objects relate to one another

What is a Data Object?

- a representation of almost any composite information that must be understood by software.
- can be an external entity, a thing, an event, a role, an organizational unit, a place, or a structure.
- The description of the data object incorporates the data object and all of its attributes.

Data Objects and Attributes

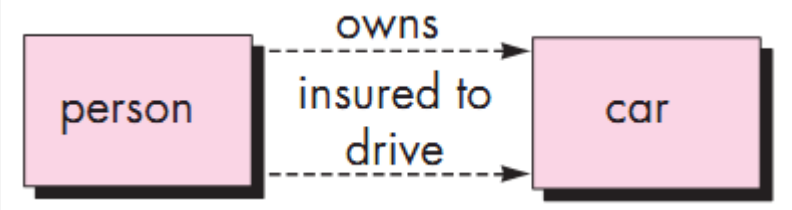
A data object contains a set of attributes that act as an aspect, quality, characteristic, or descriptor of the object

object: automobile

attributes:
- **make**
- **model**
- **body type**
- **price**
- **options code**

What is a Relationship

- Data objects are connected to one another in different ways.
  - A connection is established between **person** and **car** because the two objects are related.
    - A person *owns* a car
    - A person *is insured to drive* a car
- The relationships *owns* and *insured to drive* define the relevant connections between **person** and **car.**
- Objects can be related in many different ways

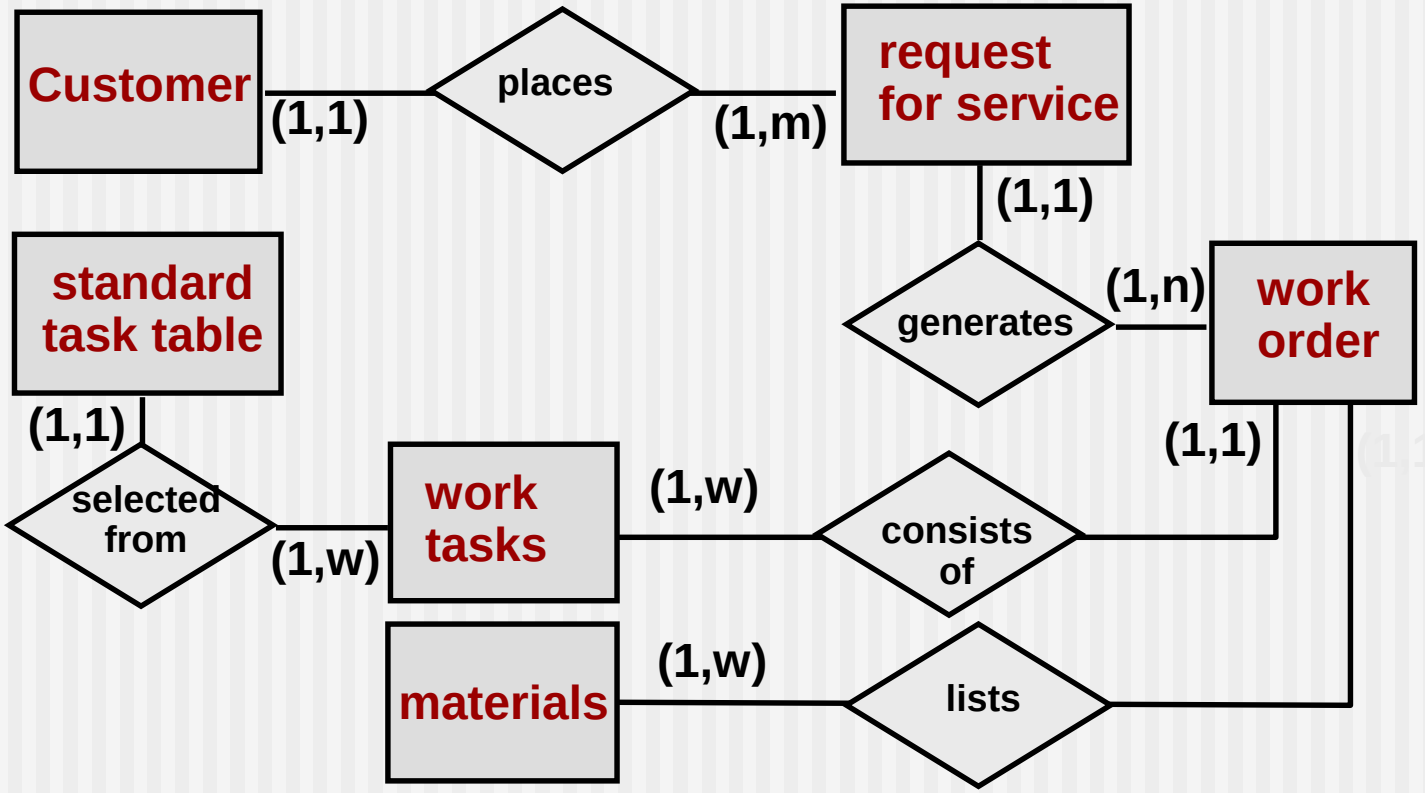


ERD Notation

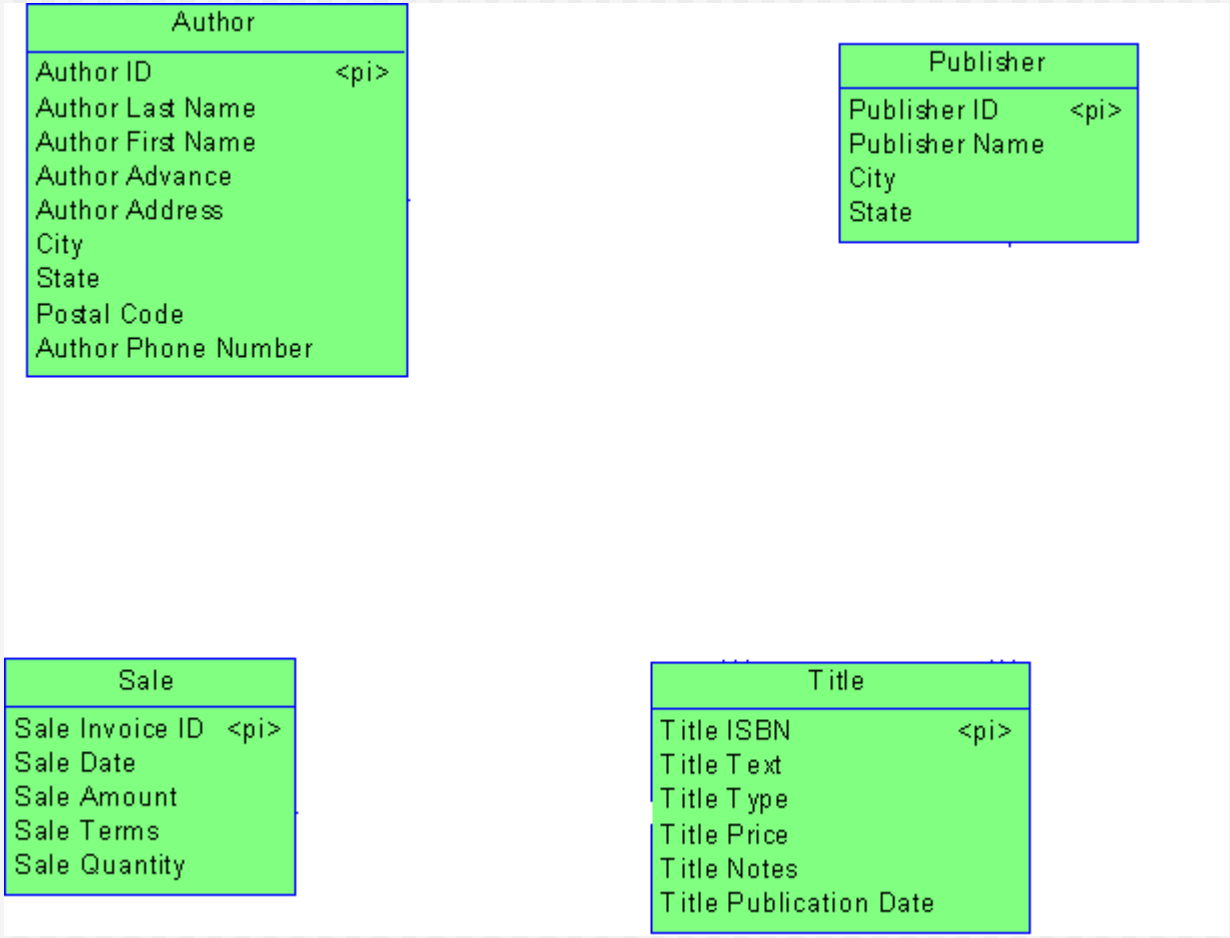
One common form:

Another common form:

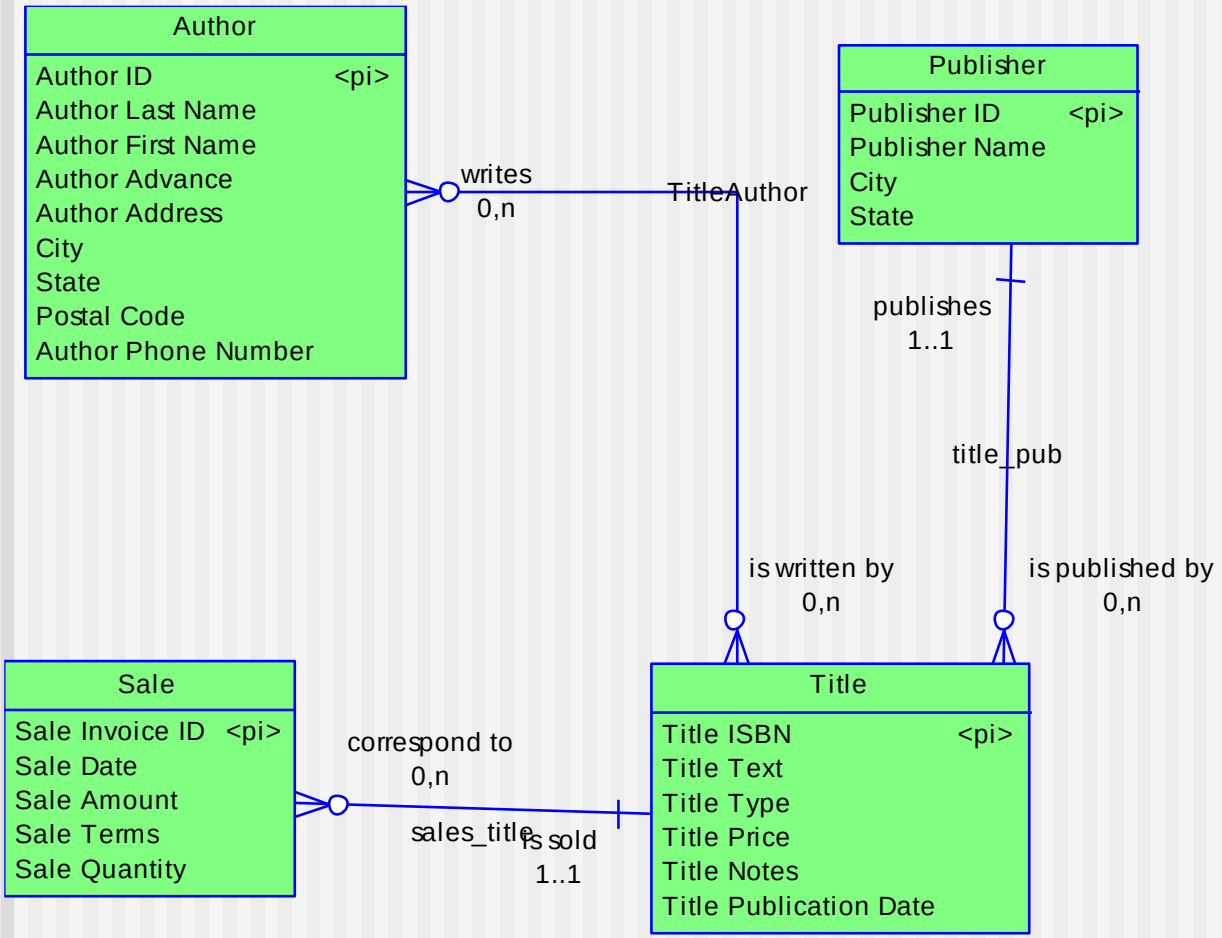
The ERD: An Example



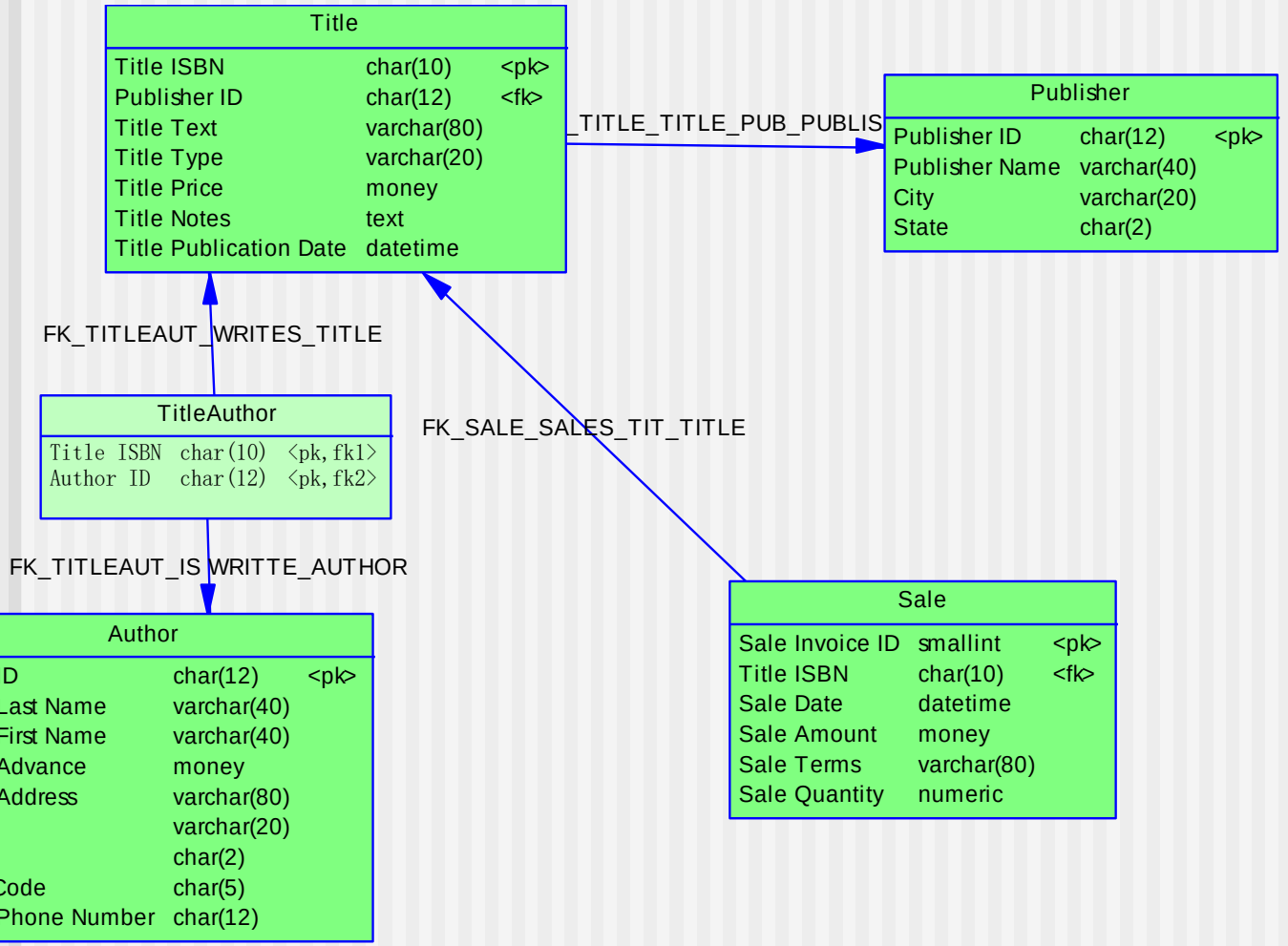
数据建模练习(1)


Author
Author ID <pi>
Author Last Name
Author First Name
Author Advance
Author Address
City
State
Postal Code
Author Phone Number
Publisher
Publisher ID <pi>
Publisher Name
City
State
Sale
Sale Invoice ID <pi>
Sale Date
Sale Amount
Sale Terms
Sale Quantity
Title
Title ISBN <pi>
Title Text
Title Type
Title Price
Title Notes
Title Publication Date

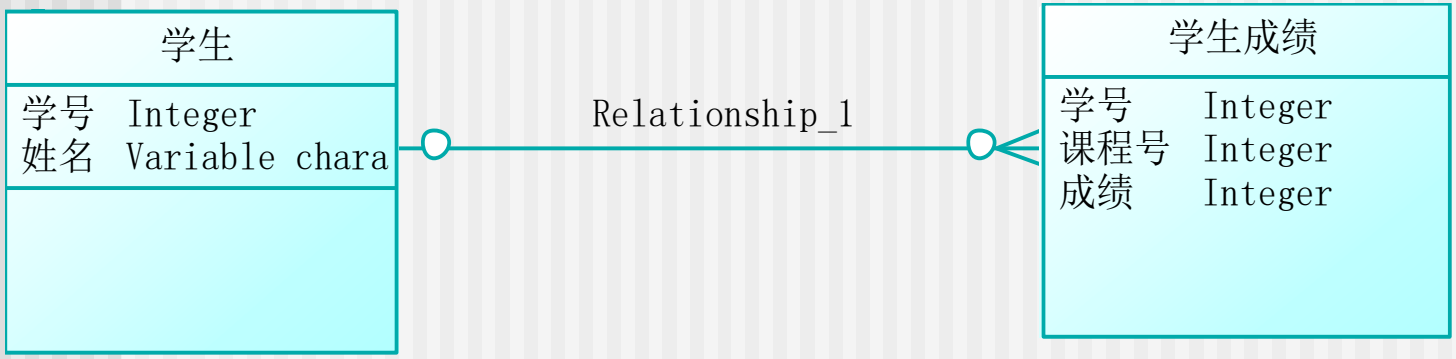
数据建模练习(2)


Author
Author ID <pi>
Author Last Name
Author First Name
Author Advance
Author Address
City
State
Postal Code
Author Phone Number
writes
0,n
TitleAuthor
Publisher
Publisher ID <pi>
Publisher Name
City
State
publishes
1..1
title_pub
is written by
0,n
is published by
0,n
Sale
Sale Invoice ID <pi>
Sale Date
Sale Amount
Sale Terms
Sale Quantity
correspond to
0,n
sales_title
is sold
1..1
Title
Title ISBN <pi>
Title Text
Title Type
Title Price
Title Notes
Title Publication Date

数据建模练习(3)

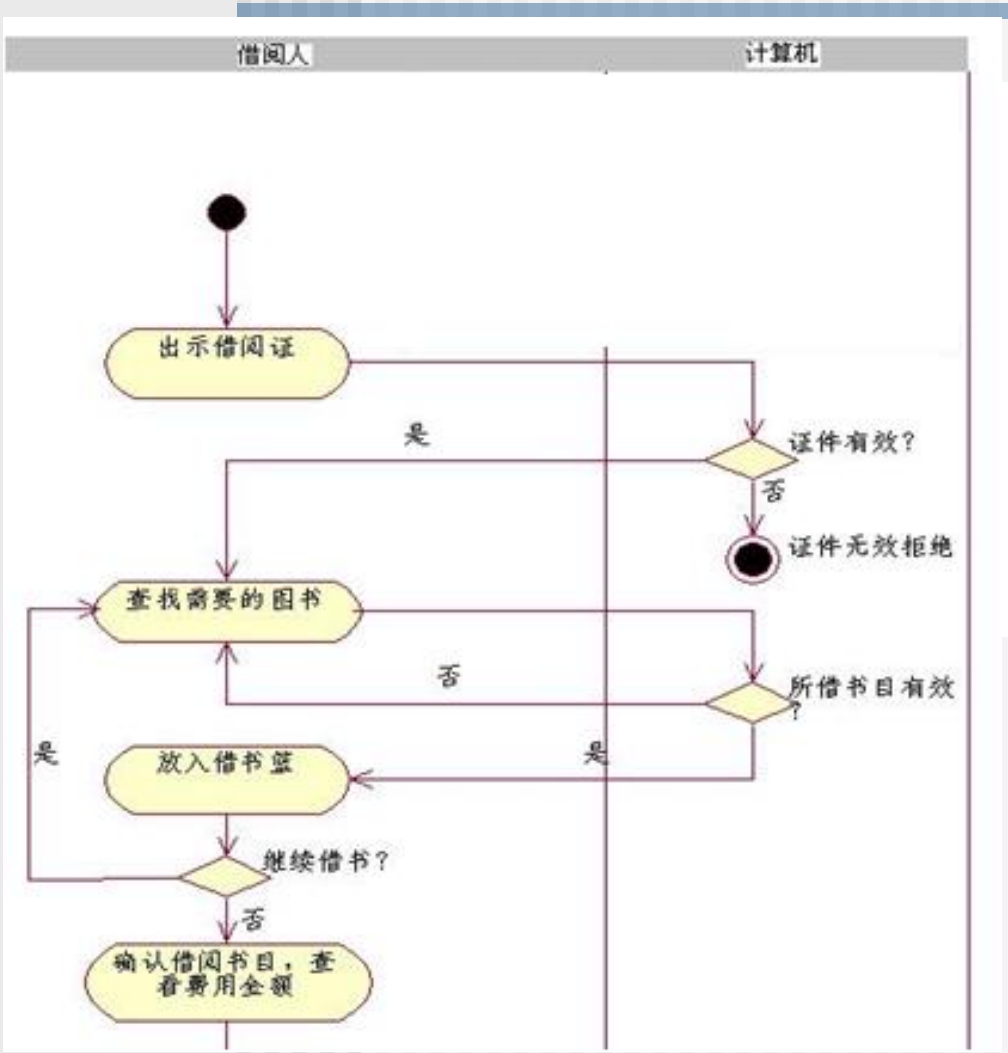

Title
Title ISBN char(10) <pk>
Publisher ID char(12) <fk>
Title Text varchar(80)
Title Type varchar(20)
Title Price money
Title Notes text
Title Publication Date datetime
_TITLE_TITLE_PUB_PUBLIS
Publisher
Publisher ID char(12) <pk>
Publisher Name varchar(40)
City varchar(20)
State char(2)
FK_TITLEAUT_WRITES_TITLE
TitleAuthor
Title ISBN char(10) <pk,fk1>
Author ID char(12) <pk,fk2>
FK_SALE_SALES_TIT_TITLE
FK_TITLEAUT_IS_WRITTE_AUTHOR
Author
Author ID char(12) <pk>
Author Last Name varchar(40)
Author First Name varchar(40)
Author Advance money
Author Address varchar(80)
City varchar(20)
State char(2)
Postal Code char(5)
Author Phone Number char(12)
Sale
Sale Invoice ID smallint <pk>
Title ISBN char(10) <fk>
Sale Date datetime
Sale Amount money
Sale Terms varchar(80)
Sale Quantity numeric

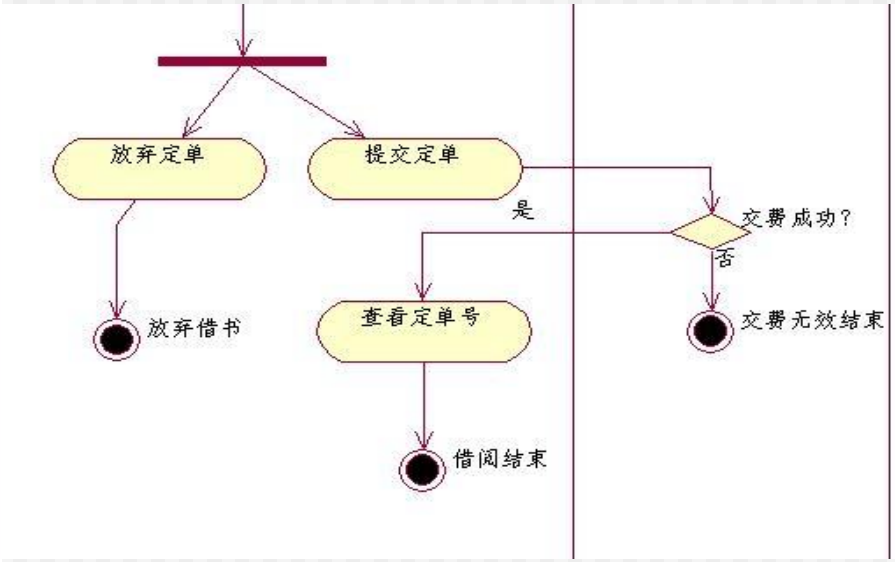
数据建模练习(4)



Right or Wrong

数据建模练习(5)





关键：

信息是否需要持久存储

第三步：领域建模

- 领域模型分析以用例模型作为输入，将系统分解为一组相互协作的概念类(也称分析类)。
- 分析类具有突出业务领域、突出概念性的特征，主要针对问题和职责，不涉及细节。

- 设计阶段，分析类可进一步演化为设计类，提供更多设计表达。

领域模型

- 包括：
  - 类图 (Classes diagram)
  - CRC模型(CRC models)
  - 包图(Packages)

类图建模

- 目标是对领域内的概念类进行分析，并抽象出对应的类及其关系等。

- 具体包括三个步骤：

  1. 从用例场景中寻找概念类

  2. 细化概念类，识别其类型（如边界类、实体类等）

  3. 为概念类添加关联和属性

(1)Identifying Analysis Classes

- Examining the usage scenarios developed as part of the requirements model and perform a “grammatical parse”
  分析用例场景
- Classes are determined by underlining each noun or noun phrase and entering it into a simple table.
  找出其中的名词或名词短语作为分析类备选
- But what should we look for once all of the nouns have been isolated?
  如何判断分离出的词是否是分析类？

Potential Classes

- 借助以下原则识别可能的分析类

- 保留信息：只有当相关信息必须被记录才能保证系统正常工作时，潜在类在分析过程中才是有用的。
- 所需服务：潜在类必须具有一种可确认的、能用某种方式改变类的属性值的操作。
- 多个属性：在需求分析过程中，焦点应在于“主”信息；事实上，只有一个属性的类可能在设计中有用，但是在分析活动阶段，把它作为另一个类的某个属性可能更好。
- 公共属性：可以为潜在类定义一组属性，这些属性适用于类的所有实例。
- 公共操作：可以为潜在类定义一组操作，这些操作适用于类的所有实例。
- 必要需求：在问题空间中出现的外部实体，或者任何系统解决方案的运行所必需的信息，几乎都被定义为需求模型中的类。

实例：学生选课用例

用例名称	学生选课(Select Course)
执行者	学生
用例简述	学生通过该用例选择感兴趣的课程。
前置条件	学生已通过身份验证。
后置条件	系统正确记录学生选课信息。
基本流程	1．学生进入选课界面，用例开始。 2．学生查询课程。 3．系统显示可选课程。 4．学生选中感兴趣的课程。 5．系统验证课程是否可选，如果可选，则保存学生选课信息。 6．系统显示“选课成功”。 7．用例结束。
备选流程	5a. 课程如不可选 1．系统提示不可选的原因（如人数已满）。 2．学生重新选课。 3．系统再次验证直至成功。
字段列表	学生信息包括以下字段：学号、姓名、密码、专业、班级。 课程信息包括以下字段：课程号、课程名、开课时间、课程学时、选修人数。
非功能需求	系统响应时间应该在30秒以内。

实例：学生选课用例

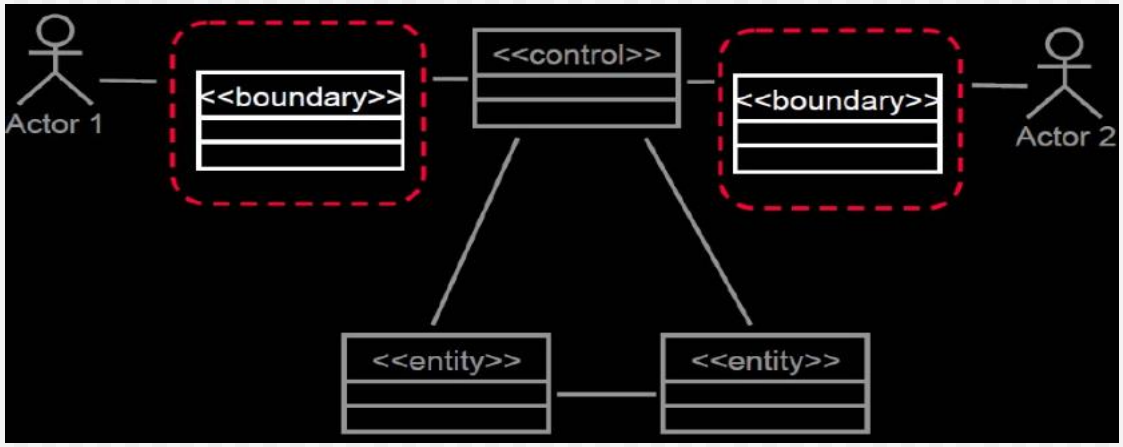
- 通过分析，可以确定以下分析类

类名	类名	类型
Student	学生	实体类
SelectCourseForm	选课界面	边界类
Course	课程	实体类
SelectSuccessForm	提示选课成功（选课成功界面）	边界类
SelectedCourse	选课信息	实体类

分析类是**概念层次**的东西，源于问题空间，与实现方式无关。

Class Types

- 《Entity》 classes 表示系统中的持久信息或贯穿应用的数据封装
- 《Boundary》 classes 实现用户与系统的交互，如用户界面或外部接口
- 《Control》 classes 实现对用例行为的封装，管理、调度其它类



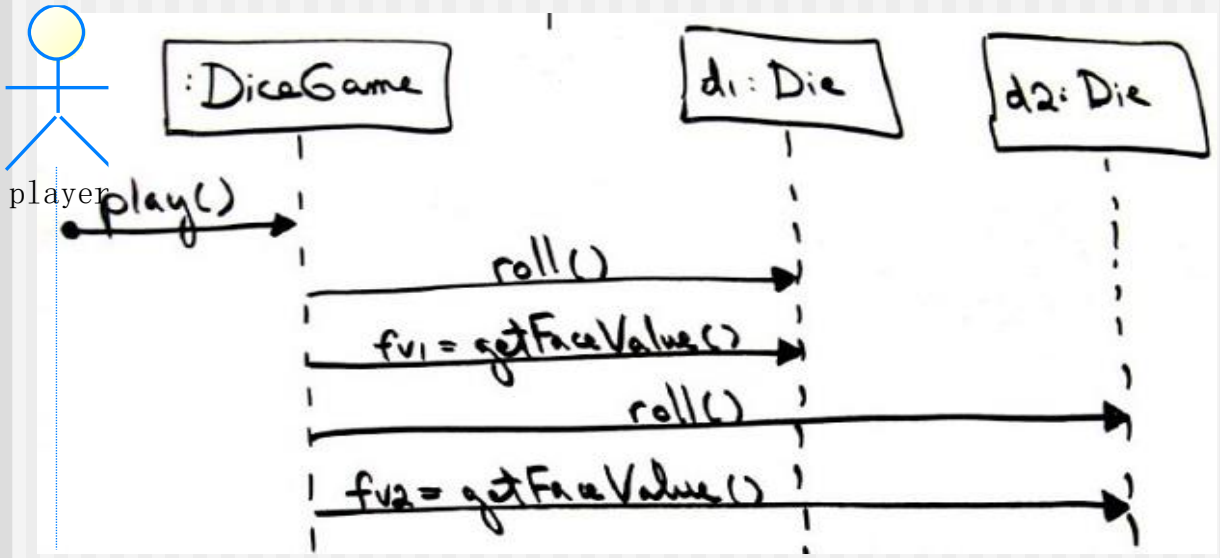
(2) Defining Attributes

- Attributes describe a class that has been selected for inclusion in the analysis model.
- Build two different classes for professional baseball players
  - **For Playing Statistics software:** name, position, batting average, fielding percentage（守备率）, years played, and games played might be relevant
  - **For Pension Fund software（养老系统）:** average salary, credit toward full vesting, pension plan options chosen, mailing address, and the like.

(3) Defining Operations

识别类所属行为的两种方式

- 方法1：Do a grammatical parse of a processing narrative and look at the verbs
- 方法2：结合动态(交互)建模，识别分析类的行为。



Example-学生选课用例

基本流程	1. 学生进入选课界面，用例开始。 2. 学生查询课程。 3. 系统显示课程。 4. 学生选择课程。 5. 系统验证课程是否可选，如果可选，保存学生选课信息。 6. 系统显示选课成功。

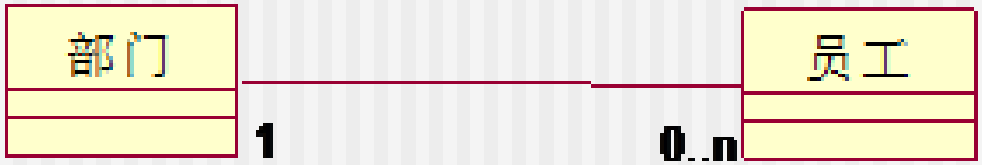
类名	属性	操作
Student	- SID: String - SName: String - SPassword: String - SSpecialty: String	+ viewCourse(): void + selectCourse(): void
Course	- CID: String - CName: String - CStartDate: Date - CPeriod: int - CStudentAmount: int	+ validateCourse(Course c): boolean + displayCourse(Course[] c) :void
SelectedCourse		+ saveSelectedCourse(Course c): boolean

(4) Defining Relationships

1.关联关系

关联关系可以使一个类可以引用另一个类的属性和方法。

一般使用成员变量来实现。




```
class  Department{
  private Employee e;
  //private Employee[] es;
  public getCost(){
     return e.getcost();
  }
}
```

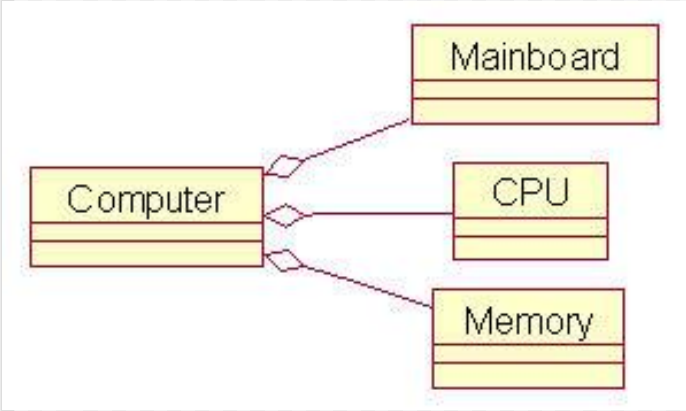
```
class Employee{
   private name;
   private cost;
   public getCost(){
      return cost;
   }
}
```

(4) Defining Relationships

2. 特殊关联之—聚合关系

表示整体和部分的关系，整体与部分可以分开,使用带空心菱形的实线来表示

如： 电脑包括主板、CPU等

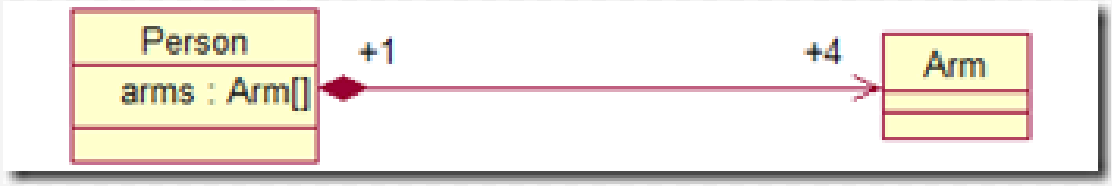



```
public class Computer{
  private CPU cpu;
  public CPU getCPU(){
     return cpu;
  }
  public void setCPU(CPU cpu){
     this.cpu=cpu;
  }
  //开启电脑
  public void start(){
     //cpu运作
     cpu.run();
  }
```

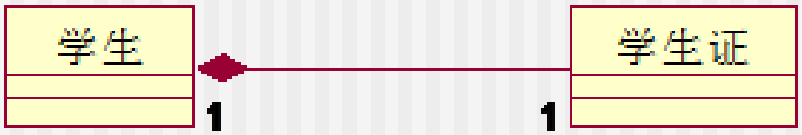
(4) Defining Relationships

3. 特殊关联之—组合关系

也是整体与部分的关系，但是整体与部分不可以分开,使用实心带菱形的实线来表示。



组合关系中整体类和部分类必须有相同的生命周期。




```
public class A {
  private B b;
  public A() {
    this.b = new B();
  }
}
```

4) Defining Relationships

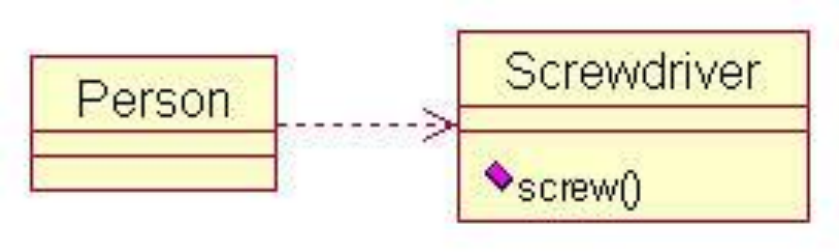
4. 依赖关系

即一个类的实现需要另一个类的协助。

多数情况下，依 赖关系表现在某个类的方法使用另一个类的对象作为参数。

比如你要拧螺丝，要借助(也就是依赖)螺丝刀(Screwdriver)来完成

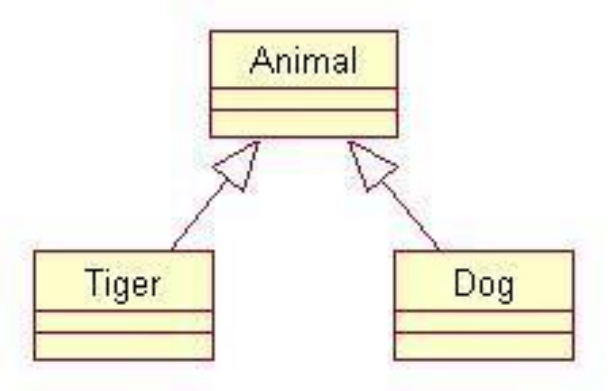
```
public class Person{
   public void screw(Screwdriver screwdriver){    /** 拧螺丝 */
     screwdriver.screw();
   }
}
```



(4) Defining Relationships

5. 泛化关系

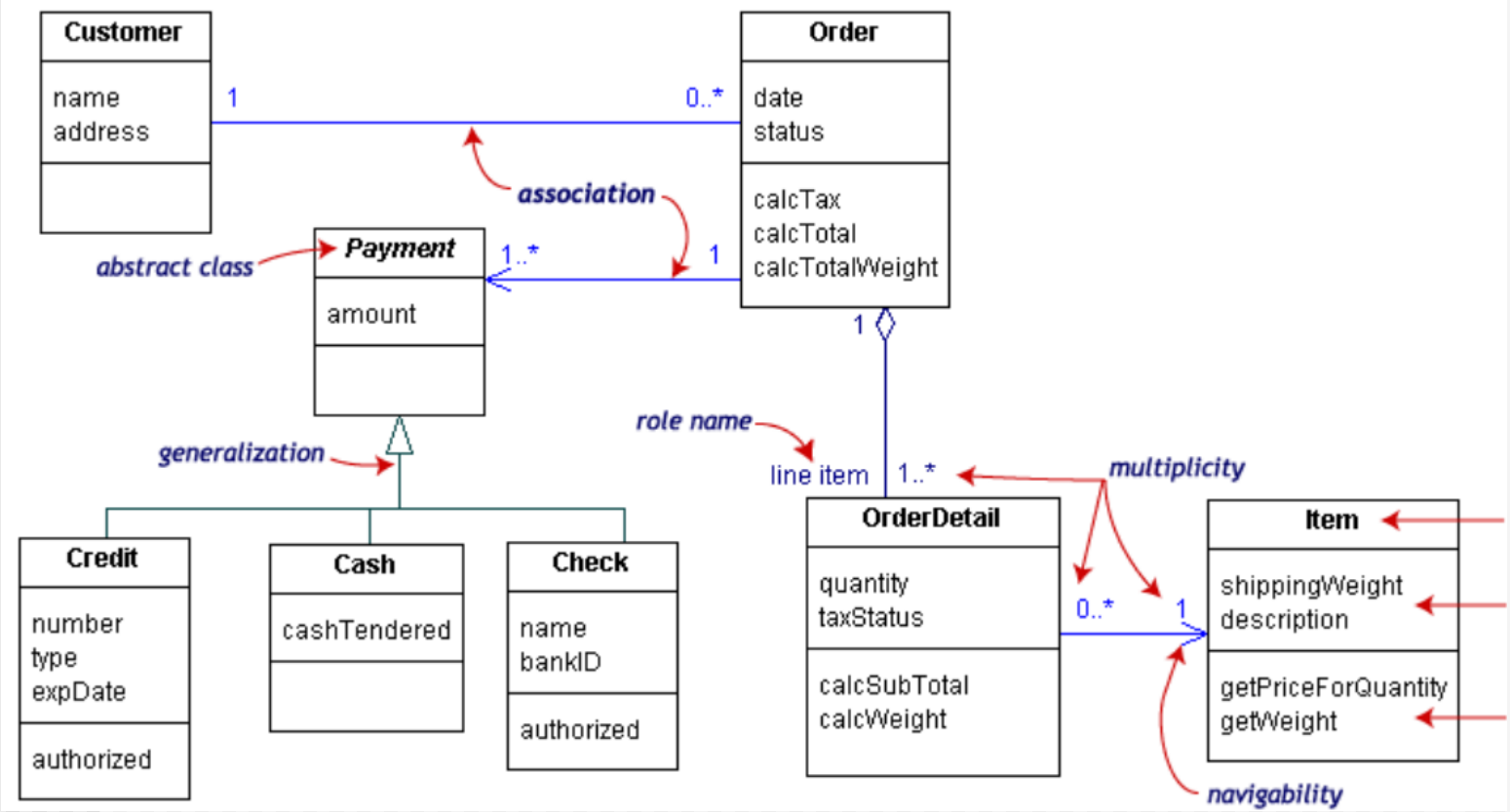
表示类与类、接口与接口之间的继承关系




```
class Animal{}
class Tiger extends Animal{}
public class Test
{
    public void test()
    {
        Animal a=new Tiger();
    }
}
```

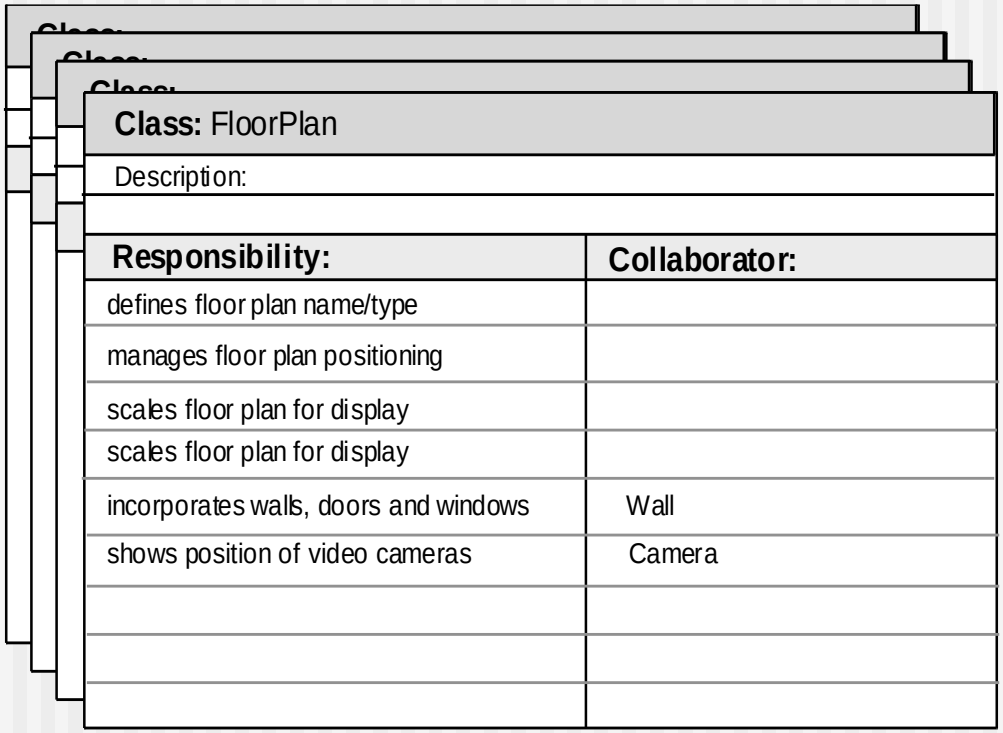
类图实例

某订单管理子系统类图



CRC Models

- Class-responsibility-collaborator (CRC) modeling
- 作用：帮助补充、完善、优化分析类（属性、行为）



Class: FloorPlan

Description:

Responsibility:	Collaborator:
defines floor plan name/type	
manages floor plan positioning	
scales floor plan for display	
scales floor plan for display	
incorporates walls, doors and windows	Wall
shows position of video cameras	Camera

Responsibilities

- 职责是与类相关的属性和操作。
- 简单的说，职责就是“类所知道的或能做的任何事情”,可理解为类的能力。
- 例如，顾客有名字、地址和电话号码，这是顾客知道的东西。顾客要借书和还书，这是顾客要完成的事情。
- 分配职责的指导原则：属性和与之相关的行为应该在同一个类中。

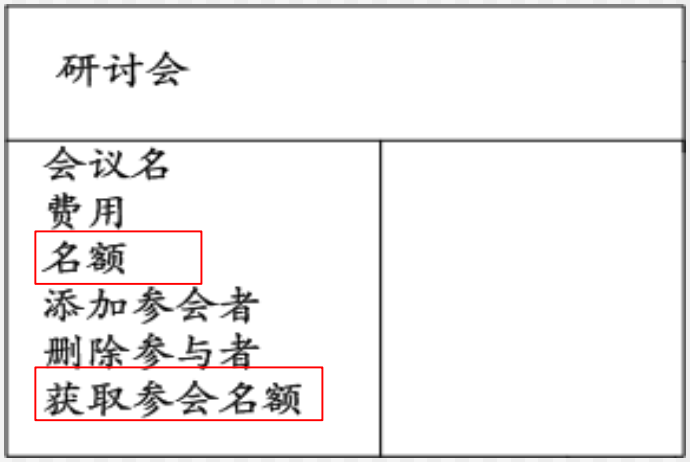
Collaborations

- 类主要通过两种方式实现其职责：
  - 通过自己内部的行为，实现特定的职责
  - 请求其它类的协作

- 如果某个对象请求协作，就会向其它对象发送消息。
- 消息接收者作为服务方，通过自己的服务作为对消息发送者的回应。

- 职责是一种较高层次的抽象，其在设计过程中可能细化为一到多个具体的行为。
  - 如：用户登录用例中，登录控制类的职责之一是“身份验证”。该职责在设计阶段，可以被细化为多个具体方法：getUser()，getRole()，getGroup()，check()

使用CRC模型

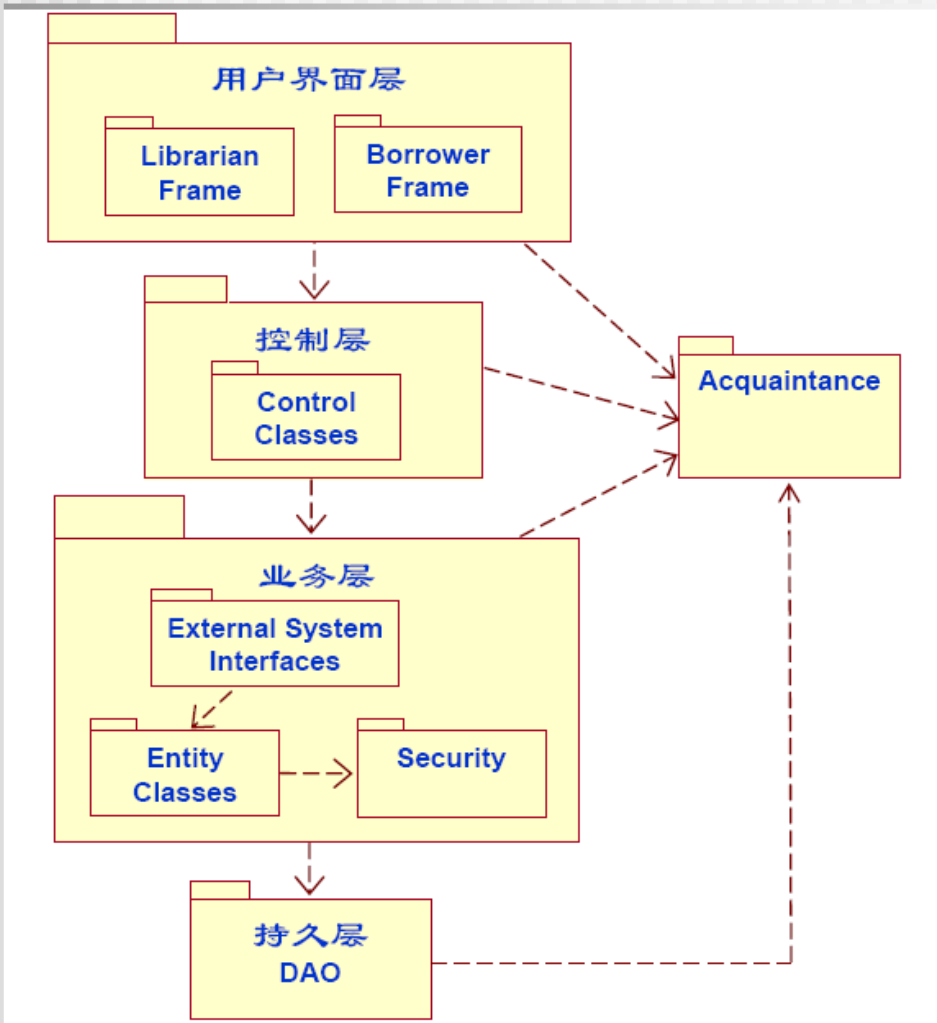


学生在执行“参加研讨会”职责时，需要知道研讨会中是否有空的名额，这就此需要研讨会类的协作（通过名额和获取参会名额）

Packages

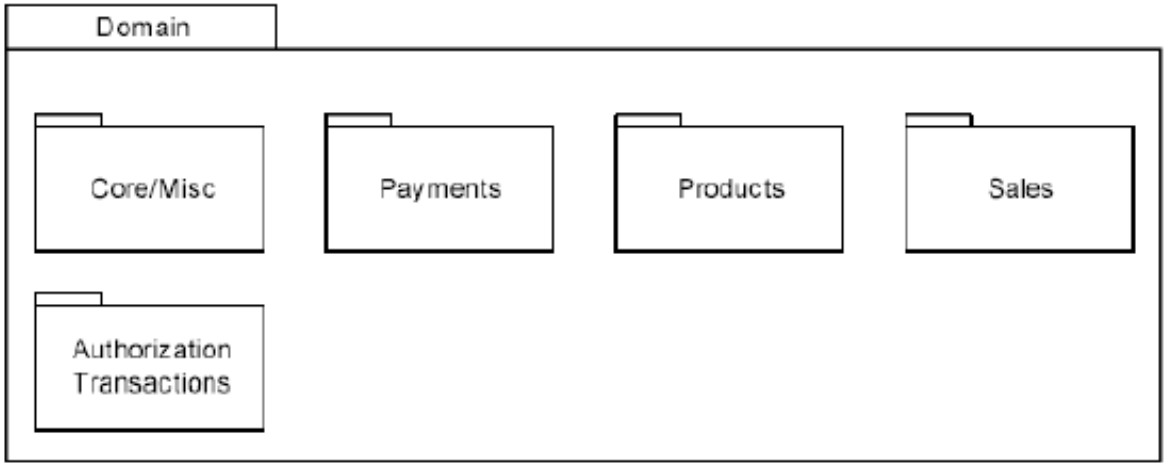
- 对于较大的项目，由于有很多的类和用例，常常会使系统结构看起非常复杂。
- 通过包图，庞大系统之间的关系变得更简单，整个系统的架构一目了然。
- 原则：将概念上或语义上相近的类纳入一个包

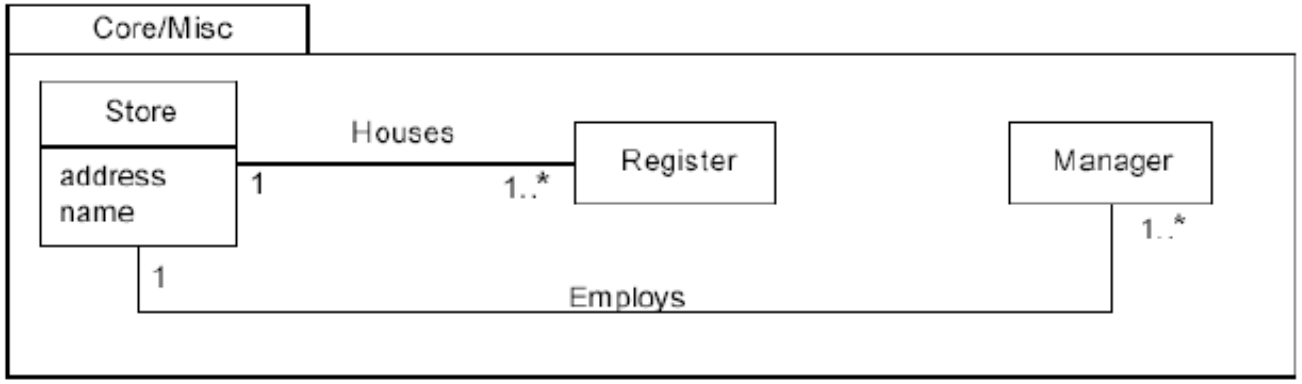
Example:Package



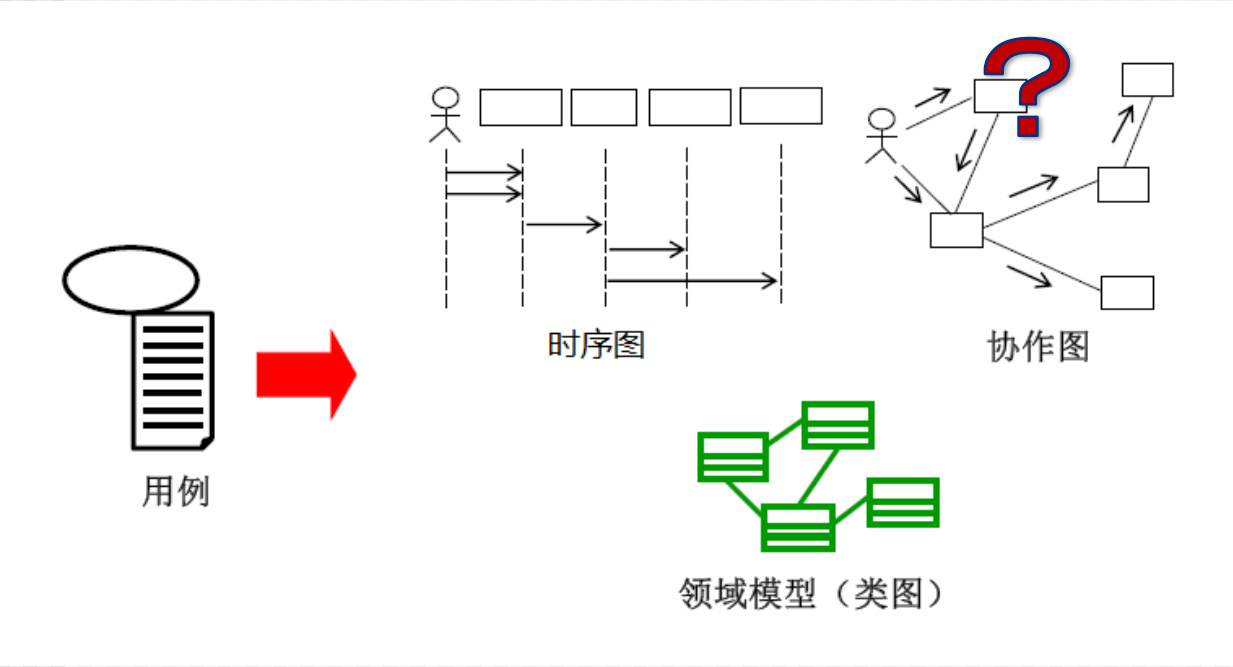
- ✓ 包之间的主要关系是依赖关系。
- ✓ 意味着两个Package内的元素之间存在着一个或多个依赖

Example:Package





OOA分析建模回顾



- 通过用例建模、领域建模（类图），我们已经确定了系统中主要的分析类。
- 但要确定分析类的行为，还要进一步对行为(动态)建模。

第四步：行为建模

- 行为模型显示了软件如何对外部事件或激励做出响应。
- 该模型通常描述一个用例中，系统在接收到特定事件后，所涉及的对象以及这些对象之间的交互（消息传递）过程。
- 行为模型主要包括：
  - Sequence diagram.
  - Collaboration diagram
  - State diagram.

Sequence Diagram

时序图建模步骤

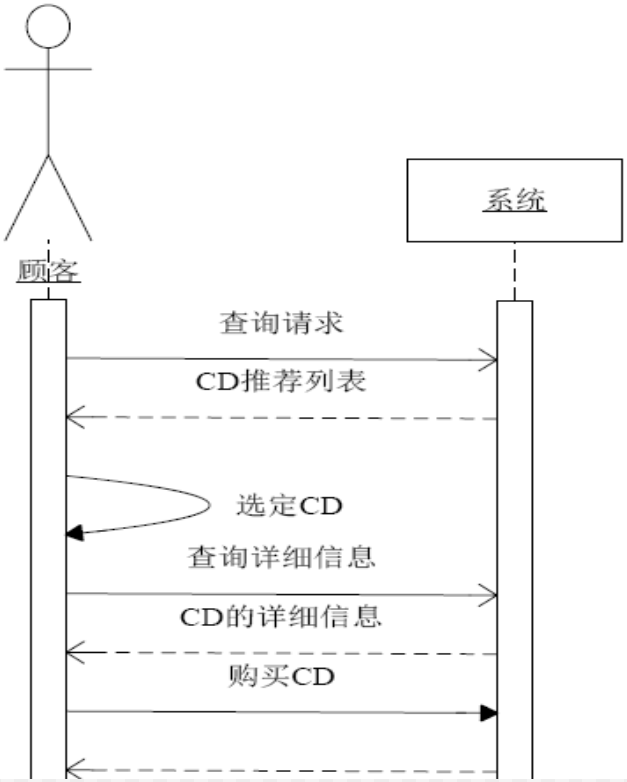
- (1)对每个用例，建立系统时序图（可选）
- (2)建立对象时序图
- (3)从图中确定分析类的行为

系统时序图

- 系统时序图中将系统作为一个黑盒子

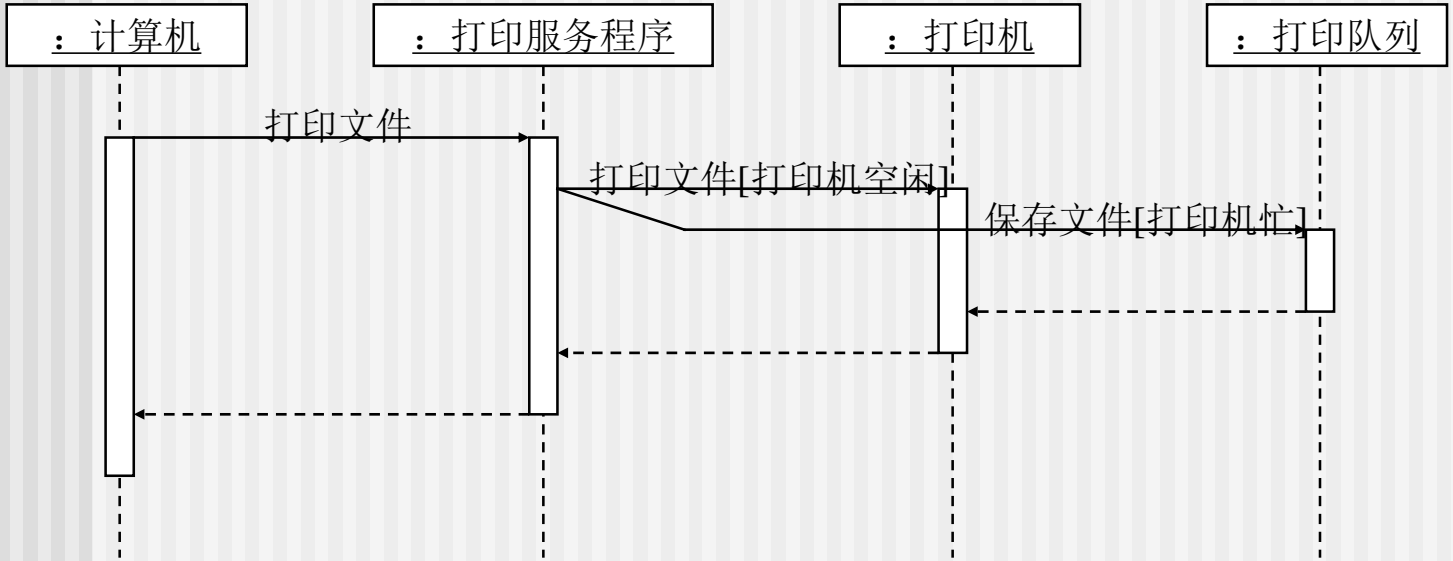
用例描述:

1. 顾客向系统提起查询请求
2. 系统根据请求为顾客提供一个CD的推荐列表
3. 顾客在推荐列表中选定一个CD，然后要求查看更详细的信息
4. 系统为顾客提供选定CD的详细信息
5. 顾客购买选定CD.
6. 顾客离开.



对象时序图

- 依据时间顺序，将系统的行为逐一分配到所识别的分析类中。
- 时序图中，纵轴是时间轴，横轴代表在用例中各对象
- 虚线表示对象存在的生命线，消息用从一个对象的生命线到另一个对象的生命线的箭头表示



绘制对象时序图

- 主要步骤

  按从左到右的顺序

  1、列出启动该用例的参与者；

  2、列出参与者使用的边界类；

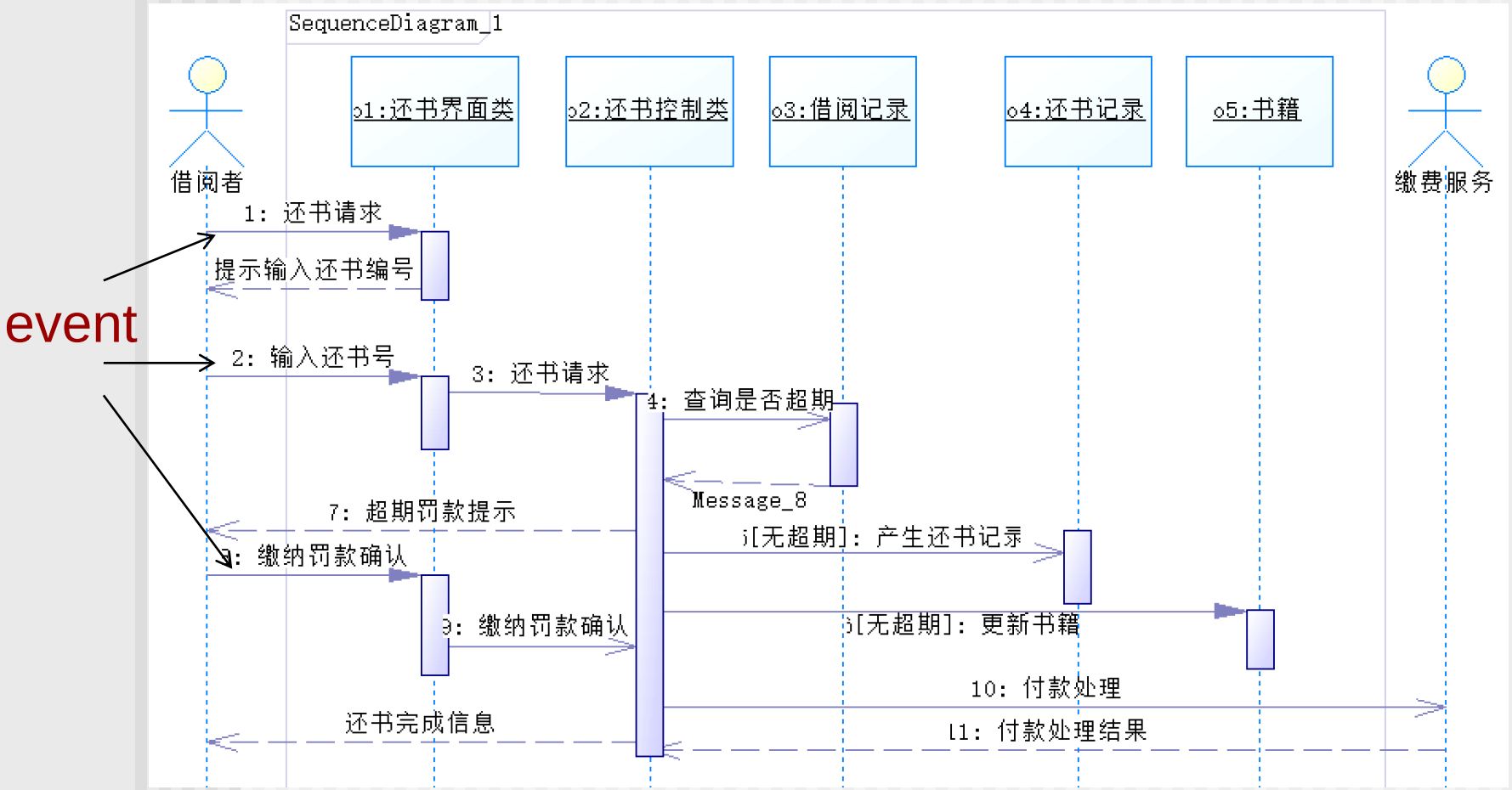
  3、列出管理该用例的控制类；

  4、根据用例的执行流程，按时间顺序列出分析类之间的消息序列。

Example:还书用例

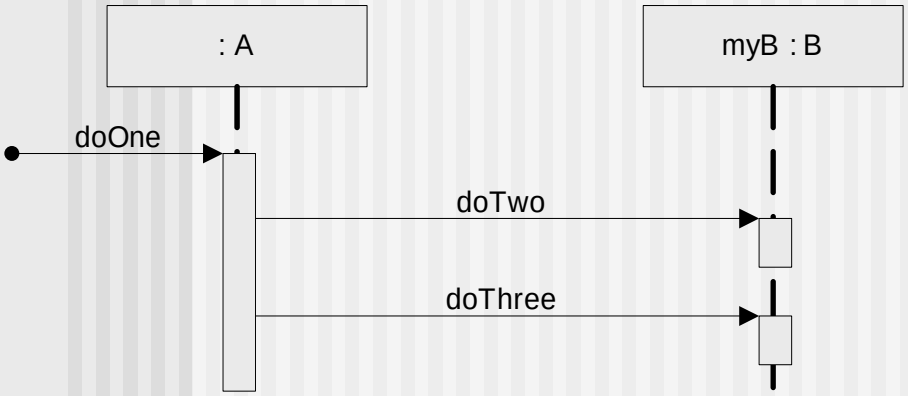
- 借阅者点击还书按钮向系统申请还书
- 系统提示借阅者输入还书号等信息
- 借阅者输入还书信息后，系统查询该书是否已超期。
- 如果未超期，则记录用户还书信息并更新图书可借状态，并返回借阅者成功还书信息。
- 如果已超期，则系统提示超期、罚款金额等信息。
- 借阅者如果认可，并确认缴纳罚款，则系统执行扣款并还书，返回借阅者成功还书信息。

Example:还书用例时序图



为分析类分配职责

- 根据消息传递过程，将职责分配到相应的分析类中。
- 等到设计阶段，职责将被进一步细化，最终对应设计类中的成员方法。




```
public class A {
    private B myB = new B();
    public void doOne() {
        myB.doTwo();
        myB.doThree();
    }
    // …
}
```

消息与职责的关系

- 消息与职责并不完全是一回事。可以理解为消息是一种请求，职责是对该请求的回应或执行。
- 消息可能不对应任何职责，如返回消息。

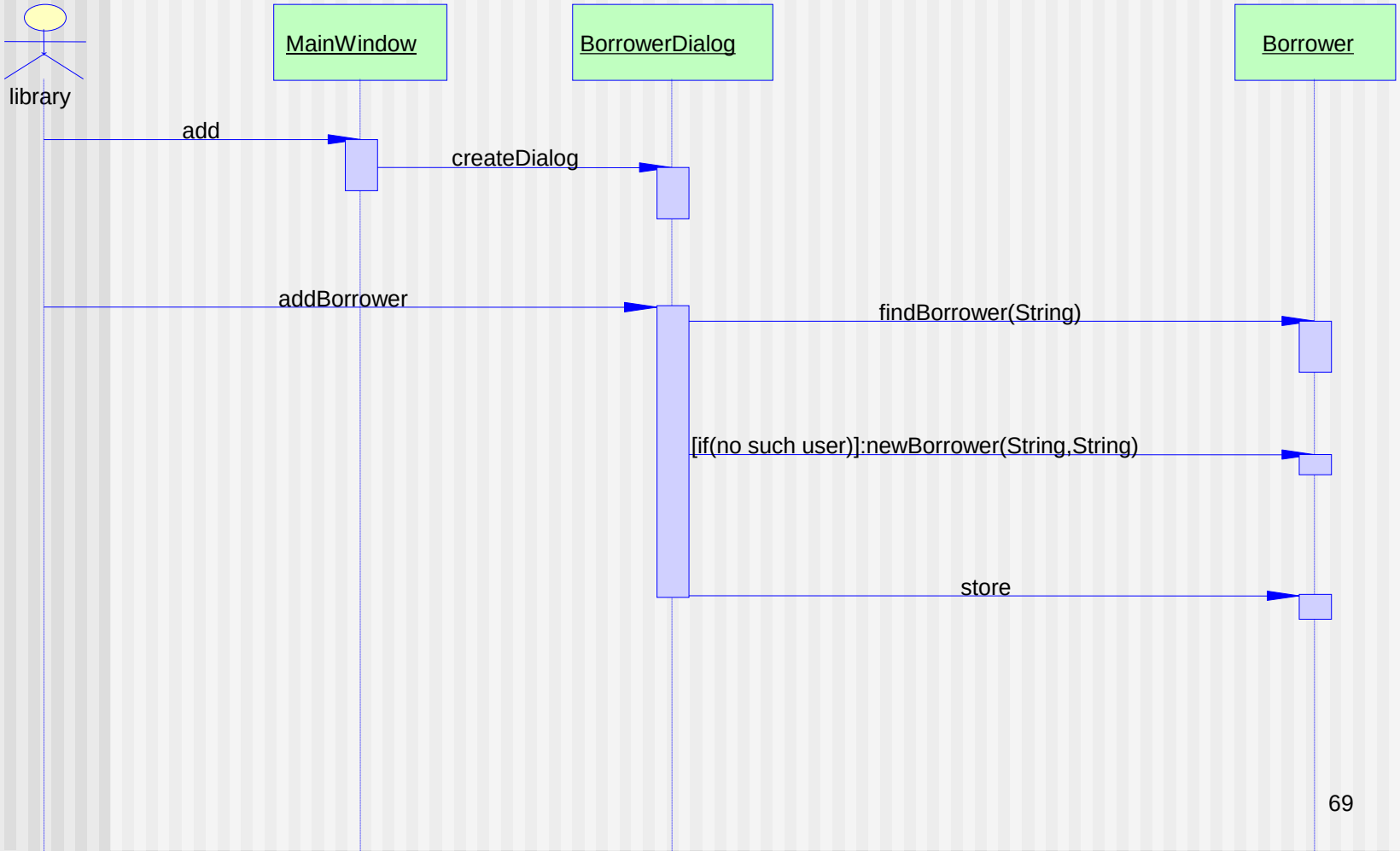
Exercise:时序图

添加借阅者用例：

管理员选择菜单项“添加借阅者”，系统弹出借阅者对话框，管理员输入借阅者信息并提交，系统根据借阅者ID查询数据库，看是否存在该借阅者，如果不存在，创建该借阅者帐户。

画出其对应的时序图。

Exercise:时序图



Exercise:时序图

类Borrower

私有属性

……

公共操作

newBorrower(name: String, address: String, zipCode: String, id: String, telNum: String)

以借阅者名字、地址等信息为参数创建Borrower对象。

findBorrower(id: String) : OID

返回指定ID号的Borrower对象的OID。

getBorrower(oid: OID) : Borrower 返回指定OID的Borrower对象。

addLoan(loan: OID) 添加借阅记录。

getNumLoans() : Integer 返回借阅记录的数目。

getLoan(index: Integer) : Loan 返回指定数组索引号的Loan对象。

……

store(out: DBFile) 将Borrower对象的属性写入数据库文件中。

Exercise:时序图

类BorrowDialog

私有属性

……

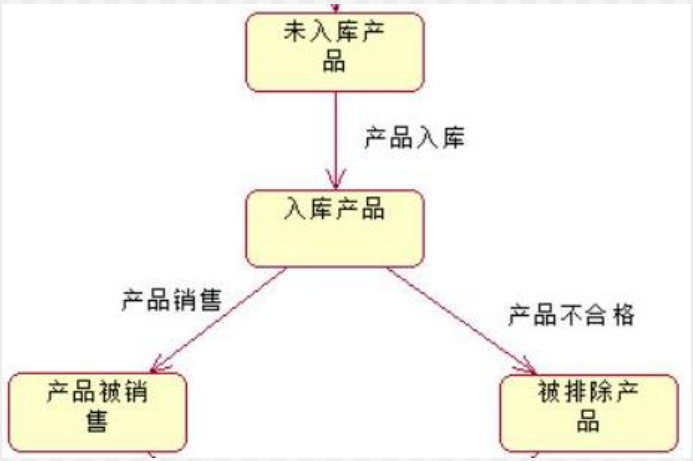
公共操作

createDialog()　　创建用于填写借阅信息的对话框

addBorrower()　　当对话框被提交时，该方法被调用

状态图（State Diagram）

- 状态图描述了一个对象生命周期内所经历的状态序列，以及引起状态转移的外部事件。

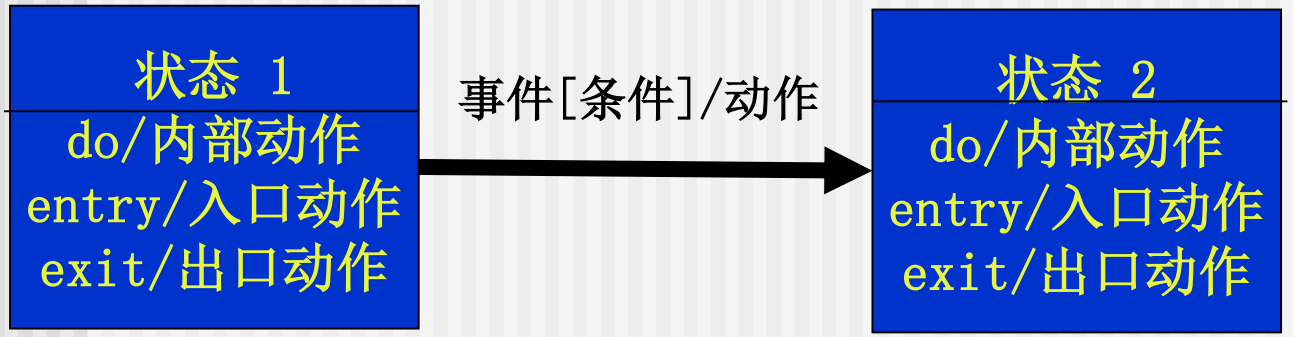


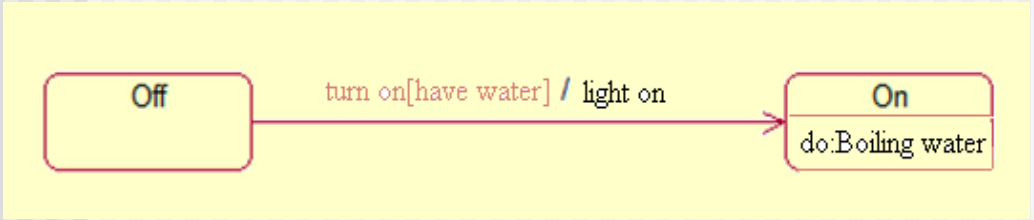
- 状态图的主要作用

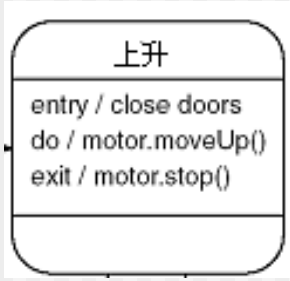
描述多个状态之间的转换顺序，即事件的执行顺序，避免非法的事件序列。

如: 必须先付款进入“已付款”状态，才能执行发货进入“已发货”状态。

状态图基本元素


状态 1
do/内部动作
entry/入口动作
exit/出口动作
事件[条件]/动作
状态 2
do/内部动作
entry/入口动作
exit/出口动作

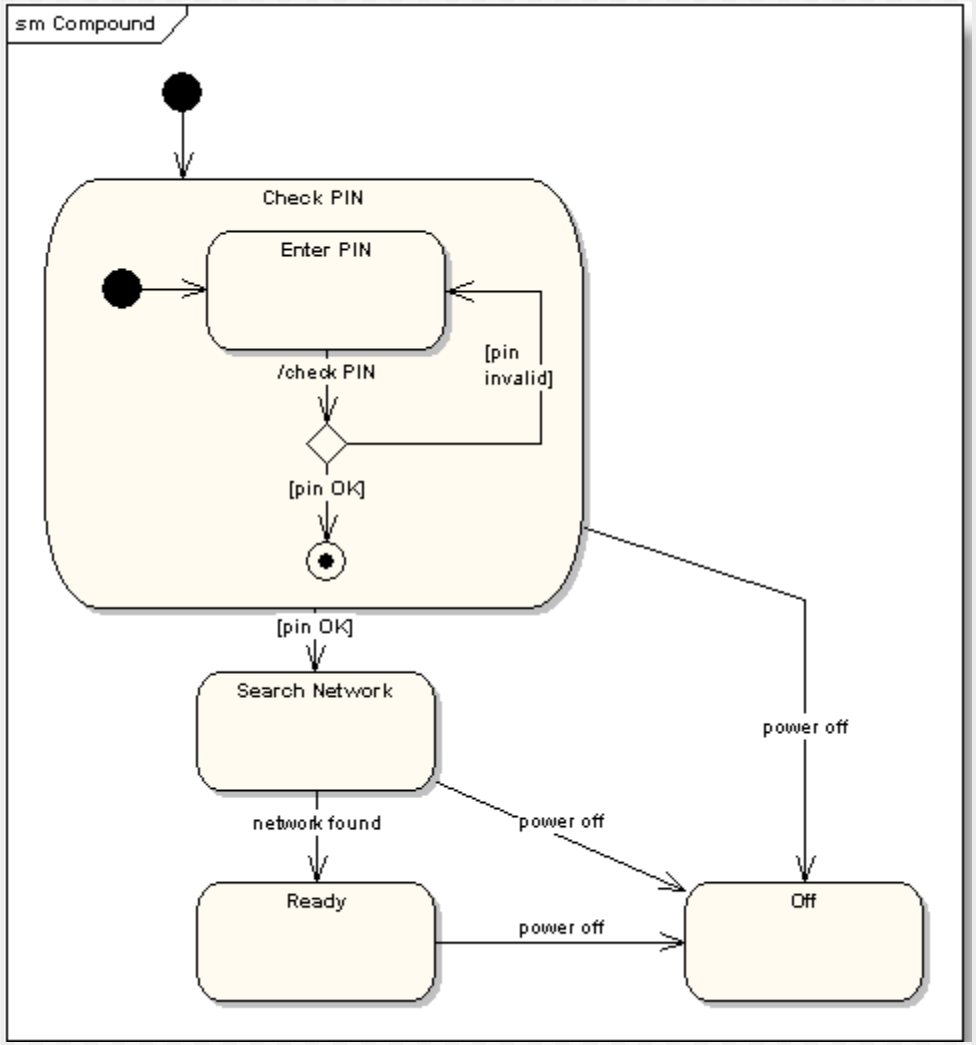

Off
turn on[have water] / light on
On
do:Boiling water


上升
entry / close doors
do / motor.moveUp()
exit / motor.stop()

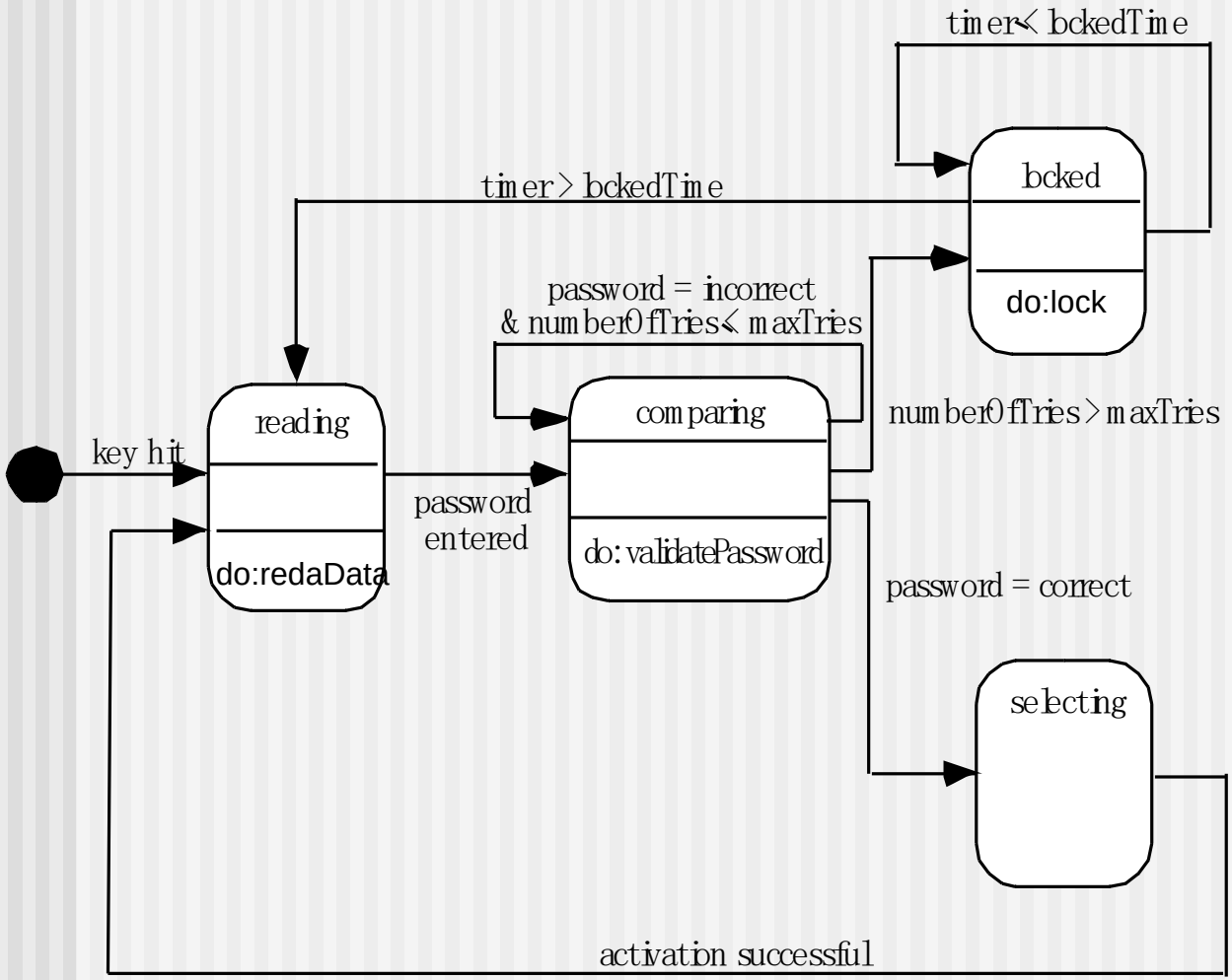
状态图基本元素

几种特殊状态

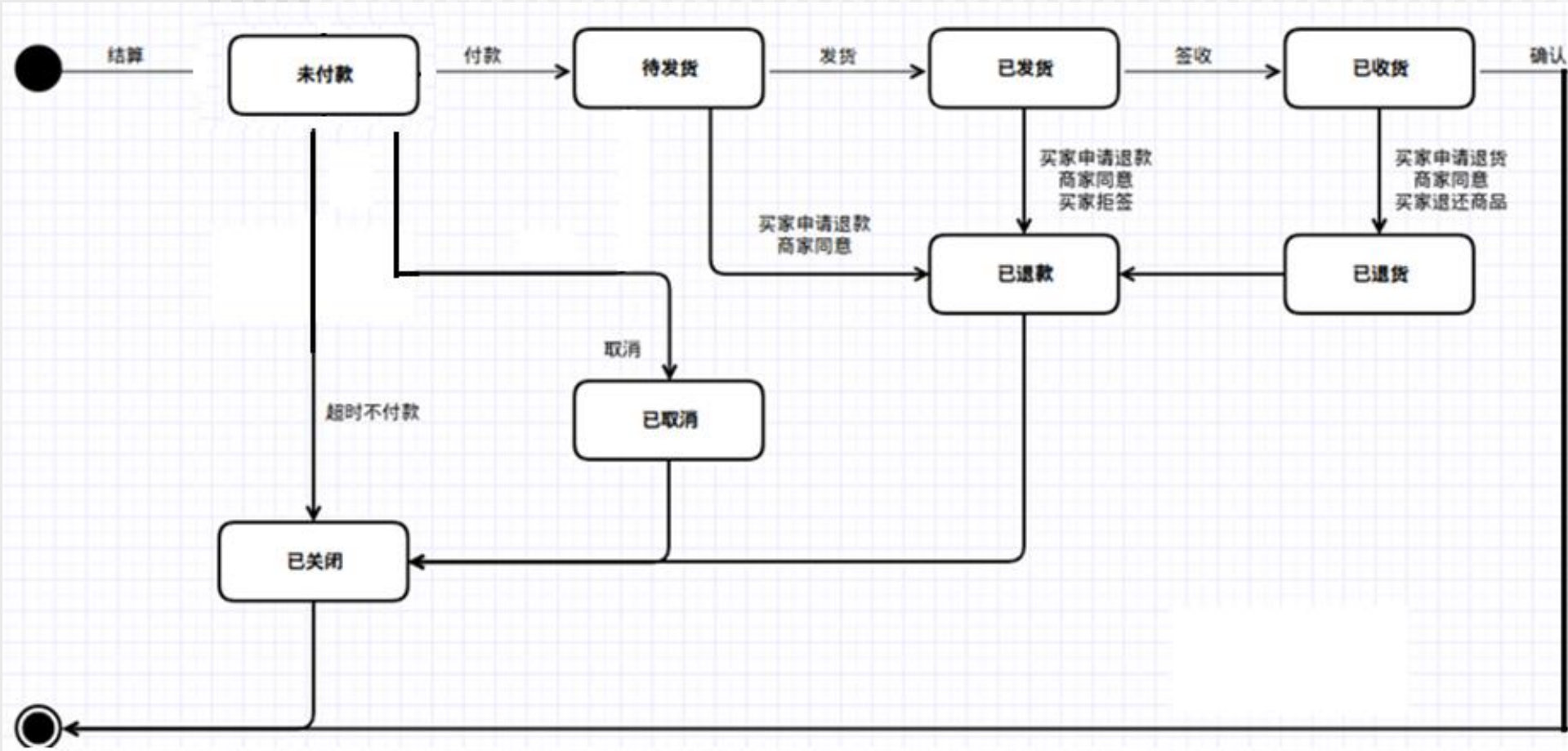
- 一个起始状态
- 多个终止状态
- 组合状态



State Diagram for ControlPanel



订单状态图


结算
未付款
付款
待发货
发货
已发货
签收
已收货
确认
买家申请退款
商家同意
买家拒签
买家申请退货
商家同意
买家退还商品
买家申请退款
商家同意
已退款
已退货
取消
超时不付款
已取消
已关闭

行为模型比较

- 状态图 vs 时序图

  (1)状态图针对单个类，跨多个用例。时序图针对的是一个用例。

  (2)状态图中的行为不仅与事件有关，还与对象所处状态有关。

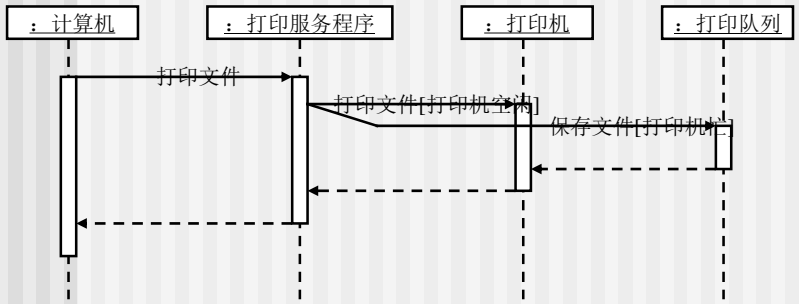
例如：电梯控制系统中，事件“向下按键”，有时会触发电梯的“停止开门”行为，有时却执行“向下运行”行为。为什么？

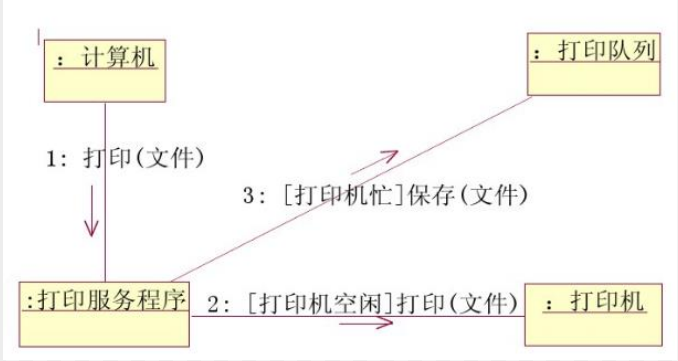
行为模型比较

- 时序图 vs 协作图

(1)时序图可等价的转化为协作图

(2)协作图更侧重表达对象间的关联关系，而非消息传递的时间先后。





Summary of OOA Modeling

1. 用例建模。寻找actor和用例，对用例场景进行描述。
2. 静态建模。从用例场景中寻找分析类，建立初步的分析类图。
3. 动态建模。用时序图来描述每个用例的交互过程，以此发现每个类对应的职责。

4. 建立包图，对分析类、用例等进行分包组织。（可选）
5. 如果系统需要用到数据库，建立数据模型（可选）
6. 如果类的生命周期比较复杂，则考虑建立状态图（可选）