

软件工程原理与实践

by PeterTheSparrow

本文档整理上海交通大学软件学院《软件工程原理与实践》课程的相关知识点，用于期末考试复习。本文档仅供学习使用；如若侵犯任何个人或者集体的利益，请联系删除。

- 软件工程原理与实践
 - chapter 0. 课程大纲
 - chapter 1. 往年期末试卷知识点选讲
 - > 1.1 类图
 - > 1.1.1 类
 - > 1.1.2 类和类的关系
 - > 1.1.2.1 泛化 (Generalization) / 继承 (Inheritance)
 - > 1.1.2.2 关联关系 (Association)
 - > 1.1.2.3 聚合 (Aggregation)
 - > 1.1.2.4 组合 (Composition)
 - > 1.1.2.5 依赖 (Dependency)
 - > 1.2 时序图
 - > 1.3 状态图
 - > 1.4 用例图 (use case)
 - > 1.4.1 用例图概述
 - > 1.4.2 识别actor和use case, 画use-case图
 - > 1.4.3 编写use-case spec.
 - > 1.4.4 优化use-case图的结构
 - > 1.5 黑盒白盒测试
 - > 1.6.1 白盒测试
 - > 1.6.2 黑盒测试
 - > 1.6 项目设计的软件过程
 - chapter 2. 上课练习题以及考试题选
 - > 2.1 考前习题课内容
 - > 2.1.1 黑盒测试1
 - > 2.1.2 黑盒测试2
 - > 2.1.3 概念题
 - > 2.1.4 软件设计过程
 - > 2.1.5 用例建模

- >2.1.6 状态图
- >2.1.7 设计模式

chapter 0. 课程大纲

1. 引论
2. 软件过程
3. 软件工程模型和方法
- 4. 软件需求【重点】**
5. 软件架构设计
6. 软件详细设计
7. 用户界面设计
8. 编码、版本管理
9. 软件测试
10. 软件运维
11. 软件项目管理

chapter 1. 往年期末试卷知识点选讲

> 1.1 类图

课本P120

以下内容很多摘自[这篇文章](#)

概述

类图：

- 类
- 类和类的关系

> 1.1.1 类

属性或者方法名称前加的加号和减号是什么意思呢？它们表示了这个属性或方法的可见性，UML类图中表示可见性的符号有三种：

- + : 表示public
- : 表示private
- # : 表示protected (friendly也归入这类)

>1.1.2 类和类的关系

类和类之间的6种关系：

泛化 (Generalization) , 实现 (Realization) , 关联 (Association) , 聚合 (Aggregation) , 组合 (Composition) , 依赖 (Dependency) 。

各种关系的强弱顺序： 泛化 = 实现 > 组合 > 聚合 > 关联 > 依赖。

类之间6种关系汇总：

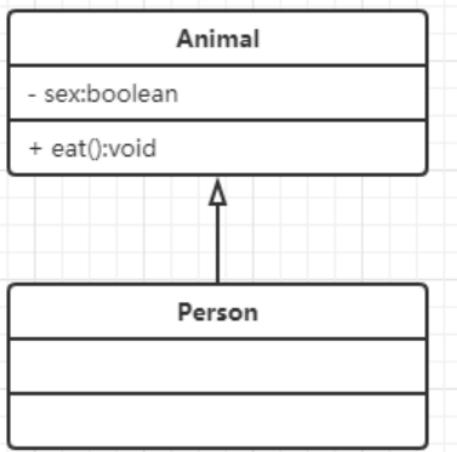
关系	关系如何使用UML图表达
泛化	使用带空心三角形的实线表示泛化(继承)关系
实现	使用带空心三角形的虚线表示实现关系
关联	单向关联使用一个带箭头的实线表示，双向关联使用一个不带箭头的实线表示，自关联使用一个带有箭头且指向自身的实线表示
聚合	聚合关系用带空心菱形和箭头的实线表示
组合	组合关系用一个带实心菱形和箭头的实线表示
依赖	依赖关系用一条带有箭头的虚线表示

>1.1.2.1 泛化 (Generalization) / 继承 (Inheritance)

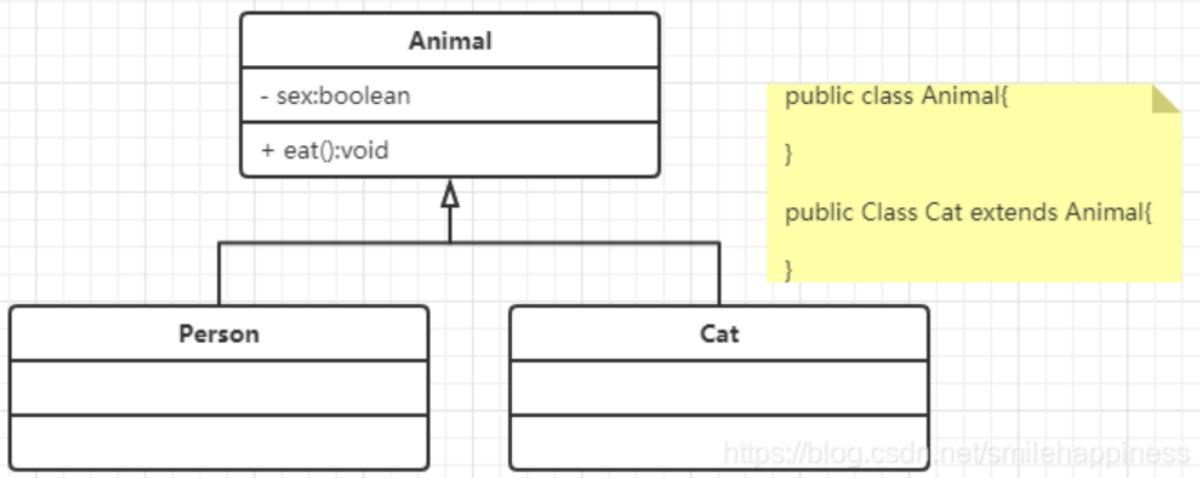
泛化关系，指的就是：类和类、接口与接口之间的继承关系。继承关系可以用 IS-A 表示。

举例：

人是一种高级动物，就可以表示为：Person是一个 (IS-A) 高级动物，这就是一种继承关系。



继承使用的是 extends 关键字，在UML类图中， 使用带空心三角形的实线表示泛化(继承)关系，如下图所示中，Person类与Cat类都继承了Animal类。Person和Cat都会继承到Animal的公有方法eat()方法。



>1.1.2.2 关联关系 (Association)

关联关系其中包括两种特殊的关联关系：

聚合 (Aggregation) 和组合 (Composition)

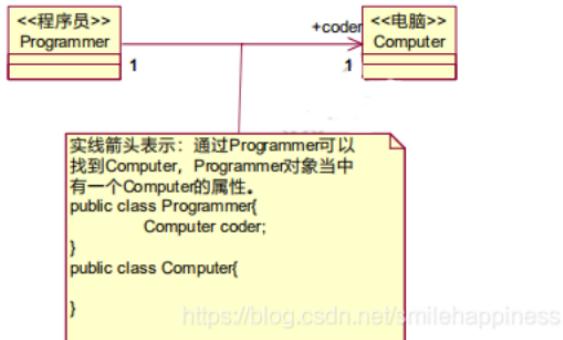
1.3 关联关系

关联关系指的是，一个类与另一个类之间有某种关联。这种关联关系可以使用 **HAS-A** 表示。

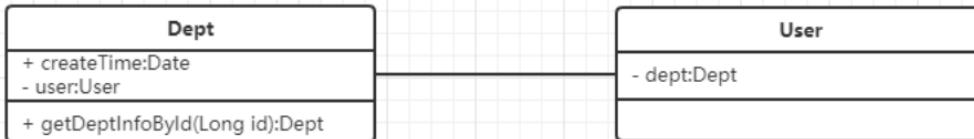
关联关系如果进行详细的划分，又可进一步分为 **单向关联**、**双向关联** 和 **自关联**。

- 单向关联

我们可以看到，在UML类图中，**单向关联**使用一个带箭头的实线表示。



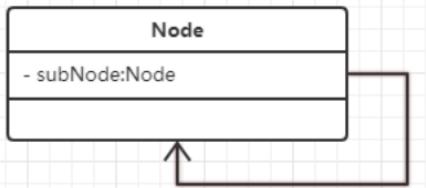
- 双向关联



从上图中我们很容易看出，所谓的双向关联就是双方各自持有对方类型的成员变量。在UML类图中，**双向关联**使用一个不带箭头的实线表示。上图中在Dept类中维护一个User类型的成员变量，在User类中，也维护了一个Dept类型的成员变量，这种就属于双向关联。

- 自关联

自关联在UML类图中，**自关联**使用一个带有箭头且指向自身的实线表示。



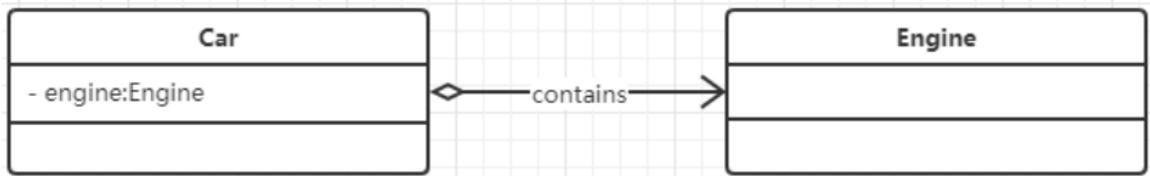
上图的意思就是Node类包含类型为Node的成员变量，也就是“自己包含自己”。

>1.1.2.3 聚合 (Aggregation)

聚合关系描述的是整体和部分的关系，是一种比较特殊的关联关系。在聚合关系中，整体的生命周期，不会决定部分的生命周期。

比如：

汽车和发动机引擎之间的关系、老师和学生之间的关系，就是整体和部分的关系。



上图中的Car类与Engine类就是聚合关系（Car类中包含一个Engine类型的成员变量）。由上图我们可以看到，在UML类图中，聚合关系用带空心菱形和箭头的实线表示。聚合关系强调是“整体”包含“部分”，但是“部分”可以脱离“整体”而单独存在。比如上图中汽车包含了发动机引擎，汽车脱离了发动机是可以单独存在的，而发动机引擎脱离了汽车也能单独存在。

>1.1.2.4 组合 (Composition)

组合关系也可以看作是一种特殊的聚合关系，是一种比聚合关系还要强的关系。整体的生命周期决定部分的生命周期，部分是依附在整体上面的，有时候也有人称为合成关系。

组合关系与聚合关系最大的不同在于：组合关系这里的“部分”脱离了“整体”是无法存活的，部分就不复存在。

比如下图人体和细胞的关系：



细胞组成人的一部分，显然，细胞脱离了人，是不能单独存在的。在UML类图中，组合关系用一个带实心菱形和箭头的实线表示。

>1.1.2.5 依赖 (Dependency)

依赖关系是比关联关系弱的一种关系，是所有关系中最弱的一种，在java语言中体现为返回值，参数，局部变量和静态方法调用。



从上图我们可以看到，汽车驾驶员CarDriver的drive方法只有传入了一个Car对象才能发挥作用，因此我们说CarDriver类依赖于Car类。在UML类图中，依赖关系用一条带有箭头的虚线表示。

>1.2 时序图

课本P124

>1.3 状态图

略

>1.4 用例图 (use case)

参考资料

- 课本P128
- [链接1](#)

>1.4.1 用例图概述

由参与者（Actor）、用例（Use Case）以及它们之间的关系构成的用于描述系统功能的动态视图称为用例图。

其中用例和参与者之间的对应关系又叫做通讯关联（Communication Association）。

用例图的作用

用例图是需求分析中的产物，主要作用是描述参与者与用例之间的关系，帮助开发人员可视化地了解系统的功能。借助于用例图，系统用户、系统分析人员、系统设计人员、领域专家能够以可视化的方式对问题进行探讨，减少了大量交流上的障碍，便于对问题达成共识。

>1.4.2 识别actor和use case，画use-case图

- Actor：参与者，可以是人，也可以是其他系统。例如：学生选课，学生就是Actor；我选课系统和jacount进行交互，甲亢就是Actor。
- Use Case：用例。学生选课，选课这件事情就是use case。

如何识别Actor

- 谁需要在系统的帮助下完成自己的任务?
- 需要谁去执行系统的核心功能?
- 需要谁去完成系统的管理和维护?
- 系统是否需要和外界的硬件或软件系统进行交互?

如何识别Use Case

- 每个actor的目标和需求是什么?
 - actor希望系统提供什么功能?
 - actor 将创建、存取、修改和删除数据吗?
 - actor是否要告诉系统外界的事件?
 - actor 需要被告知系统中的发生事件吗?

- 上面一张图的第三条, 传感器 (Actor) 要告诉系统窗户是不是开着的; 所以窗户开关信息告知就是一个use case。
- 第四条, 警报器 (actor) 需要系统告诉他火灾是否发生了, 所以火灾信息告知就是一个use case。

避免功能分解

现象

- use case 太小
- use cases 太多
- use case 没有价值回报
- 命名以低层次的操作
 - “Operation” + “object”
 - “Function” + “data”
 - Example: “Insert Card”
- 很难理解整个模型

纠正措施

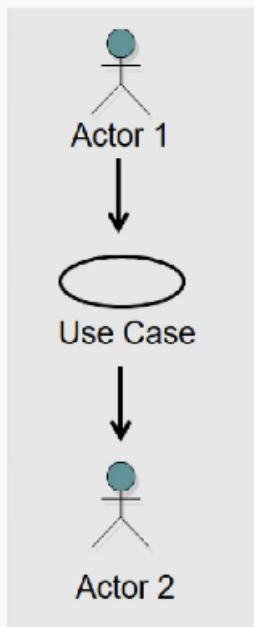
- 寻找更大的上下文
 - “为什么你要构建本系统?”
- 让自己站在用户的角度
 - “用户想得到什么”
 - “这个 use case 能满足谁的要求?”
 - “这个 use case 能增值什么?”
 - “这个 use case 背后有什么故事?”

一个用例定义了和 actor 之间的一次完整对话。

62

- 不能太小，比如增删改查，可以统一成为“信息管理”。
- use case 取名一定是一个动宾结构，使得容易理解。

通信-关联 Communicates-Association



- Actor 和 use case 间的通信渠道
- 用一条线表示
- 箭头表示谁启动通信

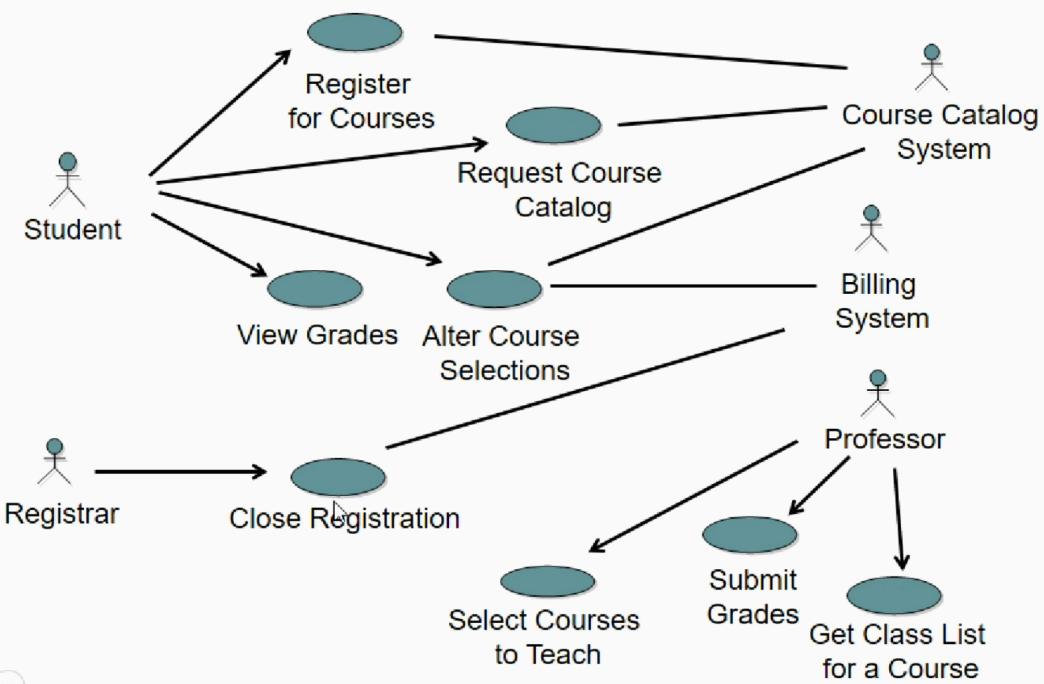
- 画箭头，谁发起的通信

上面这些都掌握以后，我们来看：

63

- 总的一个例子：

Course Registration System的用例图



64

>1.4.3 编写use-case spec.

Use Case Specification（用例规约）是一种软件需求规格说明书，它包含了一个或多个用例的详细描述、输入和输出条件、前置条件和后置条件等信息。简单来说，Use Case Specification定义了系统中一个或多个用例的执行过程和预期结果。

在Use Case Specification中，每个用例都被视为一个完整的系统功能，由以下几个部分组成：

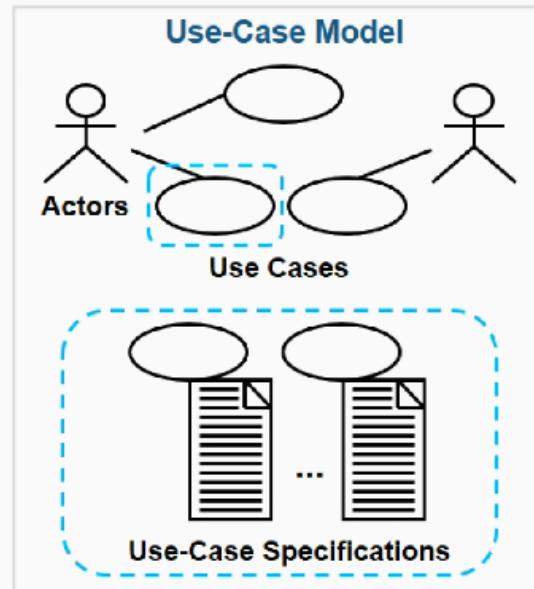
- 标题和简述：对用例的简要描述；
- 用例ID：用于唯一标识该用例的编号；
- 参与者（Actors）：用例涉及的角色或者人员；
- 前置条件（Pre-conditions）：执行该用例需要满足的条件；
- 关键流程（Main Flow）：用例中的主要步骤，以及输入和输出条件；
- 扩展流程（Alternative Flows）：当关键流程无法执行时，用例所执行的其他步骤；
- 后置条件（Post-conditions）：用例执行后所达到的状态。

通过Use Case Specification，可以帮助开发人员更好地理解系统中各个用例的执行过程和预期结果，同时也可以帮助用户或者测试人员更好地了解系统的功能和使用方法。

总之，Use Case Specification是一种重要的软件需求规格说明书，能够帮助开发团队和用户更好地理解系统中各个用例的执行过程和预期结果，从而提高软件开发的效率和质量。

Use-Case Specifications

- Name
- Brief description
- Flow of Events
- Relationships
- Activity diagrams
- Use-Case diagrams
- Special requirements
- Pre-conditions
- Post-conditions
- Other diagrams

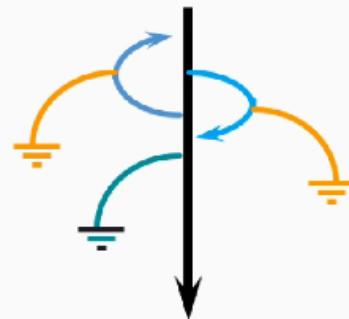


66

- 最重要的还是**简要描述+事件流**

事件流 (基本流和备选流)

- 一个基本流
 - Happy day scenario
 - Successful scenario from start to finish
- 多个备选流
 - Regular variant
 - Odd cases
 - Exceptional (error) flows



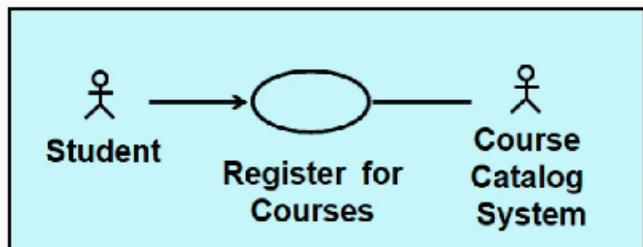
67

- 基本流：比如选课成功；
- 备选流：选课，这门课被抢完了（偶尔发生的情况，概率很小）；系统崩了……

新的概念：Scenario 场景

Scenario是user-case的实例

Scenario 是Use-Case 的实例



Scenario 1

Log on to system
Approve log on
Enter subject in search
Get course list
Display course list
Select courses
Confirm availability
Display final schedule

Scenario 2

Log on to system
Approve log on
Enter subject in search
Invalid subject
Re-enter subject
Get course list
Display course list
Select courses
Confirm availability
Display final schedule

细化基本事件流

Get Quote

Basic Flow

1. Customer Logs On

The use case starts when the Trading Customer logs on. The system validates the customer id and password. ...

2. Customer Selects “Get Quote” Function

The Trading Customer chooses to “Get Quote.” The system displays the list of securities on which it has quotes.

3. Customer Gets Quote

The Trading Customer selects from the list of securities or enters the trading symbol for a security. The system sends the trading symbol to the Quote System, and receives the Quote System Response.

4. Customer Gets Other Quotes

If the Trading Customer wishes to get additional quotes, the use case resumes at Step 3.

5. Customer Logs Off

The Trading Customer logs off the system. The use case ends.

将流分成几步

每步标识编号
和名字

描述每步
(1-3 句)

每步是一个来
回的事件

细化通用备选流

Identify Customer

Alternative Flows

A1. Unidentified Trading Customer

In Step 1 of the Basic Flow, if the system determines that the customer password is not valid, the system displays a "Sorry, not a valid customer...." message. The use case ends.

A2. Wrong Password

In Step 1 of the Basic Flow, if the system determines that the customer password id is not valid, the system displays a "Sorry, not...."

A3. Suspect Theft

In step 1 of the Basic Flow, if the customer id is on the system's list of stolen identification, the system displays a "Sorry, not ..." message.

The system also records the date, time. And computer address...

A4. Quit

The RU e-st System allows the Trading Customer to quit at any time during the use case. The use case ends.

A5. No Reply From User

At any time during the use case, if the system asks for input from the Trading Customer....

在任何步骤
都可能发生

S J T U

71

>1.4.4 优化use-case图的结构

优化的时候，实际上就是新增了：

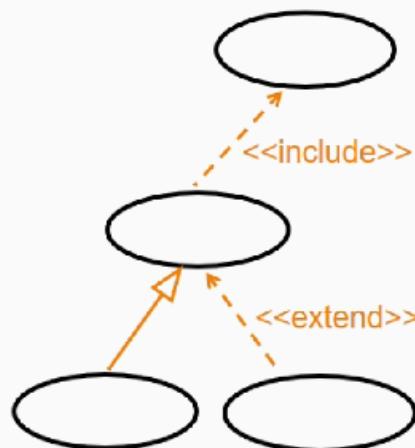
- actor和actor之间的关系
 - use case和use case之间的关系
- (因为我们之前研究的都是actor和use case之间的关系)

Use Case间的关系

□ Include 包含

□ Extend 扩展

□ Generalization 泛化

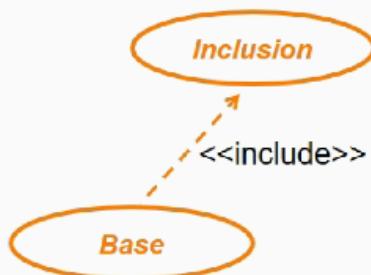


81

什么是Include 关系?

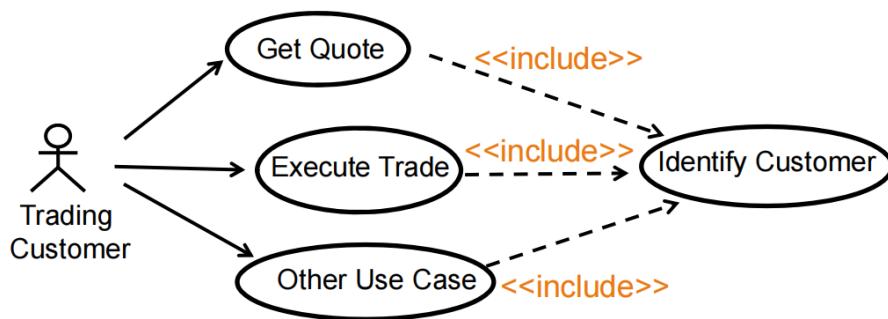
□ 基本用例包含某个包含用例

□ 包含用例定义的行为将显式插入基本用例



82

Include 关系示例



Get Quote Use Case

1. **Include "Identify Customer"** to verify customer's identity
2. Display options. Customer selects "Get Quote"
3. ...

Identify Customer Use Case

1. Log on
 2. Validate logon
 3. Enter password
 4. Verify password
- A1: Not valid logon
A2: Wrong password
A3: ...

S J T U

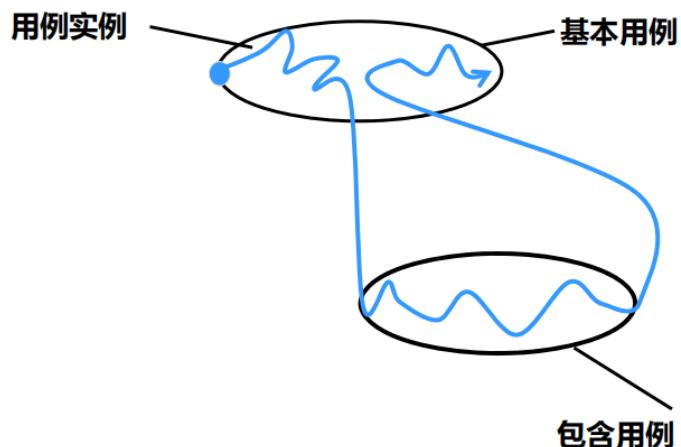
83

这张扭得和方便面一样的图是什么意思呢：

- 执行base case的时候，会执行include的use case（执行完了再回来）

执行Include

- 到达插入点时执行包含用例

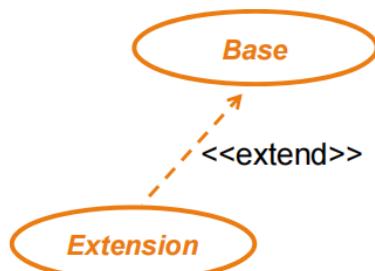


S J T U

84

什么是Extend关系?

- 从基本用例到扩展用例的联接
 - 将扩展用例的行为插入基本用例
 - 只有当扩展条件为真是才进行插入
 - 在一个命名的扩展点插入基本用例

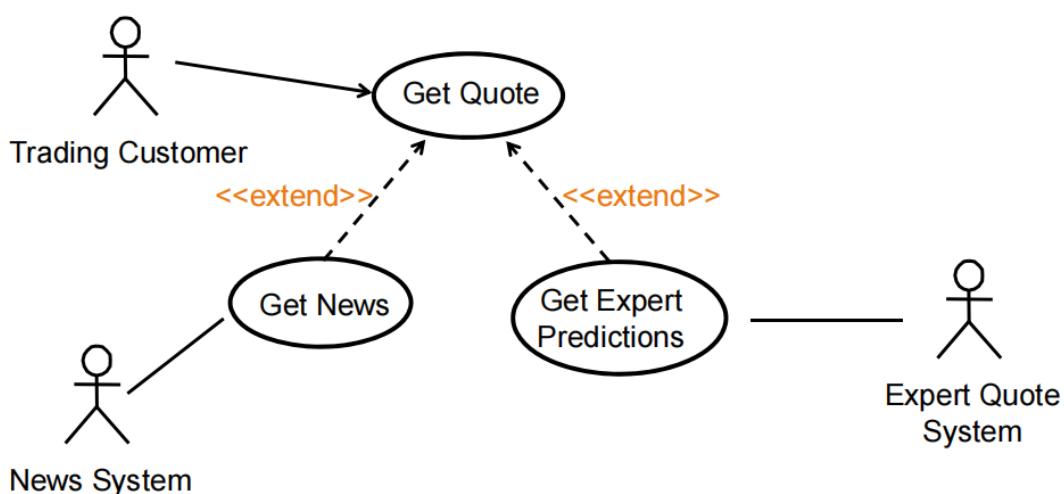


S J T U

85

- 例如：炒股软件中，看报价单，可能会去看一下新闻；又可能看看专家的预测。**include是一定去做，extend是可能去做。**

Extend 关系示例



S J T U

86

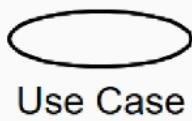
Actor 和 Use Case



Actor

和系统交互的系统外的某些人或某些东西：

- 最终用户
- 外界软件系统
- 外界硬件设备

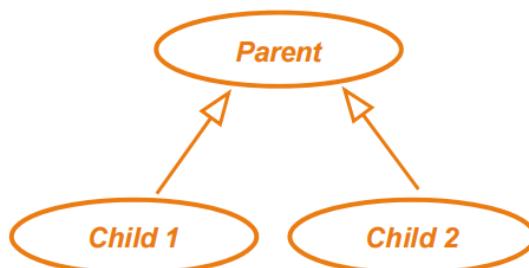


Use case

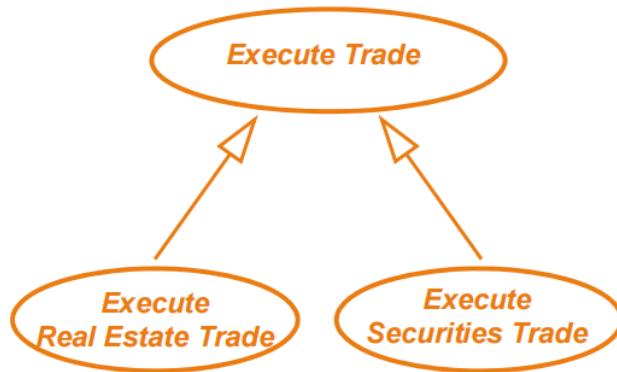
Actor想使用系统去做的事

什么是 Use-Case Generalization?

- 从子用例到父用例的关系
 - 在父用例中描述通用的共享的行为
 - 在子用例中描述特殊的行为
 - 共享共同的目标

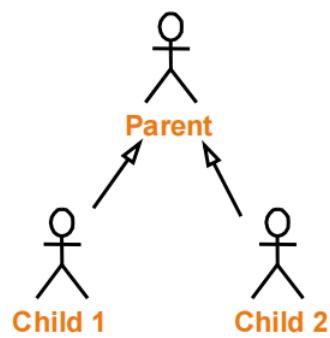


Generalization示例

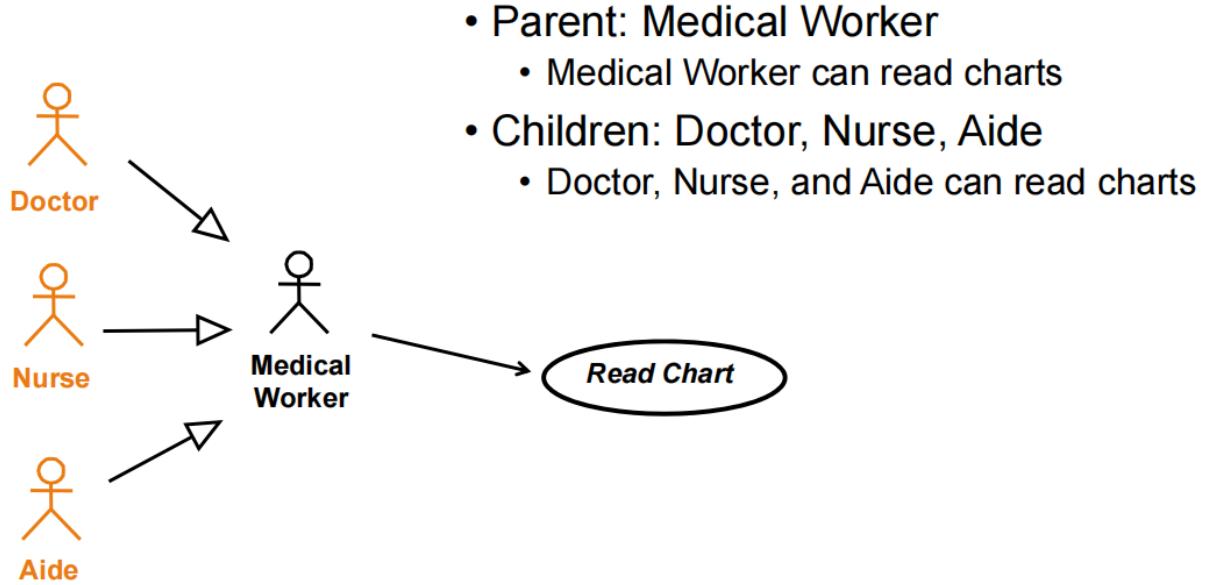


什么是Actor Generalization?

- Actors 可能有公共的属性
- 多个actors 和Use-Case交互时可能有公共的角色或目的



Actor Generalization示例



>1.5 黑盒白盒测试

黑盒测试和白盒测试是软件测试中两种不同的方法。

黑盒测试是一种测试方法，其中测试者不需要了解被测试系统的内部工作原理。测试者只需以用户的角度进行测试，并检查系统是否按照规格说明或其他要求正确运行。因此，黑盒测试通常被认为是从外部对系统进行测试的一种方法。

白盒测试是另一种测试方法，在此测试者需要具有对被测试系统的内部结构和代码的了解。测试者可以查看源代码并进行调试，以确保系统的每个部分都按照预期方式运行。因此，白盒测试通常被认为是从内部对系统进行测试的一种方法。

总之，黑盒测试更侧重于检查系统的功能是否满足用户需求，而白盒测试更侧重于检查系统是否按照设计实现，并识别潜在的编程错误和逻辑错误。根据测试的目的和要求，测试人员可以选择使用黑盒测试、白盒测试或两种测试方法的组合。

>1.6.1 白盒测试

下面内容来自沈备军老师授课ppt。

控制流测试（属白盒测试）

- 1) 语句覆盖法
- 2) 判定覆盖 (分支)
- 3) 条件覆盖
- 4) 判定/条件覆盖
- 5) 条件组合覆盖
- 6) 路径覆盖

1) 语句覆盖法

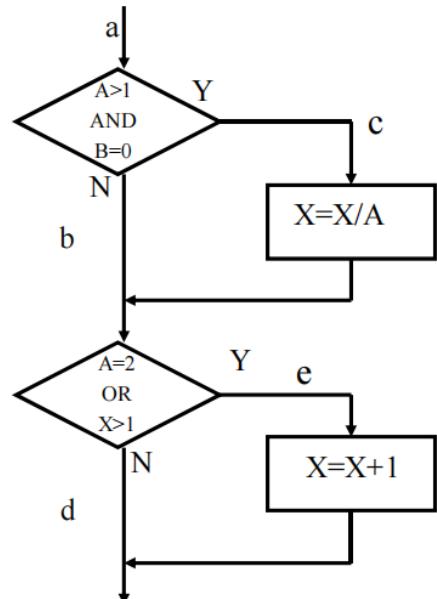
- 使得程序中的每一个语句至少被遍历一次
- 举例：对下列子程序进行测试

```
procedure example(y,z:real;var x:real);
begin
  if (y>1) and (z=0) then x:=x/y;
  if (y=2) or (x>1) then x:=x+1;
end;
```

- 测试用例：

A=2, B=0, X=3

程序流程图



2) 判定覆盖 (分支)

使得程序中每一个分支至少被遍历一次

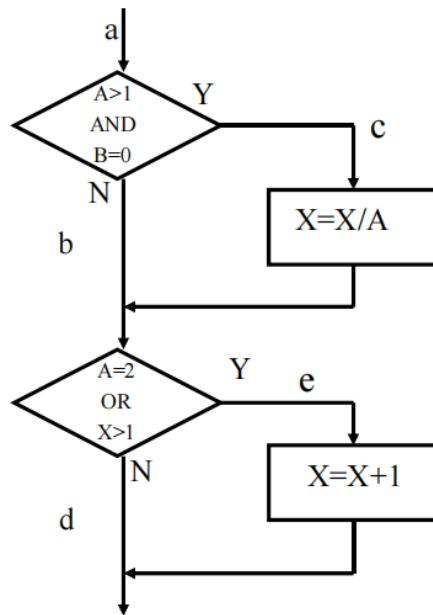
测试用例

1. $A=2, B=0, X=1$

(沿路径 ace)

2. $A=1, B=0, X=0$

(沿路径abd)



40

3) 条件覆盖

使得每个判定的条件获取各种可能的结果

在a点 $A > 1, A \leq 1, B = 0, B \neq 0$

在b点 $A = 2, A \neq 2, X > 1, X \leq 1$

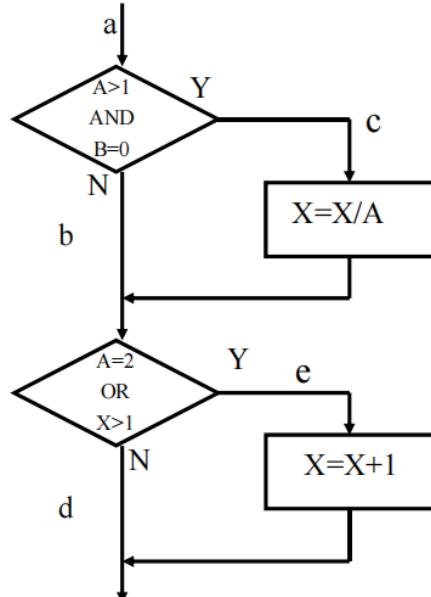
测试用例

1. $A=2, B=0, X=4$

(沿路径ace)

2. $A=1, B=1, X=1$

(沿路径abd)



41

4) 判定/条件覆盖

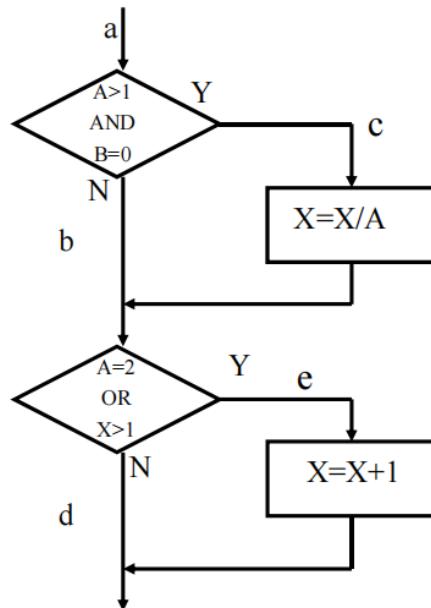
- 使得判定中的条件取得各种可能的值，并使得每个判定取得各种可能的结果
 - 测试用例

1. A=2, B=0, X=4

(沿路径ace)

2. A=1, B=1, X=1

(沿路径abd)



5) 条件组合覆盖

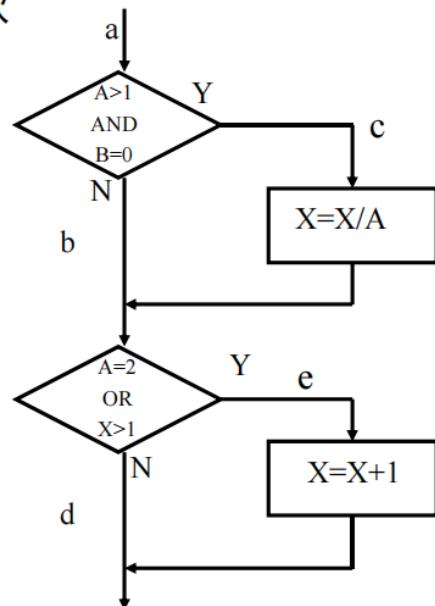
- 使得每个判定条件的各种可能组合都至少出现一次

□ 要求

- | | |
|--------------|-------------|
| 1. A>1, B= 0 | 5. A=2, X>1 |
| 2. A>1, B≠0 | 6. A=2, X≤1 |
| 3. A≤1, B=0 | 7. A≠2, X>1 |
| 4. A≤1, B≠0 | 8. A≠2, X≤1 |

□ 测试用例

- 1. A=2, B=0, X=4
 - 2. A=2, B=1, X=1
 - 3. A=1, B=0, X=2
 - 4. A=1, B=1, X=1

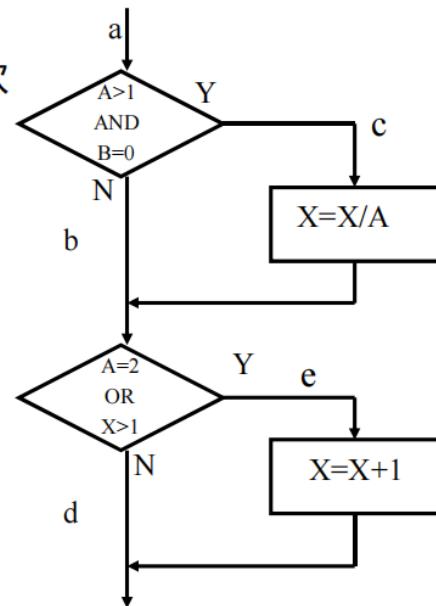


6) 路径覆盖

- 覆盖程序中所有可能的路径
- 如果程序中包含环路，则要求每条环路至少经过一次

测试用例：

A	B	X	覆盖路径
2	0	3	a c e L ₁
1	0	1	a b d L ₂
2	1	1	a b e L ₃
3	0	1	a c d L ₄



44

>1.6.2 黑盒测试

下面内容来自沈备军老师授课ppt。

- 黑盒测试把程序看成一个黑盒子，完全不考虑程序内部结构和处理过程。
- 黑盒测试是在程序接口进行测试，它只是检查程序功能是否按照需求规约正常使用。
- 黑盒测试又称功能测试、行为测试，在软件开发后期执行。

几种黑盒测试技术

- 1) 等价类划分法
- 2) 边界值分析法
- 3) 判定表法
- 4) 错误推测法

1) 等价类划分法(equivalence partitioning)

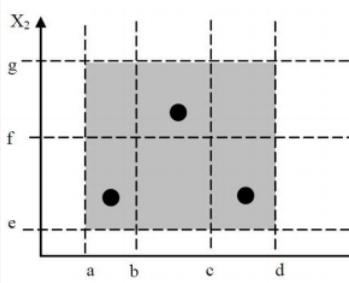
- 试遍所有输入数据是不可能的
 - 等价类划分的办法是把程序的输入域划分成若干部分，然后从每个部分中选取少数代表性数据当作测试用例。
 - 输入的数据划分为有效等价类和无效等价类
 - 输出也同样可以划分
- ✓ 例1：每个学生可以选取修1至3门课程
 - 有效等价类：选修1至3门课程
 - 无效等价类：没选修课程；超过3门课程
- ✓ 例2：录入百分数成绩，输出五级：A、B、C、D、F
 - 有效等价类为：A、B、C、D、F级的成绩
 - 无效等价类为：负分；大于100分

END

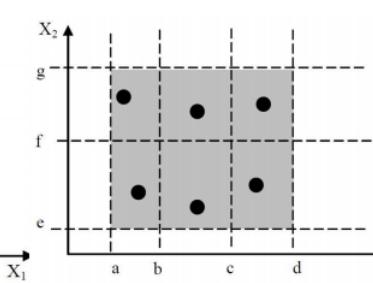
基于等价类的测试用例设计

- 根据是否考虑无效等价类，可划分为一般和健壮
- 根据测试时基于单缺陷还是多缺陷，可划分弱和强

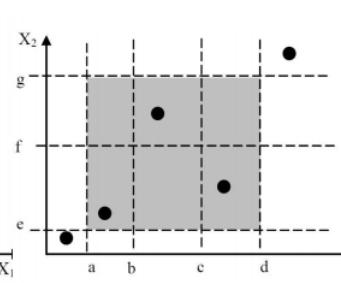
	弱（单缺陷假设）	强（多缺陷假设）
一般	弱一般等价类测试	强一般等价类测试
健壮	弱健壮等价类测试	强健壮等价类测试



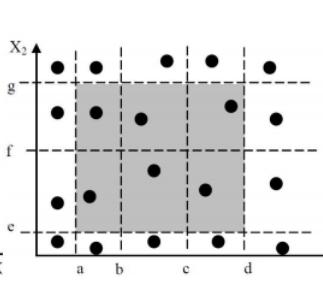
弱一般等价类测试



强一般等价类测试



弱健壮等价类测试



强健壮等价类测试

57

示例

输入货品信息，最后给出货品存放指示

- 编号必须为英文字母与数字的组合，由字母开头，包含6个字符，且不能有特殊字符
- 货品的登记数量在10到500之间（包含10和500）
- 货品的类型是设备、零件、耗材中的一种
- 货品的尺寸是大型、中型、小型中的一种，大型货品存放在室外堆场，中型货品存放在专用仓库，小型货品存放在室内货架
- 违反以上要求的登记信息被视为无效输入

采用弱健壮等价类测试法：

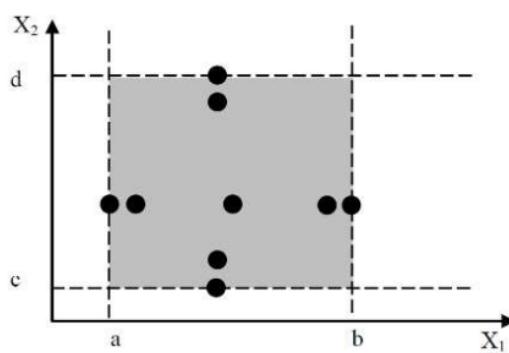
输入数据	有效等价类	无效等价类
货品编号	(1) 符合规则的编号	(7) 编号长度不为6 (8) 编号有特殊字符 (9) 编号不以字母开头
登记数量	(2) $10 \leq \text{数量} \leq 500$	(10) 数量 < 10 (11) 数量 > 500
货品类型	(3) {设备，零件、耗材}	(12) 非设备、零件、耗材中的一种
货品尺寸	(4) 大型 (5) 中型 (6) 小型	(13) 非大型、中型、小型中的一种

输入数据	预期输出	覆盖的等价类
A=EQ0101, B=30, C=设备, D=大型	合法登记信息，存放地为室外堆场	(1) (2) (3) (4)
A=CM0202, B=100, C=零件, D=中型	合法登记信息，存放地为专用仓库	(1) (2) (3) (5)
A=MT0303, B=400, C=耗材, D=小型	合法登记信息，存放地为室内货架	(1) (2) (3) (6)
A=EQ01023, B=30, C=设备, D=大型	非法登记信息	(7)
A=EQ0102#, B=30, C=设备, D=大型	非法登记信息	(8)
A=0102EQ, B=30, C=设备, D=大型	非法登记信息	(9)
A=EQ0101, B=0, C=设备, D=大型	非法登记信息	(10)
A=EQ0101, B=600, C=设备, D=大型	非法登记信息	(11)
A=EQ0101, B=30, C=装置, D=大型	非法登记信息	(12)
A=MT0202, B=100, C=耗材, D=中小型	非法登记信息	(13)

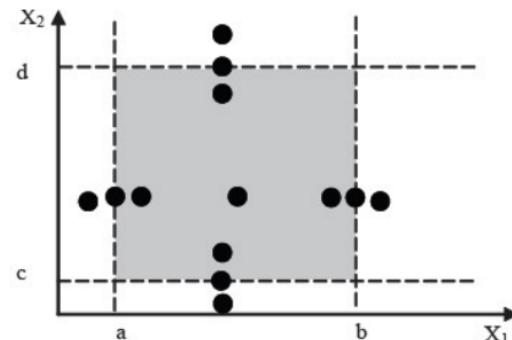
58

2) 边界值分析法(boundary value analysis)

- 边界值分析法使被测程序在边界值及其附近运行，从而更有效地暴露程序中潜藏的错误，是等价类划分方法的一种补充
- 不仅根据输入条件，它还根据输出情况设计测试用例



一般边界值测试



健壮边界值测试

59

3) 判定表法

□ When

- 检查输入条件的各种组合情况
- 在等价类划分方法和边界值方法中未考虑输入条件的各种组合，当输入条件比较多时，输入条件组合的数目会相当大

□ How

- 画出判定表，然后为判定表的每一例设计测试用例

举例

□ 例如，有一个处理单价为5角钱的饮料自动售货机软件，其需求规约如下：

□ 有一个处理单价为1元5角的盒装饮料的自动售货机软件。若投入1元5角硬币，按下“可乐”，“雪碧”或“红茶”按钮，相应的饮料就送出来。若投入的是两元硬币，在送出饮料的同时退还5角硬币。



3) 判定表法

□ When

- 检查输入条件的各种组合情况
- 在等价类划分方法和边界值方法中未考虑输入条件的各种组合，当输入条件比较多时，输入条件组合的数目会相当大

□ How

- 画出判定表，然后为判定表的每一例设计测试用例

根据判定表设计测试用例

为判定表的每个有意义的列设计一个测试用例

用例编号	规则	输入	预期输出
1	R1	投入1元5角, 按“可乐”	送出“可乐”饮料
2	R2	投入1元5角, 按“雪碧”	送出“雪碧”饮料
3	R3	投入1元5角, 按“红茶”	送出“红茶”饮料
4	R5	投入2元, 按“可乐”	找5角, 送出“可乐”
5	R6	投入2元, 按“雪碧”	找5角, 送出“雪碧”
6	R7	投入2元, 按“红茶”	找5角, 送出“红茶”

65

4) 错误推测法(error guessing)

- 基本概念
 - 猜测被测程序在哪些地方容易出错
 - 针对可能的薄弱环节来设计测试用例
- 例子1：对一个排序程序，可测试：
 - 输入表为空
 - 输入表中只有一行
 - 输入表中所有的值具有相同的值
 - 输入表已经是排序的
- 例子2：测试二分法检索子程序，可考虑：
 - 表中只有一个元素
 - 表长为 $2n$
 - 表长为 $2n-1$
 - 表长为 $2n+1$

66

课堂练习

请针对以下软件采用黑盒测试方法设计测试用例：

从键盘上输入三个整数（最大值为255），这三个数值表示三角形三条边的长度。然后，输出信息，以表明这个三角形是等腰、等边或是一般三角形，或不能构成三角形。

按照输入划分等价类

1、有效等价类：输入三个1-255的整数

2、无效等价类：

- (1) 输入的整数小于1
- (2) 输入的整数大于255
- (3) 输入的数字不是整数
- (4) 输入了非数字的字符

黑盒测试用例设计：

- 1、输入三个相等的整数 (50, 50, 50), 预期输出等边三角形
- 2、输入三个相等的整数 (-1, -1, -1), 预期输出不能构成三角形
- 3、输入三个相等的整数 (256, 256, 256), 预期输出不能构成三角形
- 4、输入三个整数 (其中两个相等), (2, 2, 25), 预期输出不能构成三角形
- 5、输入三个整数 (其中两个相等), (25, 25, 24), 预期输出等腰三角形
- 6、输入三个整数 (其中两个相等), (257, 257, 256), 预期输出不能构成三角形
- 7、输入三个整数 (其中两个相等), (-1, -1, 2), 预期输出不能构成三角形
- 8、输入三个互不相等的整数, (6, 7, 8), 预期输出一般三角形
- 9、输入三个互不相等的整数, (255, 256, 257), 预期输出不能构成三角形
- 10、输入(a,b,123), 预期输出不能构成三角形
- 11、输入 (12.67, 12, 2), 预期输出不能构成三角形

边界检查：

- 12、输入 (255, 255, 255), 预期输出等边三角形
- 13、输入 (1, 1, 1), 预期输出等边三角形
- 14、输入 (0, 0, 0), 预期输出不能构成三角形
- 15、输入 (255, 255, 254), 预期输出等腰三角形

>1.6 项目设计的软件过程

chapter 2. 上课练习题以及考试题选

>2.1 考前习题课内容

>2.1.1 黑盒测试1

黑盒测试



- 请采用事件流覆盖的黑盒测试技术，为比赛报名用例设计测试用例。

基本流

学生登入运动会管理系统进行项目报名。基本步骤如下：

1. include <<login>> 用例，学生登录系统
2. 系统显示所有已报名的和可报名的比赛项目
3. 学生在可报名比赛项目中选择喜欢的比赛，点击报名
4. 系统验证其总报名比赛项目数少于3个
5. 系统保存报名信息
6. 学生选择继续，回至2

备选流

3a 比赛报名数超过3项，验证未通过，用例结束

2-6a 学生选择退出，用例结束

Software Engineering

10

沈各军

事件流覆盖 (Event Flow Coverage) 是一种基于黑盒测试的测试方法，用于检查系统在不同输入条件下的行为是否符合预期。它通过描述各种可能的输入事件及其相应的响应事件之间的关系，来确定测试用例的设计和执行。

在事件流覆盖测试中，测试用例通常被设计为**一系列输入事件**，这些事件按照特定的顺序触发，并且每个事件都有一个或多个预期输出。测试人员需要根据事件之间的关系设计测试用例，以确保所有可能的事件序列都得到了覆盖。

具体来说，使用事件流覆盖测试时，测试人员通常需要执行以下步骤：

1. 根据需求分析和设计文档，确定所有可能的输入事件以及它们对应的输出事件；
2. 根据输入事件之间的关系，设计并执行测试用例，确保所有可能的事件组合都得到了覆盖；
3. 记录测试结果，并根据测试结果对系统进行改进。

通过事件流覆盖测试，可以有效地检查系统在不同场景下的行为是否符合预期，从而降低系统出错的风险。同时，它也能够提供对系统中潜在问题的深入理解，从而帮助开发人员提高代码质量和系统可靠性。

总之，事件流覆盖是一种基于黑盒测试的测试方法，能够帮助测试人员发现系统中潜在的问题，并提高测试的覆盖率和质量。

question：需要几个测试用例？

覆盖所有事件流——一共两个事件流（基本流一个，备选流一个；基本流里面有六个步骤，备选流里面有三个步骤）。总共需要两个测试用例。

>2.1.2 黑盒测试2

黑盒测试

- ◆ 在公司人事软件中，工资计算的规则为：
 - (a)年薪制员工：严重过失，扣当月薪资的4%；过失，扣年终奖的2%.
 - (b)非年薪制员工：严重过失，扣当月薪资的8%；过失，扣当月薪资的4%.
- ◆ 请采用判定表，设计测试用例。

这里的关键是条件判断（条件组合）：

1. 是不是年薪制员工？
2. 有没有严重过失？
3. 有没有过失？

总而言之，输入如果有条件组合，就要用判定表来覆盖所有情况。

黑盒测试

- 在公司人事软件中，工资计算的规则为：
(a)年薪制员工：严重过失，扣当月薪资的 4 %；过失，扣年终奖的2%。
(b)非年薪制员工：严重过失，扣当月薪资的 8 %；过失，扣当月薪资的 4 %。
• 请采用判定表，设计测试用例。

答：条件： C1：年薪制 C2：严重过失

结果： e1:扣月4% e2:扣月8% e3:扣年2%

		1	2	3	4
条件：	C1	1	1	0	0
	C2	1	0	1	0
动作	e1	✓			✓
	e2			✓	
	e3		✓		

名词解释

①模块化 ②回归测试 ③重构

答：

- ◆ 模块化 (modularity)：将一个复杂的系统自顶向下地分解成若干模块 (module)，每个模块完成一个特性，所有的模块组装起来，成为一个整体，完成整个系统所要求的特性。
- ◆ 回归测试：为了保证软件返工时没有引进新的错误，要全部或部分地重复以前做过的测试。
- ◆ 重构 (refactoring, restructuring)：对需求、设计或代码（在同一抽象级别上）进行改进，而不改变原有软件的功能。

>2.1.4 软件设计过程

设计软件过程

项目组计划为上海市政府开发一个志愿者管理系统，该系统具有以下功能：

- 1)市民可以报名担任某项活动的志愿者工作
- 2)市政府进行志愿者的遴选
- 3)市政府给志愿者分配任务
- 4)志愿者可以查到分配的任务，并填写完成进度和状态
- 5)活动结束后生成统计报表，并进行多维分析

项目组经过估算，该系统需要5个月开发周期，从2023年6月1日开始开发，市政府希望10月1日本系统可以投入运行，因为正好有一个重要活动在10月1日开始招募志愿者。显然本项目进度很紧。同时项目决定采用微服务架构，但项目组成员对此技术都不熟悉。请识别出项目的主要风险，采用RUP或Scrum等演化过程为本项目设计一个合适的软件过程，并为每个迭代标上计划的起止日期。注：每个迭代中应列出其包括的多个活动或任务。

进度风险如何解决？（不能外包）

1. 复用前人的ui代码，复用开源代码
2. 采用增量式递交——前三个功能或者前四个功能先做，在十月一日先递交，余下的功能在十月三十一日递交。
 - 每个迭代都对某个风险进行缓解

参考答案

1. 初始阶段 (1个迭代) 6月1日 - 6月30日

目标：开发界面原型，解决需求风险

1.1 确定stakeholders

1.2 调查需求

1.3 编写《需求规格说明书》

1.4 实现用户界面原型

1.5 审评《需求规格说明书》和界面原型

2. 细化阶段 (1个迭代) 7月1日 - 7月31日

目标：开发技术原型（又称架构原型），解决架构风险

2.1 架构分析

2.2 架构设计

2.3 架构编码

2.4 架构测试

3. 构造阶段 (1个迭代) 8月1日 - 8月31日

目标：实现优先级高的需求，解决进度风险

3.1 R1 Beta需求分析，包括功能1-4

3.2 R1 Beta设计

3.3 R1 Beta实现

3.4 R1 Beta测试

4. 移交阶段 (3个迭代) 9月1日 - 10月30日

目标：优先级高的需求先实现，并增量式递交系统，解决进度风险

4.1 移交迭代1(R1) 9月1日 - 9月17日

4.1.1 在用户环境中安装R1 Beta

4.1.2 用户培训R1 Beta

4.1.3 用户测试和试用R1 Beta

4.1.4 根据用户反馈修改R1 Beta，形成R1

4.1.5 在用户环境中安装R1

4.2 移交迭代2 (R2 Beta) 9月18日 - 10月13日

4.2.1 R2 Beta需求分析，包括功能5

4.2.2 R2 Beta设计

4.2.3 R2 Beta实现

4.2.4 R2 Beta测试

4.3 移交迭代3(R2) 10月14日 - 10月31日

4.3.1 在用户环境中安装R2 Beta

4.3.2 用户培训R2 Beta

4.3.3 用户测试和试用R2 Beta

4.3.4 根据用户反馈修改R2 Beta，形成R2

4.3.5 在用户环境中安装R2

-----以下为期中考试之前的部分-----

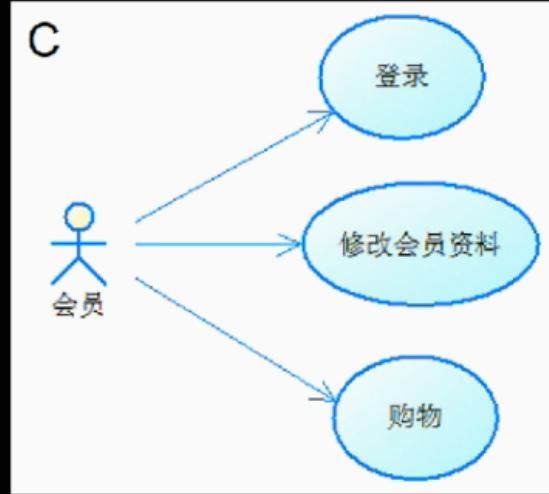
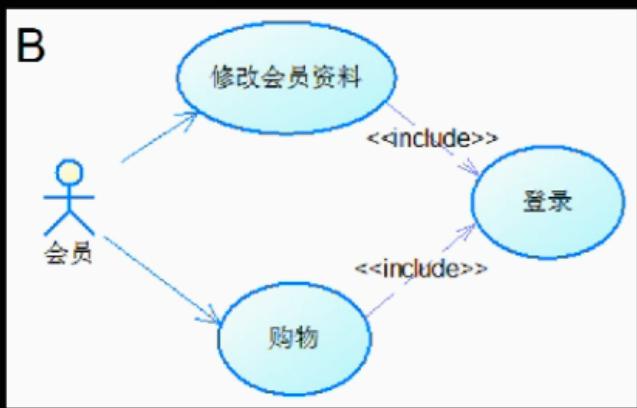
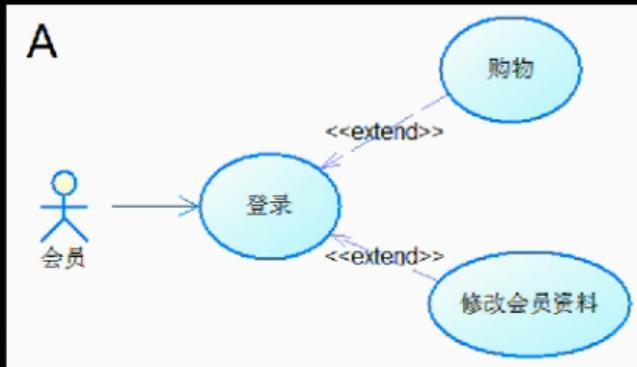
面向对象分析与设计习题课

用UML分析，包括三个点：

1. 建立用例模型（用例图）
2. 建立概念模型（类图）
3. 建立分析模型

>2.1.5 用例建模

哪个更好？



答案：B

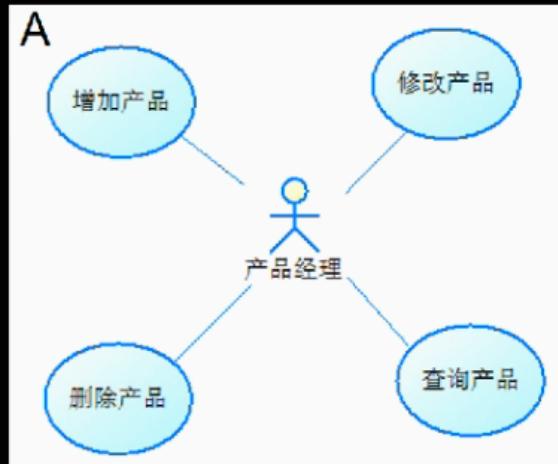
extend是有的时候做，有的时候不做，显然是不对的。

而且在这个系统中，最重要的、最有价值的部分是购物和修改会员资料，因此这两应该是base case，选择B。

【用例是能卖的价值】

对还是不对？

产品销售系统



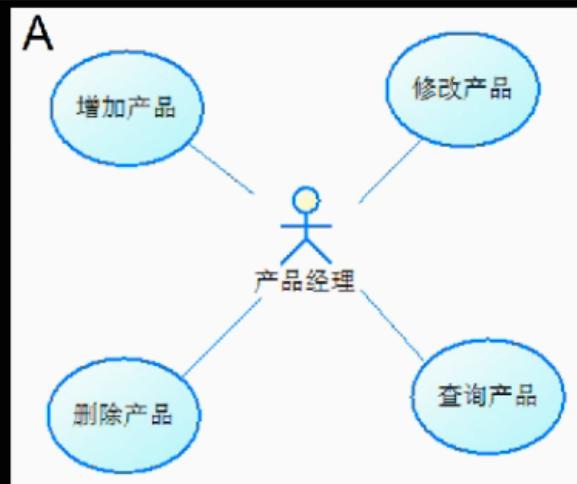
对还是不对？好还是不好？

actor肯定是对的。

问题在于粒度太细了，增删改查经常是一起做的。经常一起做的事情放在一起。include常常是几个use-case的共性拿出来，千万别做功能分解。

对还是不对？

产品销售系统

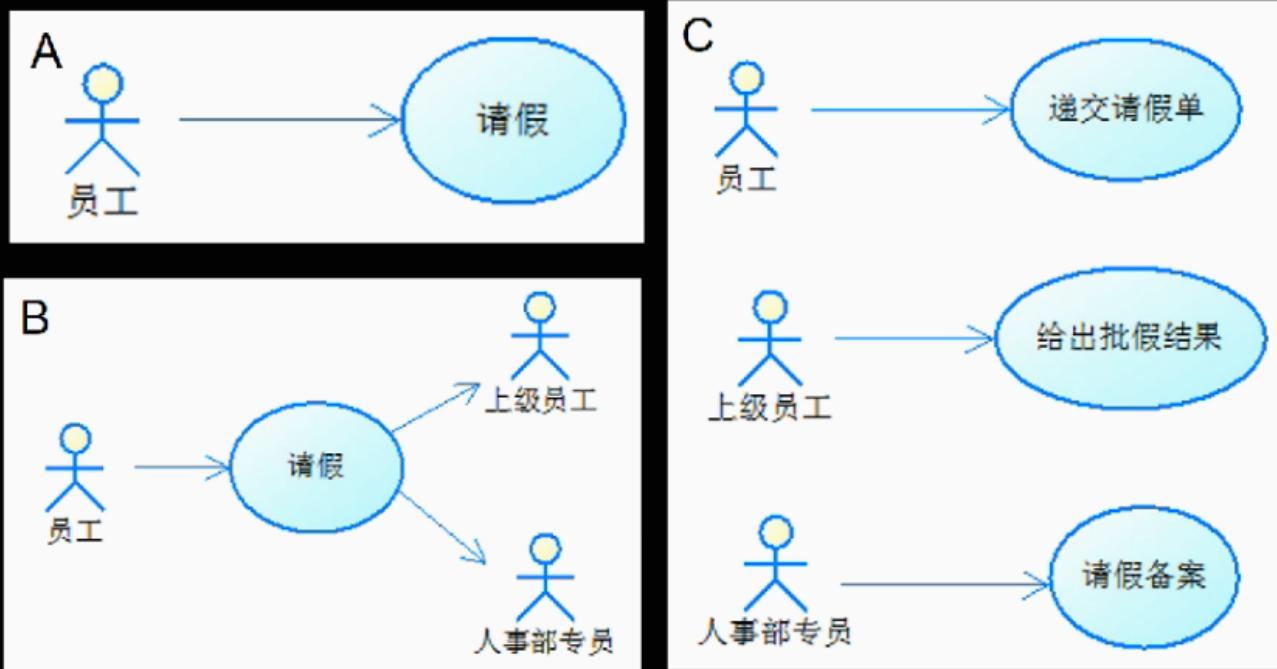


B



哪个对？

人事系统



Software Engineering

6

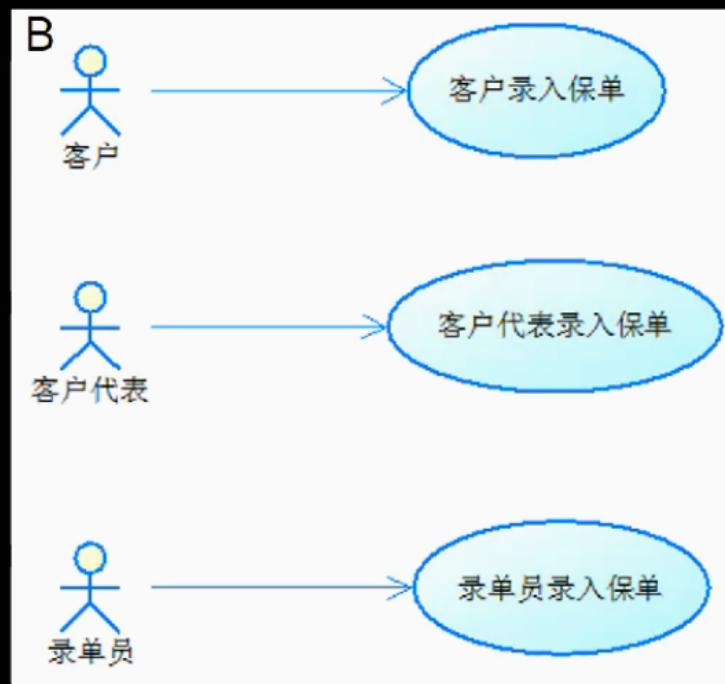
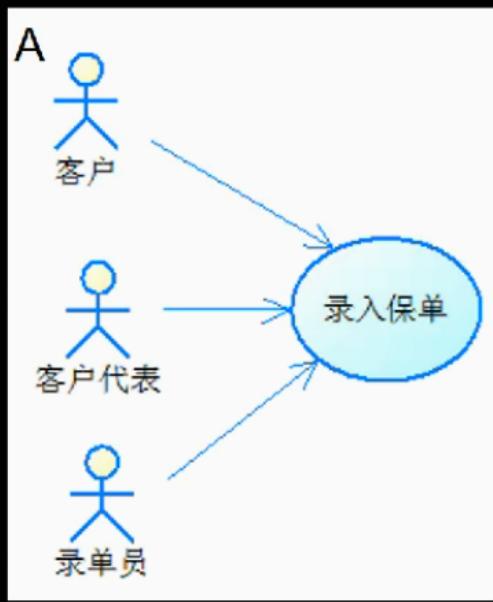
沈各军

哪种方案更好？

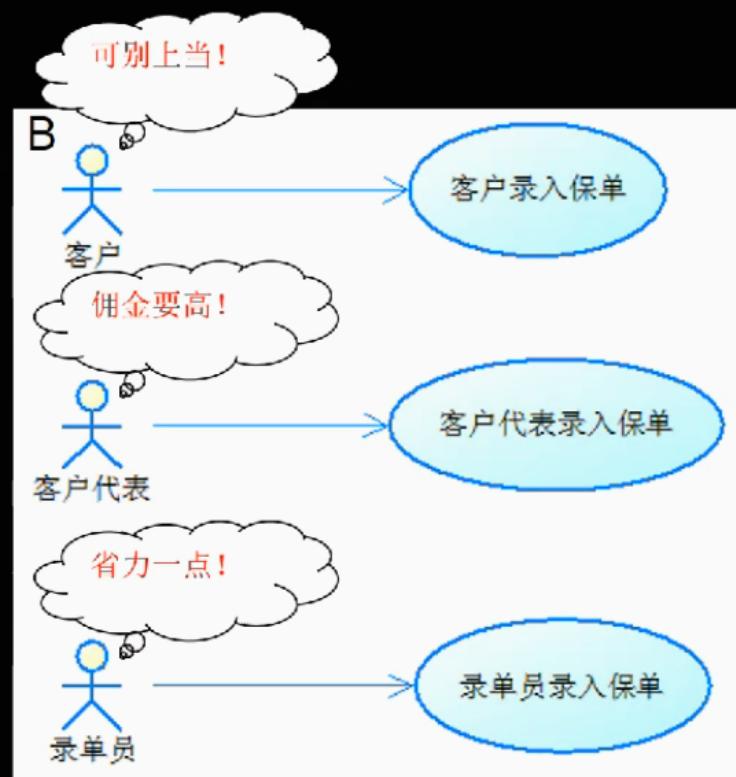
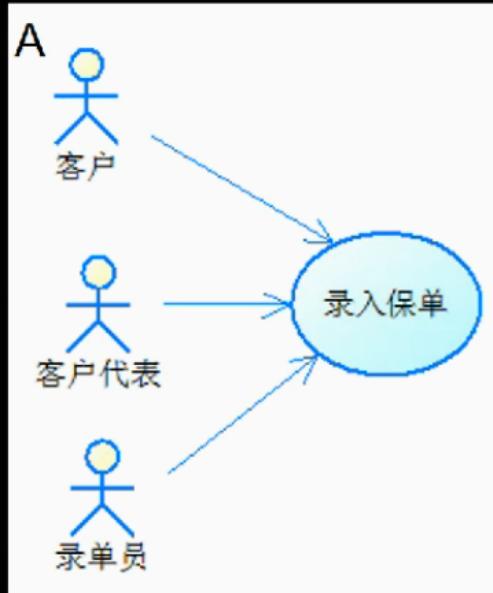
还是考粒度的问题，人事部的人其实也是员工 (((

最好的还是C。因为这三件事情不是一起做的。因为这三件事情不是同步的，因此分三件事情。

有什么不同？



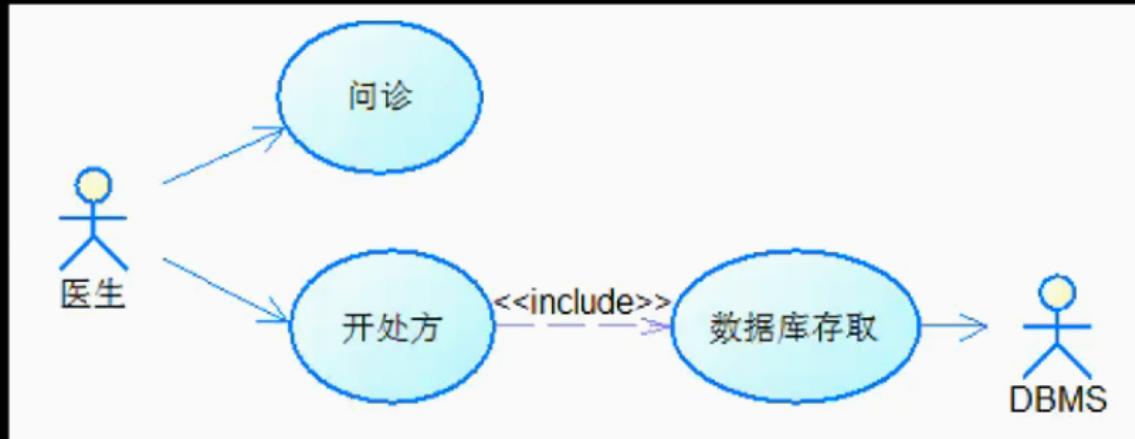
有什么不同？



因为这仨人想法不一样，因此界面要做不一样的.....

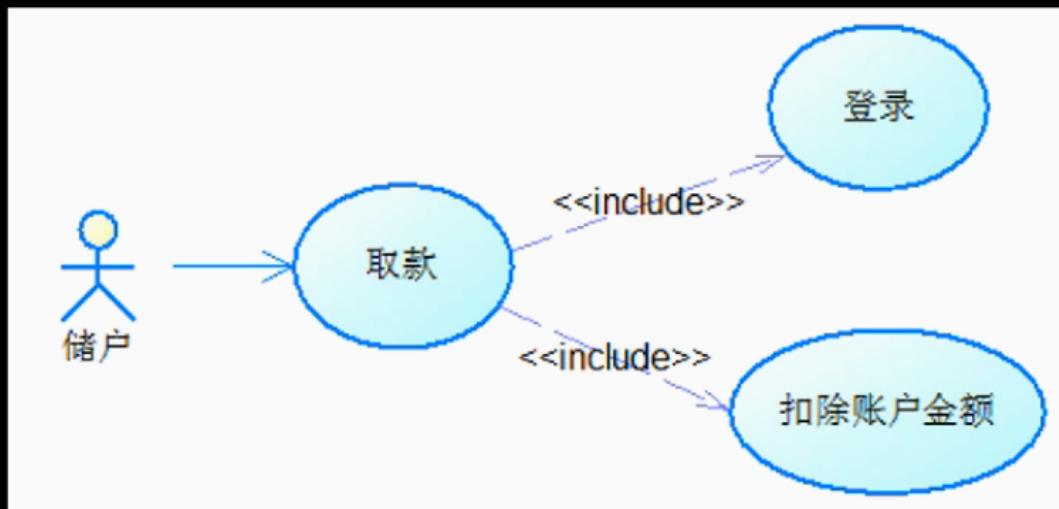
对还是不对？

医院管理系统



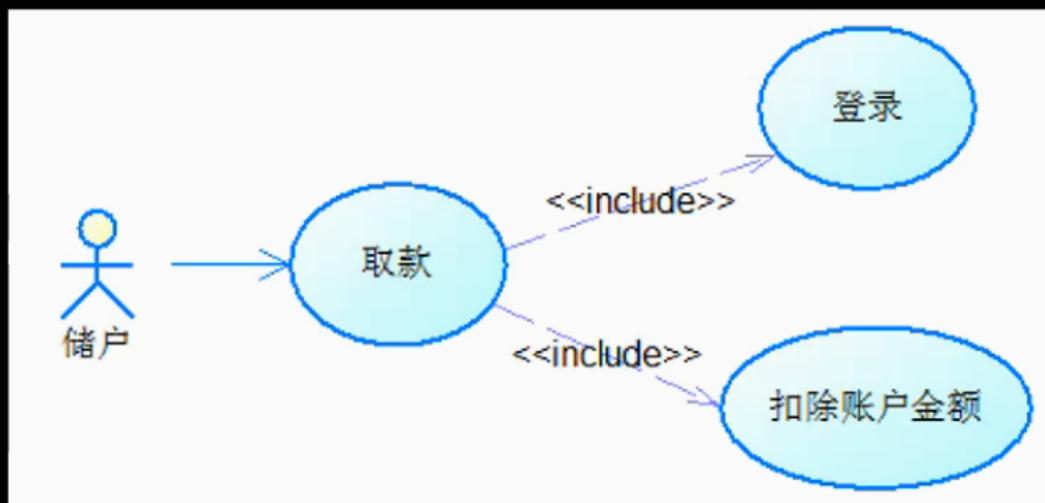
对还是不对？

ATM系统



对还是不对？

ATM系统

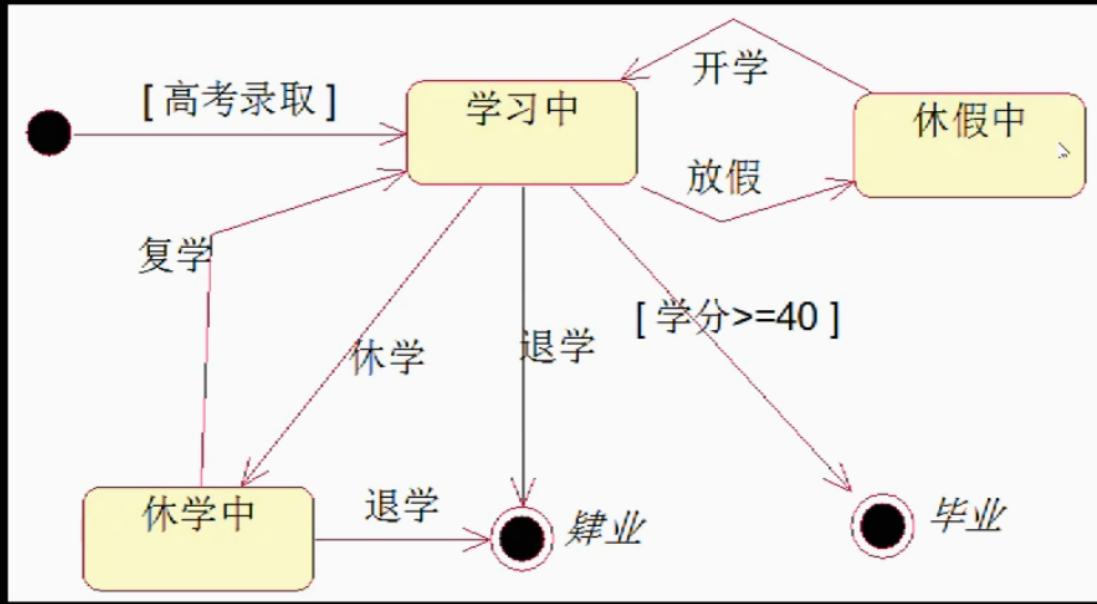


最常犯“粒度”错误：把步骤当作用例

减钱的粒度太小了。这只是一个比较小的步骤而已。

状态图

- ◆ 按以下陈述画出大学生（student）这个类的状态图
- ◆ 一个学生通过高考被一所大学录取，在学习期间，学生可能会因为生病等原因暂时休学，休学后仍可重新复学；也可能退学，中止学业；当然，大多数学生会修完40学分后正常毕业。学习期间，学生每年还有两个假期进行休假。



>2.1.7 设计模式

创建型设计模式（Creational Design Patterns）和结构型设计模式（Structural Design Patterns）都是面向对象编程中常用的设计模式。

创建型设计模式 用于处理对象的创建机制，包含以下几种模式：

1. 工厂方法模式（Factory Method Pattern）：定义一个用于创建对象的接口，让子类决定实例化哪个类。
2. 抽象工厂模式（Abstract Factory Pattern）：提供一个接口，用于创建相关或依赖对象的家族，而不需要明确指定它们的具体类。
3. 建造者模式（Builder Pattern）：将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。
4. 单例模式（Singleton Pattern）：确保一个类只有一个实例，并提供对该实例的全局访问点。
5. 原型模式（Prototype Pattern）：用原型实例指定创建对象的种类，并且通过克隆这些原型来创建新的对象。

结构型设计模式 用于处理类或对象间的组合关系，包含以下几种模式：

1. 适配器模式 (Adapter Pattern) : 将一个类的接口转换成客户端希望的另一个接口, 使得原本由于接口不兼容而不能在一起工作的那些类能够在一起工作。
2. 桥接模式 (Bridge Pattern) : 将抽象部分与它的实现部分分离, 使它们都可以独立地变化。
3. 组合模式 (Composite Pattern) : 将对象组合成树形结构以表示“部分-整体”的层次结构, 使得客户端对单个对象和组合对象的使用具有一致性。
4. 装饰器模式 (Decorator Pattern) : 动态地给一个对象添加一些额外的职责, 即增加其功能。
5. 外观模式 (Facade Pattern) : 为子系统中的一组接口提供一个统一的接口, 用来访问子系统中的各种功能。
6. 享元模式 (Flyweight Pattern) : 运用共享技术来有效地支持大量细粒度对象的复用。

总之, 创建型设计模式和结构型设计模式是面向对象编程中常用的两种设计模式。前者主要用于处理对象的创建机制, 后者主要用于处理类或对象间的组合关系。选择适当的设计模式可以提高代码的可读性、可维护性和可扩展性。