题型？？？？？？？？？？？？？？？？？？？？

第一章

快速原型模型基本思想：

通过开发系统原型和用户反复交互，以明确需求，使系统在不断调整与修改中得以进化成熟。

增量模型：

● 增量模型也称为渐增模型，是Mills等人1980年提出来的。

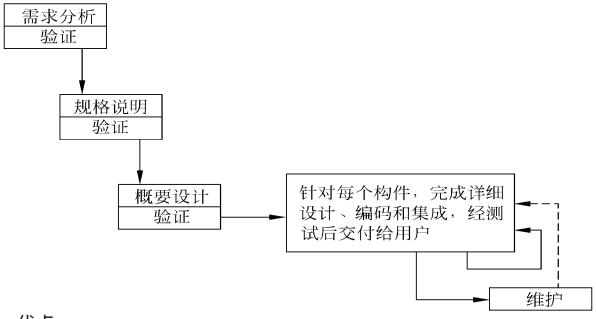
使用增量模型开发软件时，把软件产品作为一系列的增量构件来设计、编码、集成和测试

每个构件由多个相互作用的模块构成，并且能够完成特定的功能。

● 注意：

分解成增量构件时，应该使构件的规模适中

把新构件集成到现有软件中时，所形成的产品必须是可测试的



● 优点

能在较短时间内向用户提交可完成一些有用的工作产品

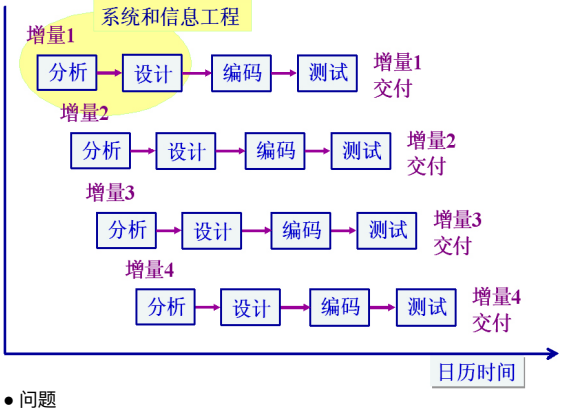
用户有较充裕的时间学习和适应新产品

项目失败的风险较低

最重要的系统服务将接受最多的测试。

（优先级最高的服务首先交付，然后再将其他增量构件逐次集成进来）

● 增量构件的开发可以采用瀑布模型的方式，如图所示。

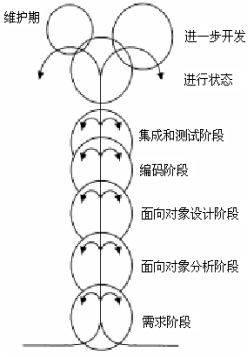


● 问题

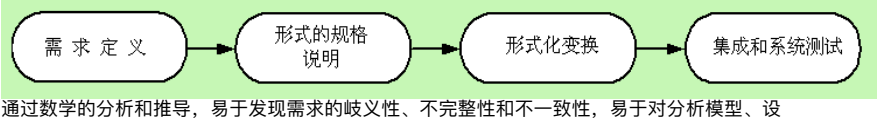
在把每个新的增量构件集成到现有软件体系结构中时，必须不破坏原来已经开发出的产品。

软件体系结构必须是开放的，即向现有产品中加入新构件的过程必须简单、方便。

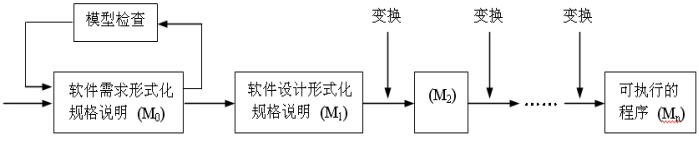
因此，采用增量模型比采用瀑布模型和快速原型模型更需要精心的设计。
● 增量开发的作用
增量式开发是20世纪70年代早期由Mills提出的。但直到80年代末，当Mills和他的助手们的净室论文和专项报告开始出现的时候才获得认同。
Fred Brooks描述了增量式开发方法的深刻影响，针对该方法在软件实践方面的作用，给出了影响广泛的评论：“没有银弹：软件工程的本质和偶然性。”Brooks的观察在工业实践中得到了证实。
喷泉模型是典型的面向对象生命周期模型（所有模型里只有它是面向对象的），“喷泉”一词体现了迭代和无间隙特性。图中代表不同阶段的圆圈相互重叠，这明确表示两个活动之间存在重叠。



形式化方法变换成程序（形式化程序求精）
形式化系统开发模型是一种基于形式化数学变换的软件开发方法
凡是采用严格的数学语言，具有精确的数学语义的方法，都称为形式化方法。
它可将系统规格说明转换为可执行的程序。



通过数学的分析和推导，易于发现需求的岐义性、不完整性和不一致性，易于对分析模型、设计模型和程序进行验证。
通过数学的演算，使得从形式化功能规约到形式化设计规约，以及从形式化设计规约到程序代码的转换成为可能。
● 形式化方法的主要特点
用数学记号表达的详细的形式化规格说明。
设计、实现和单元测试等开发过程由变换开发过程代替。
通过一系列变换将形式化规格说明细化成为程序。（形式化程序求精）



形式化变换

极限编程：
敏捷过程中最重要的部分，广泛应用于需求模糊且经常改变的场合。
● 有效实践（6条）课本p26
✓ 客户是开发团队的成员：
必须至少有一名客户代表在项目的整个开发周期中与开发人员在一起紧密地配合工作，客户代表负责确定需求、回答开发人员的问题并且设计功能验收测试方案。

✓短交付周期：
每两周完成一次的迭代工程实现了用户的一些需求，交付出目标系统的一个可工作的版本。通过向有关的用户演示迭代生产的系统，获得他们的反馈意见。
✓结对编程：
结对编程就是由两名开发人员在同一台计算机上共同编写解决同一个问题的程序代码，通常是一个人编码，另一个人对代码进行审查与测试，以保证代码的正确性与可读性。结对编程是加强开发人员相互沟通与评审的一种方式。
✓集体所有：
极限编程强调程序代码属于整个开发小组集体所有，小组每个成员都有更改代码的权利，每个成员都对全部代码的质量负责。
✓可持续的开发速度：
开发人员以能够长期维持的速度努力工作。XP规定开发人员每周工作时间不超过40小时，连续加班不可以超过两周，以避免降低生产率。
✓及时调整计划：
计划应该是灵活的、循序渐进的。制定出项目计划之后，必须根据项目的进展请款及时进行调整，没有一成不变的计划。

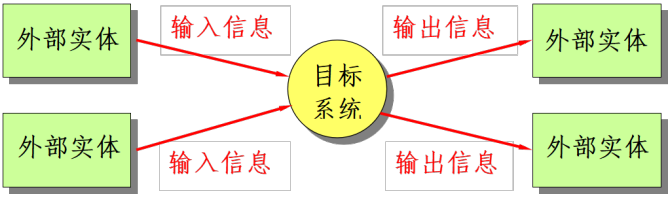
第二章

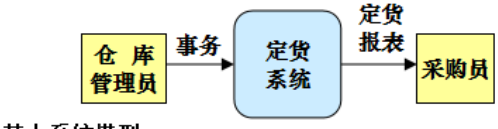
数据流图

并且要用第五章知识建房子

订货系统例子（三个图：

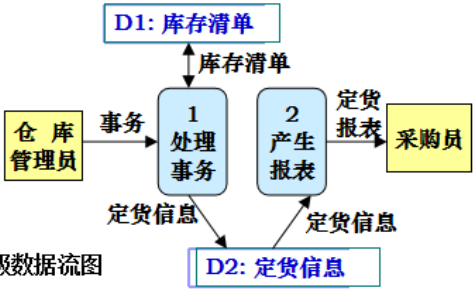
顶层数据流图





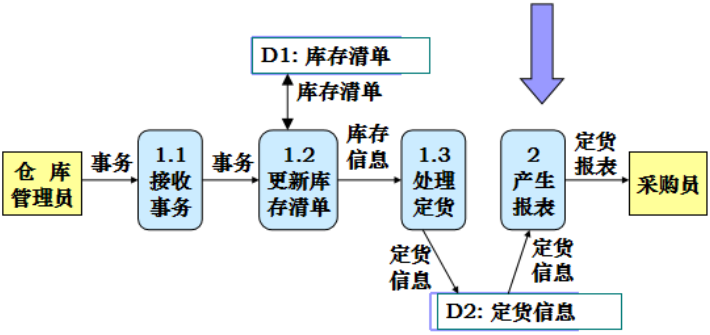
基本系统模型

功能级数据流图



功能级数据流图

进一步细化数据流图



)

？？？？？？？？？？？？？ 这个软件结构图怎么画？？？？？？？？？？？？？？

考务系统的简单看一下

医院的仔细看（数据流图、软件结构图）

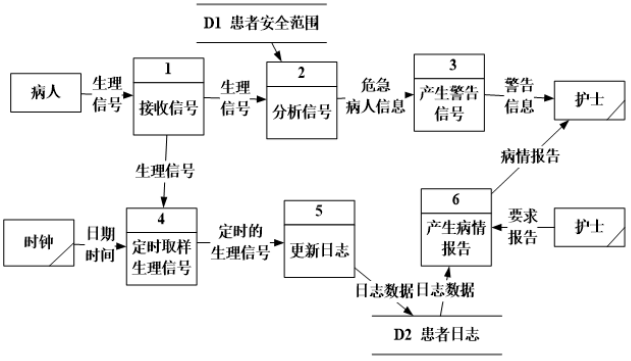
系统功能要求：

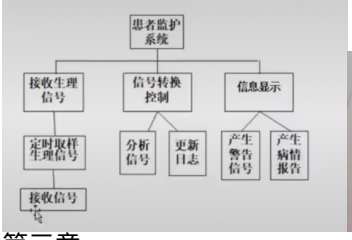
1、监视病员的病症（血压、体温、脉搏等）

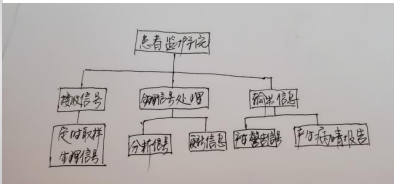
2、定时更新病历

3、病员出现异常情况时报警。

4、随机地产生某一病员的病情报告。







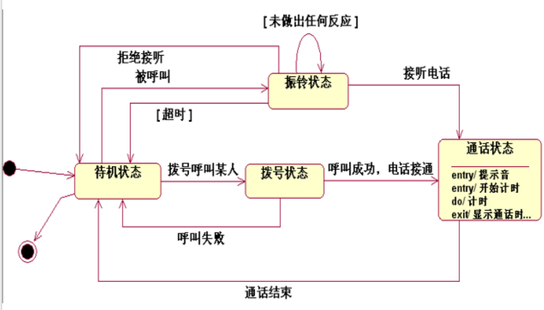
第三章

需求分析是解决做什么的问题，总体设计和详细设计都是解决怎样做的问题

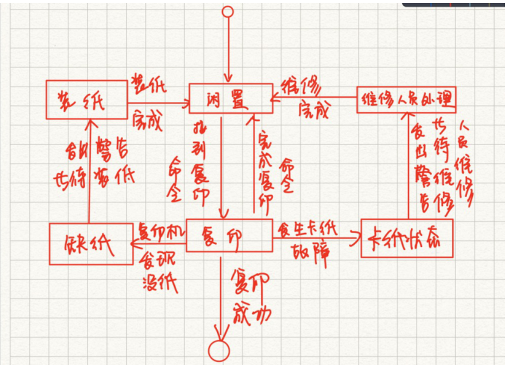
需求分析生成的三个模型

(1) 必须定义软件应完成的功能，要求建立功能模型

(2) 必须理解并描述问题的信息域，应该建立数据模型
(3) 必须描述作为外部事件结果的软件行为，要求建立行为模型
会画状态图，看**手机状态图**



课堂作业：复印机
工作过程大致如下：
未接到复印命令时处于闲置状态
一旦接到复印命令则进入复印状态
完成一个复印命令规定的工作后又回到闲置状态，等待下一个复印命令；
如果执行复印命令时发现没纸，则进入缺纸状态，发出警告，等待装纸
装满纸后进入闲置状态，准备接收复印命令；
如果复印时发生卡纸故障，则进入卡纸状态，发出警告等待维修人员来排除故障
故障排除后回到闲置状态



第四章

形式化规格说明建模（多选，字母缩写就不用记了）
操作类：基于状态和转移
Petri网、有限状态机和状态图
描述类：基于数学公理和概念
基于逻辑的描述方法：命题线性时态逻辑、一阶线性时态逻辑、计算树逻辑
基于代数的描述方法：Z语言、VDM和Larch
双重类：兼有操作类和描述类两者的特点
扩展状态机（ESM）、实时时态逻辑（RTTL）
形式化程序求精
研究:如何从形式化的规格说明推演出具体的面向计算机的程序代码的全过程
? ?
有穷状态机要知道每一项都是什么意思？？

$$M = (Q, \Sigma, \delta, q_0, F)$$

$Q = \{q0, q1, ..., qn\}$有限状态集合

$\Sigma = \{\sigma 1, \sigma 2, ..., \sigma m\}$有限输入字符集合

$\delta : Q \times \Sigma \to Q$状态转换函数

$q0 \in Q$初始状态

$F \subseteq Q$终结状态集合

Z语言

Z语言是由牛津大学程序设计研究小组开发的一种描述形式语言，在ISO指导下的国际标准化Z工作于2002年完成。

Z语言将事物的状态和行为用数学符号形式化表达的语言，为编写计算机程序和验证计算机程序的正确性提供依据

Z语言是软件工程中编码之前的规格说明语言

Z语言是一种以一阶谓词演算为主要理论基础，是一种功能性语言

第五章

画房子? ?

软件设计任务（怎样做）2：

概要设计：大房子（软件结构图（上下级为调用关系））

详细设计：（设计算法）（完了之后写程序）

模块独立性（耦合、内聚（能对应起来））

耦合：是对一个软件结构内不同模块之间互联程度的度量（应追求尽可能松散耦合的系统）

➢ 内容耦合（耦合程度最高，应坚决避免）

一个模块访问另一个模块的内部数据

一个模块不通过正常入口而转到另一个模块的内部

两个模块有一部分程序代码重叠（只可能出现在汇编程序中）

一个模块有多个入口（这意味着一个模块有几种功能）

➢ 公共环境耦合

两个或多个模块通过一个公共数据环境相互作用

➢ 特征耦合

当把整个数据结构作为参数传递而被调用的模块只需要使用其中一部分数据元素时

➢ 控制耦合

如果传递的信息中有控制信息(尽管有时这种控制信息以数据的形式出现)

➢ 数据耦合

如果两个模块彼此间通过参数交换信息，而且交换的信息仅仅是数据

（耦合程度：（喜欢）数据耦合<控制耦合和特征耦合<公共环境耦合<内容耦合（不喜欢））

内聚：

内聚标志着一个模块内各个元素彼此结合的紧密程度，他是信息隐藏和局部化概念的自然扩展（力求高内聚）（模块的高内聚往往意味着模块间的松耦合）

● 低内聚：

➢ 巧合内聚（Coincidental Cohesion）

当几个模块内正好有一段代码是相同的，将它们抽取出来形成单独的模块，即巧合内聚模块

➢ 逻辑内聚

这种模块(例如，读一个记录，写一个记录)把几种相关的功能组合在一起

每次被调用时，由传送给模块的判定参数来确定该模块应执行哪一种功能。

➢ 时间内聚

一个模块包含的任务必须在同一段时间内执行

● 中内聚：

➢ 过程内聚

一个模块内的处理元素是相关的，而且必须以特定次序执行

➢ 通信内聚

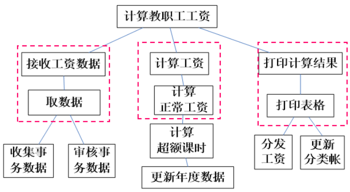
如果一个模块内各功能部分都使用了相同的输入数据，或产生了相同的输出数据

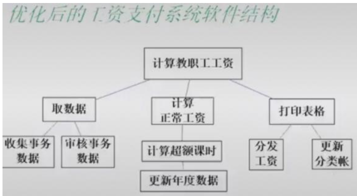
● 高内聚：

➢ 顺序内聚：

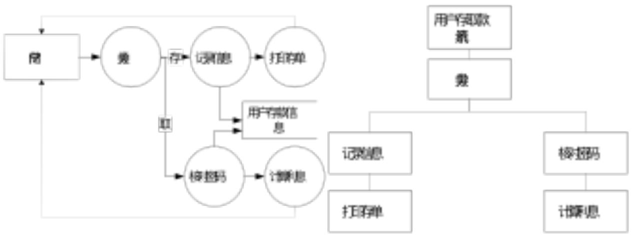
一个模块内的处理元素和同一个功能密切相关，而且这些处理必须顺序执行(通常一个处理元素的输出数据作为下一个处理元素的输入数据)

➢ 功能内聚：
模块内所有处理元素属于一个整体，完成一个单一的功能.
启发性规则
1. 争取低耦合、高内聚（增加内聚 > 减少耦合）
2. 模块规模适中：过大不易理解；太小则接口开销过大。注意分解后不应降低模块的独立性。
3. 适当控制：深度 = 分层的层数。过大表示分工过细。
□ 宽度 = 同一层上模块数的最大值。过大表示系统复杂度大。
（扇入扇出简单看一下
扇出 = 一个模块直接调用\控制的模块数。（扇出太多说明模块需要分解）
□ 扇入 = 直接调用该模块的模块数
在不破坏独立性的前提下，扇入大的比较好。
尽可能减少高扇出结构，随着深度增大扇入。
模块的扇出数过大，就意味着该模块过分复杂，需要协调和控制过多的下属模块
应当适当增加中间层次的控制模块）
4、作用域在控制域内（仔细）
控制域：这个模块本身以及所有直接或间接从属于它的模块的集合
作用域：M中的一个判定所影响的模块
重头戏：面向数据流的设计方法：
数据流分为：
变换流：事实上所有信息流都可归结为变换流
事务流：当信息流有明显的“发射中心”时
（知道怎么变房子就行，支付、医患的例子）
工资支付系统软件结构图





储户存款系统软件结构



第六章
详细设计的任务也是解决怎样做的问题，决定了最终代码质量关系
程序流程图会画（视频里说课下绘画（不可能是原题，描述性的要学会怎么画）伪码 123都要回? ？？？？？？？？？？？？？？？？？？？？？？？？？？？？）
判定表看看（不考原题）
判定表表示复杂的条件(input)组合与应做动作(output)之间的对应关系。
判定表的出现
先于软件工程十几年，但不失用于过程设计
判定表的性质
提供了一种把各种动作与条件（在处理解说词中描述的）转换为表格形式的表达式
判定表的好处
不容易产生误解，还可作为一种表驱动算法的机器直接输入使用

第七章软件测试

（单元测试完了是集成测试）集成测试分为渐增式（几种策略看一下，看动态图）和非渐增式
渐增测试 的几种策略

（1）自顶向下Top-down testing
第1步：测试顶端模块，用存根程序(stub只是对低层模块的模拟)代替直接附属的下层模块
第2步：根据深度优先或宽度优先的策略，每次用一个实际模块代换一个stub。
第3步：在结合进一个模块的同时进行测试。
第4步：回归测试(regression testing)——全部或部分地重复以前做过的测试。
（2）自底向上Bottom - up testing
第1步：把低层模块组合成族，每族实现一个子功能。
第2步：用驱动程序(Driver)协调测试数据的I\O，测试子功能族。
第3步：去掉Driver，自下而上把子功能族合成更大的子功能族。
注意：两种策略的优、缺点刚好互补，但单用其中任一种都不实际，通常根据软件的特点将二者混用。

白盒测试（三种）：语句覆盖，判定覆盖，条件覆盖（要求会投应例，设计应例）

语句覆盖:每个语句至少执行一次。
判定覆盖(Branch coverage)：在⑴的基础上，每个判定的每个分支至少执行一次。
条件覆盖(Condition coverage)：在⑴的基础上，使每个判定表达式的每个条件都取到各种可能的结果

第八章软件维护

软件维护四个维护，看看（包括数字，能对应）

软件维护——在软件已经交付使用之后，为了改正错误或满足新的需要而修改软件的过程。
A:改正性维护(corrective maintenance)——诊断和改正错误,约占全部维护活动的 17~20%；
B：适应性维护(adaptive maintenance) ——为了和变化了的环境（如软\硬件升级、新数据库等）适当地配合而修改软件，约占全部维护活动的18~25%；
C：完善性维护（perfective maintenance) ——为了增加新功能，修改已有功能，改造界面，增加HELP等，而修改软件,约占全部维护活动的50~66%；
D：预防性维护(preventive maintenance) ——为了改进未来的可维护性或可靠性，或为了给未来的改进奠定更好的基础而修改软件，与其它维护活动共占总维护的4%左右。
注：① 一般维护的工作量占生存周期70%以上，维护成本约为开发成本的4倍(80 - 20 Rule)；
② 文档维护与代码维护同样重要。

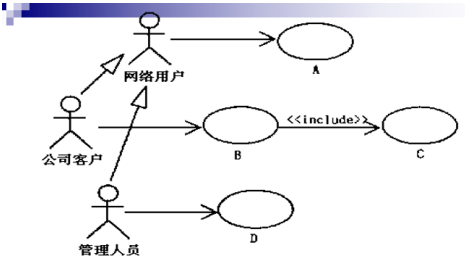
第九章

面向对象方法生成三个模型：对象模型、动态模型、功能模型。

对象模型：描述系统数据结构，最重要，最基本，最核心
动态模型：描述系统控制结构（状态图、用事件跟踪图（ＤＦＤ））
功能模型：描述系统功能

用例图：

例：（软件设计师考题）2004年上半年试题三（此题跟考试题很像，但是是直接画图）
某电话公司决定开发一个管理所有客户信息的交互式网络系统。系统的功能如下：
1. 浏览客户信息：任何使用Internet的网络用户都可以浏览电话公司所有的客户信息(包括姓名、住址、电话号码等)。
2. 登录：电话公司授予每个客户一个帐号。拥有授权帐号的客户，可以使用系统提供的页面设置个人密码，并使用该帐号和密码向系统注册。
3. 修改个人信息：客户向系统注册后，可以发送电子邮件或者使用系统提供的页面，对个人信息进行修改。
4. 删除客户信息：只有公司的管理人员才能删除不再接受公司服务的客户的信息。
（首先是站在用户的角度：分析有几个用户？
用户和用户之间有没有存在泛化关系（子类指向父类）
写每个用户的功能：包括
include：父功能和子功能：如修改个人信息和登录必须是先登录后修改信息，故登录是修改信息的子类（父功能指向子功能）
exclude：（非必须的扩充功能）扩展的功能指向被扩展的功能）
再看一下自动售货机那个例子p227（可以看，用来理解）



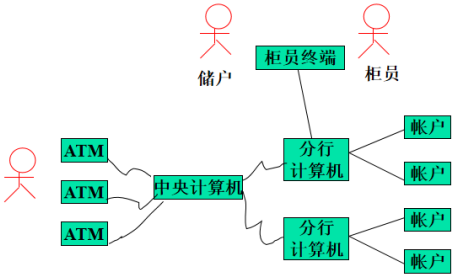
[问题1] 在需求分析阶段，采用UML的用例图描述系统功能需求，如图所示。请指出图中的A、B、C 和D分别是哪个用例?

第十章

泛化（公司客户、管理人员可以泛化成网络用户）、聚合：（用来画第十章的类图，图要看教材上的（会挑一部分考，不能多答））

类图：p246；

ATM系统网络拓扑图（ATM机上可以使用现金兑换卡，可以访问用户的多个账户，功能是取钱和查询；）



ATM系统需求陈述

某银行拟开发一个自动取款机(ATM)系统

是由一个自动取款机、中央计算机、分行计算机及柜员终端组成的网络系统

➢ ATM和中央计算机由总行投资购买

➢ 总行拥有多台ATM，分别设在全市各主要街道上

➢ 分行负责提供分行计算机和柜员终端

➢ 柜员终端设在分行营业厅及下属的各营业所内

✓银行柜员——使用柜员终端，处理储蓄事务

✓储户——向自己的某个帐户内存款或开新帐户

从自己的帐户取款

拥有银行帐户的储户有权申请领取现金兑换卡。

✓现金兑换卡-----用它通过ATM访问自己的帐户

仅限于现金兑换卡在ATM机上提取现金（即取款）

查询自己帐户的信息（例如，某个指定帐户上的余额）。

将来可能还要求使用ATM办理转帐、存款等事务。

ATM系统需求陈述--现金兑换卡

◇ 现金兑换卡

分行代码，唯一标识总行下属的一个分行

卡号，确定了这张卡可以访问哪些帐户

◇ 一张卡

可以访问储户的若干个帐户

但是不一定能访问这个储户的全部帐户

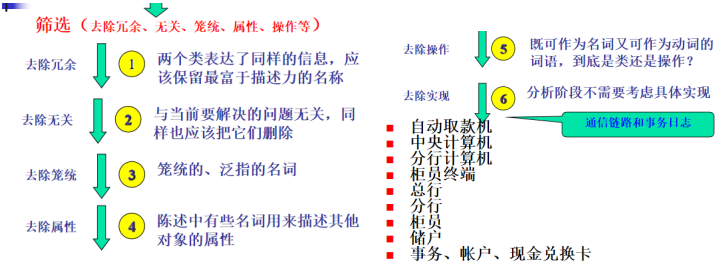
◇ 每张现金兑换卡
仅属于一个储户所有，但是，同一张卡可能有多个副本
必须考虑同时在若干台ATM上使用同样的现金兑换卡的可能性。即系统应该可以处理并发的访问。

ATM系统需求陈述—操作

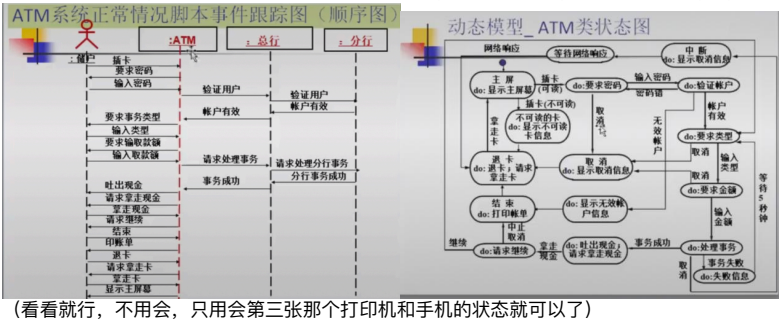
现金兑换卡插入ATM， ATM与用户交互，获取有关这次事务的信息，并与中央计算机交换关于事务的信息
Step1, ATM要求用户输入密码
Step2, ATM把密码传给中央计算机，请求核对并处理
Step3, 中央计算机分行代码确定这次事务与分行的对应关系，并委托相应分行计算机验证用户密码
Step4, 如果密码是正确的， ATM就要求用户选择事务类型（取款、查询）（查询中央计算机可以直接处理）
Step5, 当选择取款时，ATM请求用户输入取款额
Step6, ATM从出口吐出现金，并打印帐单交用户

以需求陈述为依据：

A. 把陈述中的名词作为类或对象的候选者（包含类的名字、类的属性、（类的方法考试不考））
B. 把形容词和名词词组作为属性的线索
C. 从动词或动词短语得到类与对象关联



限定可以使原来一对多的变成一对一
1：*表示一对多；聚合用黑色菱形（表示意思：如：如果总行没有了，那么它的下级聚合也就没有了）；泛化（远程事物和柜员事物可以泛化成事物；ATM机和柜台终端可以泛化成输入站）
注意：课本上对于现金兑换卡加了一个卡权限，上面放的卡号密码，作为一个存放数据的东西，现金兑换卡访问账户需要卡权限才能访问，即把它分化成了卡权限和现金兑换卡两个类
方法从状态图（一张状态图描绘的是一个对象）和ＤＦＤ（用事件跟踪图）中抽取



（看看就行，不用会，只用会第三张那个打印机和手机的状态就可以了）

面向对象分析小结

针对每个类建立的动态模型，描述了类实例的生命周期或运行周期
状态转换驱使行为发生，这些行为在数据流图中被影射为处理，在例图中被映射为处理，在例图中被映射为用例，它们同时与类中的服务相对应。
功能模型中的处理（或用例）对应于对象模型中的类所提供的服务

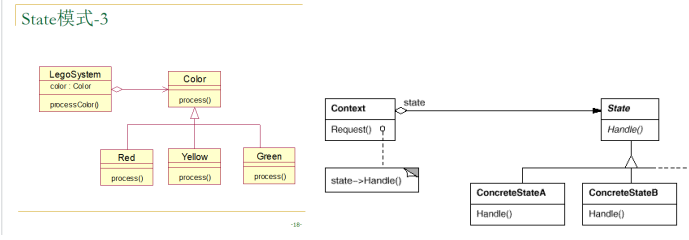
数据流图中的数据存储以及数据的源点 / 终点，通常是对象模型中的对象
数据流图中的数据流往往是对象模型中对象的属性值，也可能是整个对象
用例图中的行为者可能是对象模型中的对象
功能模型中的处理（或用例）可能产生动态模型中的事件
对象模型描述了数据流图中的数据流、数据存储以及数据源点 / 终点的结构

第十一章　面向对象设计

面向对象的设计模式（比较成功的模式，在设计软件时可以套用）
创建型模式
类模式--将对象的部分创建工作延迟到子类
对象模式--将它延迟到另一个对象中
结构型模式
类模式--使用继承机制来组合类（子类父类）
对象模式--描述了对象间的组装方式
行为型模式
类模式--使用继承描述算法与控制法
对象模式--描述一组对象怎样协作完成单个对象无法完成的工作
只讲两种：Adapter（考）/State（考）（背描述和结构图）
State：

```
LegoSystem::ProcessColor(){
  switch (color) {
    case RED:
      RedProcess();
      break;
    case GREEN:
      GreenProcess();
      break;
    case YELLOW:
      YellowProcess();
      break;
    }
};
```

弊端：当需要增加属性时，实际上改掉了类，需要重新编译，我们就觉得很不好。
State目的：允许一个对象在其内部状态改变时改变其行为

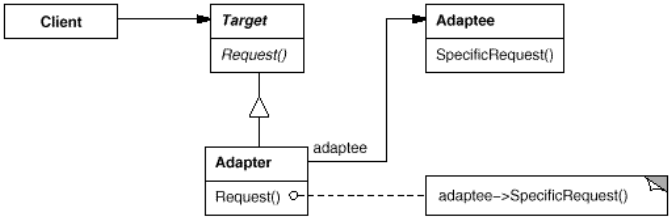



```
LegoSystem::ProcessColor(){
   Color->Process();
};
CColor{
public:
   virtual void Process();
}
void CRed::Process(){
   RedProcess();
};
void CGreen::Process(){
   GreenProcess();
};
void CYellow::Process(){
   YellowProcess();
};
```

```
//仅需要增加新的类，
//原有代码不需要任何变动
void CBlue::Process(){
   BlueProcess();
};
```

Adapter适配器模式（适配器就是一个类）（类似于加上一个转接头）
假设我们已经有一个软件系统，原来使用了一个第三方类库A。现在有一个新的第三方类库B，其功能等各方面都更加强大。我们希望用B来替换A，以改善我们的系统。但是B的接口与A不一样。那怎么办呢？
（Target目标对象类；Adapter适配器，调用另一个接口，作为转换器；Adaptee适配器，Adapter

需要接入）（这个和下边这个图背下来（箭头是泛化，Java不能有两个父类，所以把一个父类做成接口
））



第十三章软件项目管理

1.信息域特性（简答题）

功能点技术（基于软件功能的度量方法）定义了信息域的5个特性：
输入项数(Inp)：用户向软件输入的项数，这些输入给软件提供面向应用的数据。
输出项数(Out)：软件向用户输出的项数，它们向用户提供面向应用的信息，
查询数(Inq)：查询即是一次联机输入，它导致软件以联机输出方式产生某种即时响应。
主文件数(Maf)：逻辑主文件（即数据的一个逻辑组合，他可能是大型数据库的一部分或是一个独立的文件）的数目。
外部接口数(Inf)：机器可读的全部接口的数量（如磁盘或磁带上的数据文件），用这些接口把信息传送给另一个系统。

2.具体来说，软件配置管理主要有5项任务：标识、版本控制、变化控制、配置审计和报告。（简答题）（软件配置管理的目标是使变化更正确且更容易被适应，在必须变化时减少变化所需花费的工作量）

1. 标识软件配置中的对象

为了控制和管理软件配置项，必须单独命名每个配置项，然后用面向对象方法组织它们。可以标识出两类对象：
基本对象，是软件工程师在分析、设计、编码或测试过程中创建出来的“文本单元”。
聚集对象，是基本对象和其他聚集对象的集合。

2. 版本控制

版本控制联合使用规程和工具，以管理在软件工程过程中所创建的配置对象的不同版本。
借助于版本控制技术，用户能够通过选择适当的版本来指定软件系统的配置。

3. 变化控制

典型的变化控制过程如下：
首先评估该变化在技术方面的得失、可能产生的副作用、对其他配置对象和系统功能的整体影响以及估算出的修改成本。
为每个被批准的变化都生成一个“工程变化命令”。把要修改的对象从项目数据库中“提取”出来，进行修改并应用适当的SQA活动（SQA软件质量保证活动）。
最后，把修改后的对象“提交”进数据库，并用适当的版本控制机制创建该软件的下一个版本。

4. 配置审计

通常从下述两方面采取措施：
正式的技术复审，关注被修改后的配置对象的技术正确性。
软件配置审计，通过评估配置对象那些通常不在复审过程中考虑的特征，是对正式技术复审的补充。

5. 状态报告

书写配置状态报告回答下述问题：
发生了什么事?
谁做的这件事?
这件事是什么时候发生的?

它将影响哪些其他事物?
再看看6/7/8/9/10/11/12的选择题
OOA面向对象分析
OOD面向对象设计
PAD图一般指问题分析图，是详细设计中常用的图形工具