

第1章 概述

刘其成

ytliuqc@163.com

2018-9





- 教育部高等学校计算机科学与技术教学指导委员会制定的《软件工程（本科）专业规范》中，“软件设计与体系结构”已经作为一门核心课程单独列出，并有相应的知识单元和知识点。
- 本课程紧密结合软件工程专业规范，在充分考虑普通高等院校学生实际情况的基础上，覆盖规范中“软件设计与体系结构”课程要求的知识单元和知识点。



- 本课程特别考虑到普通高等学校本科学生的实际理解和接受能力。
- 与以往许多软件工程相关课程主要以理论为主不同，本课程突出实用性，将复杂的理论融于具体的实例和程序之中。
- 书中的实例都是经过精心设计的，程序代码都做了认真调试，可以直接运行，方便理解和使用。

参考资料



- (美)**Braude**著, 李仁发等译. 软件设计——从程序设计到体系结构. 电子工业出版社, **2007**
- 孙玉山, 刘旭东. 软件设计模式与体系结构. 高等教育出版社, **2013**
- 邵维忠, 杨芙清. 面向对象的系统设计. 第**2**版. 清华大学出版社, **2007**
- 麻志毅, 邵维忠. 面向对象方法基础教程. 高等教育出版社, **2004**
- 张友生. 软件体系结构原理、方法与实践 (第**2**版). 清华大学出版社, **2014**
- 阎宏. **Java**与模式. 电子工业出版社, **2002**
- 程杰. 大话设计模式. 清华大学出版社, **2007**

课程简介



- 必修课
- 考试课(闭卷)+平时(作业、实验、考勤.....)
- 上课**32**学时 / 上机**16**学时
- 课程设计
- 上课时间
- 课代表



- 任课教师简介（学院网站）
 - 刘其成
 - 毕远伟

1.1 软件工程方法学



- 软件生存周期
 - 软件从功能确定、设计，到开发成功投入使用，并在使用中不断地修改、增补和完善，直到停止该软件的使用的全过程。
- 软件工程方法学
 - 软件生命周期全过程中使用的一整套技术方法的集合
- 软件工程方法学三要素
 - 方法、工具和过程。
- 常用的软件开发方法
 - 结构化方法和面向对象方法

goto例

面条式代码



```
#include <stdio.h>
#include<math.h>
void main()
{
    float a=0,b=0,c=0,s=0,p=0;
    char slct=100;
    printf("请输入需要计算面积的三角形的三边，用英文逗号“,”隔开：\n");
    loop:scanf("%f,%f,%f",&a,&b,&c);
    {if(a+b>c)
        {if(a+c>b)
            goto shabby;
        else printf("三边长度无法组成三角形，请重新输入:\n");
        goto loop;}
    else printf("三边长度无法组成三角形，请重新输入:\n");
    goto loop;}

shabby:p=(a+b+c)/2;
s=sqrt(p*(p-a)*(p-b)*(p-c));
printf("客官您要的三角形的面积是%f\n",s);
scanf("%c",&slct);
printf("需要再算一组吗亲？(y/n):");
replay:scanf("%c",&slct);
    if(slct==121)
        {printf("那么，请再次输入本次三角形的三边，用英文逗号“,”隔开：\n");
        goto loop;}
    else {if(slct==110)
        {printf("感谢您的使用，再见！\n");
        goto result;}
        else
        {printf("您未做出选择！\n请重新选择：(y/n)");
        scanf("%c",&slct);}
        goto replay;}
    result:;
}
```





```
10 i = 0
20 i = i + 1
30 PRINT i; " squared = "; i * i
40 IF i >= 10 THEN GOTO 60
50 GOTO 20
60 PRINT "Program Completed."
70 END
```

```
Public Sub Main()
    For i As Integer = 1 To 10
        Console.WriteLine("{0} squared = {1}", i, i ^ 2)
    Next
    Console.WriteLine("Program Completed.")
End Sub
```



- 一开始用goto语句的时候，感觉还挺不错的，能随意跳转。
- 但当在写一个程序，其中用上了好几条goto语句，最后把自己都给弄糊涂了。
- 最后代码看上去会像一碗堆起的意大利面条，通常称这样的代码为“面条式(Spaghetti)”代码。
- “面条式”代码缺点
 - 可读性差，很难确定它到底完成什么工作。
 - 实际上“面条式”代码很少是高效的，因为太难确定它到底完成什么功能，从而也很难使用更好的算法来改进。因此，最后导致代码效率更低。



```
if ( ) {  
    .....  
}else{  
    .....  
}
```

```
for(){  
    .....  
}
```

```
class c1{  
    void f1(){  
        if ( ) {  
            .....  
        }else{  
            .....  
        }  
    }  
}
```

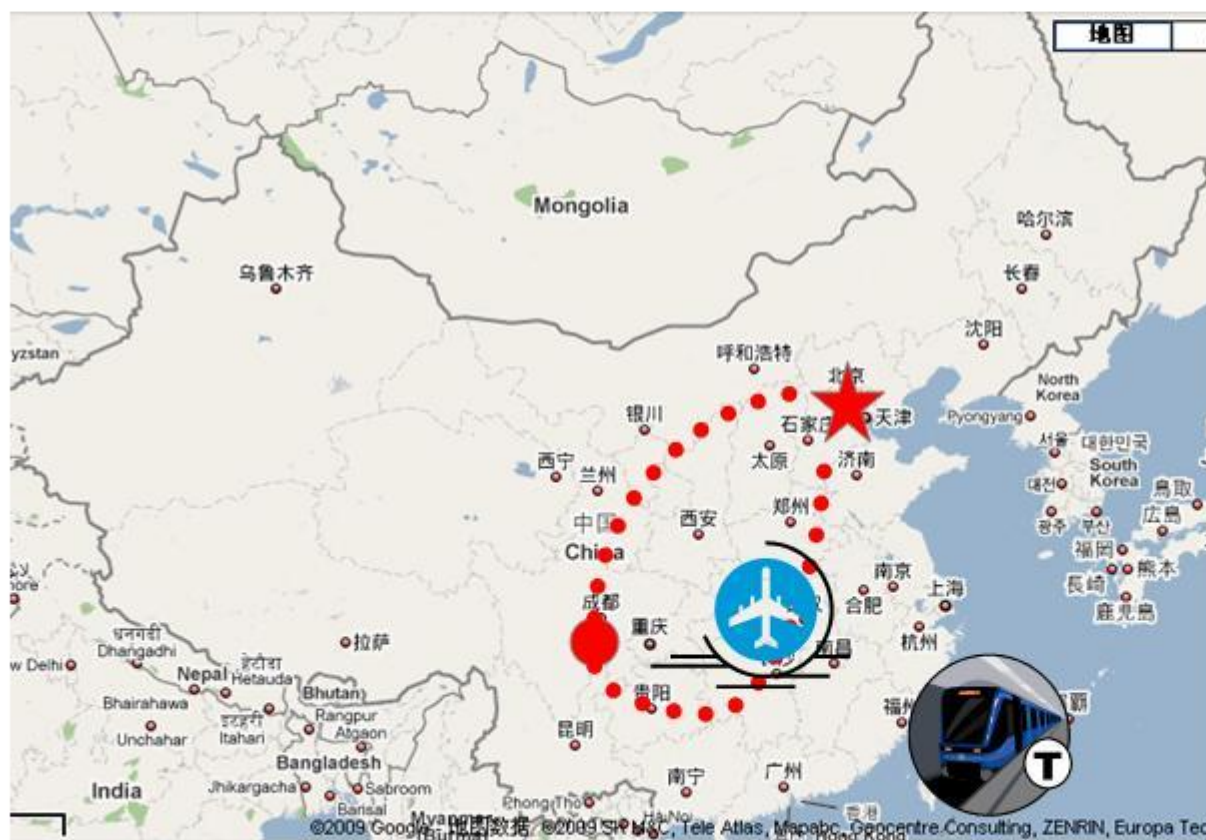


- 软件工程方法学

- 传统方法学 SA+ SD+ SP

- 面向对象方法学 OOA+ OOD +OOP

开发软件的方法不同





1.1.1 结构化方法

- 几个主要阶段
 - 分析、设计、编码、测试、运行和维护。

问题域

需求分析

分析与设计鸿沟

总体设计

详细设计

编程

测试

维护

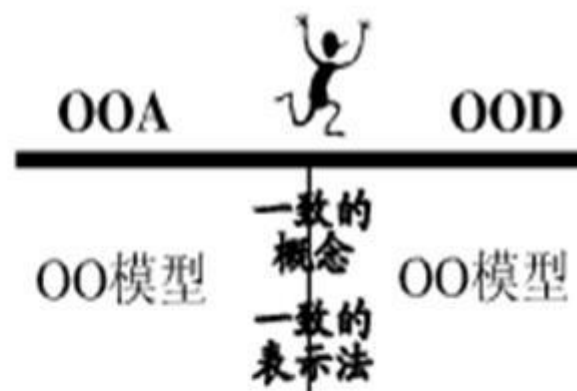
ATM例



● 结构化方法分析与设计之间的鸿沟



传统方法分析与设计之间的鸿沟



面向对象的分析与设计之间不存在鸿沟

软件体系结构



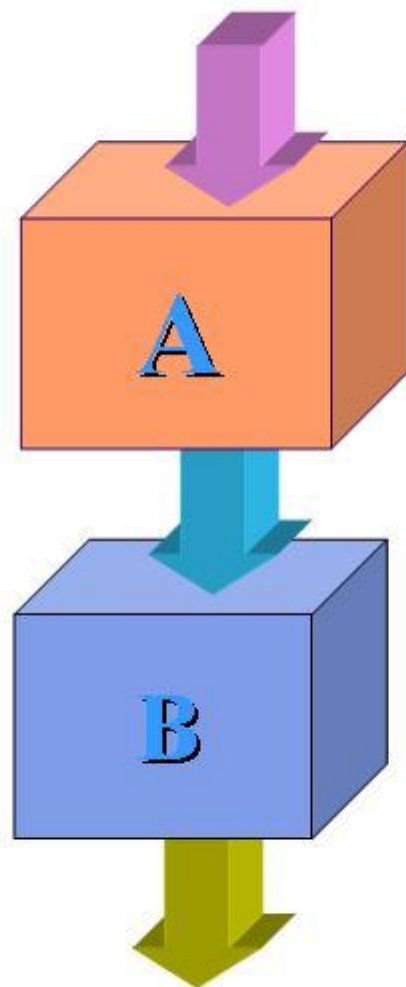
- 软件总是有体系结构的，不存在没有体系结构的软件。
- 可以把软件比作一座楼房，具有基础、主体和装饰：
 - 基础设施软件是操作系统
 - 主体应用程序实现计算逻辑
 - 用户界面程序方便用户使用
- 从细节上来看，每一个程序也是有结构的。
 - 早期的结构化技术将软件系统分成许多模块，模块间相互作用，自然地形成了体系结构。
 - 但是采用结构化技术开发的软件，程序规模不大，采用自顶向下、逐步求精，并注意模块的耦合性就可得到相对良好的结构，所以并未特别研究软件体系结构。



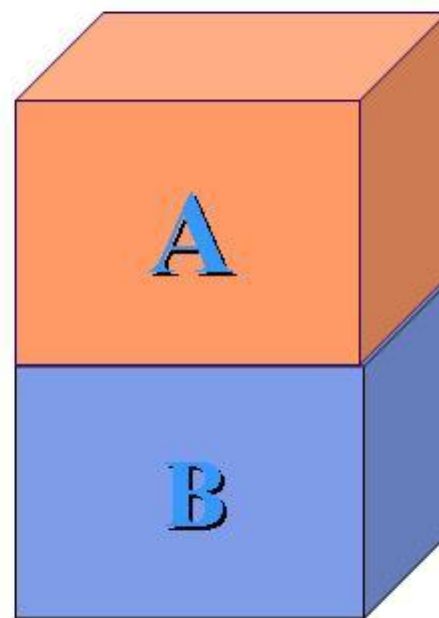
- 三大结构
 - 顺序结构
 - 从头到尾一次执行每一个语句
 - 分支结构
 - 根据不同的条件执行不同的语句或者语句体
 - 循环结构
 - 重复执行语句或者语句体，达到重复执行一类操作的目的

三大结构

- 顺序结构



传统流程图



N-S图

三大结构



例：根据给定半径求圆的周长。

```
#include "stdio.h"
```

```
main( )
```

```
{
```

```
    int radius;
```

```
    float circle;
```

```
    radius=4;
```

```
    circle=2*3.141592*radius;
```

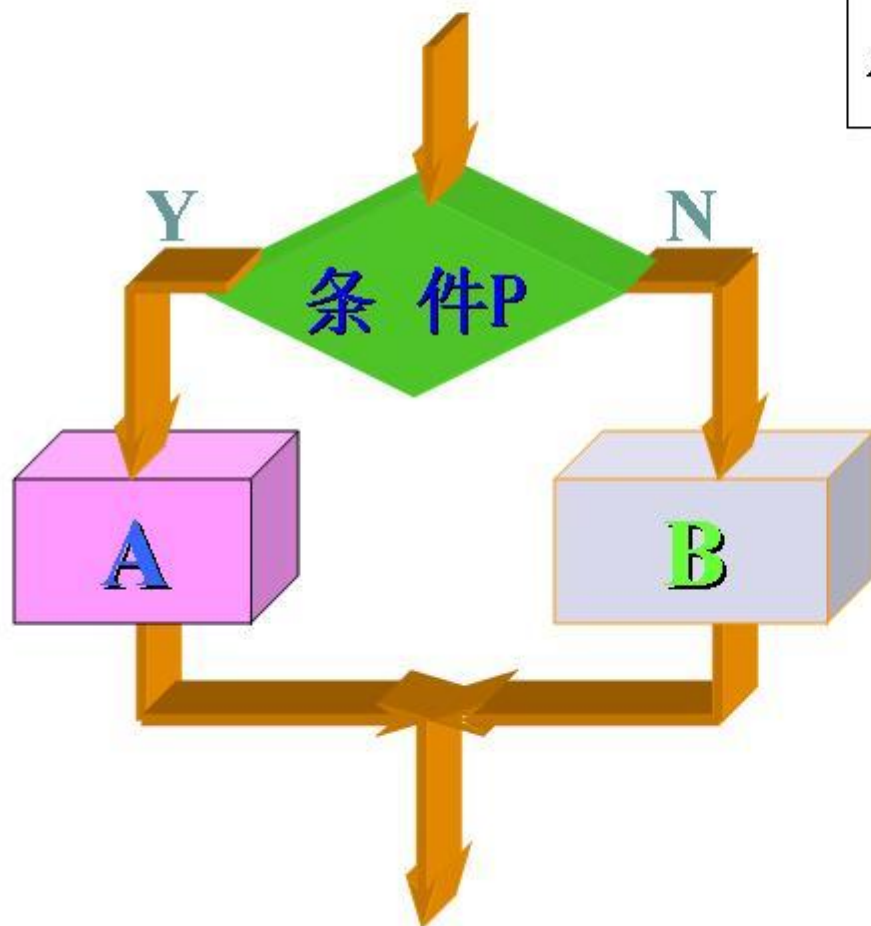
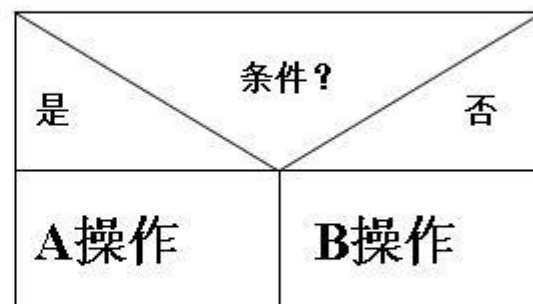
```
    printf("圆周长=%f\n",circle);
```

```
}
```

三大结构

- 分支结构（选择结构）

分支结构



如果 成绩 <60 那么

通知补考

否则

告知你考试成绩

三大结构



例：将两个数由小到大排列

```
#include "stdio.h"
```

```
main()
```

```
{
```

```
    int a,b,t;
```

```
    scanf("%d%d",&a,&b);
```

```
    printf("a=%d,b=%d\n",a,b);
```

```
    if(a>b)
```

```
    { t=a;
```

```
      a=b;
```

```
      b=t;
```

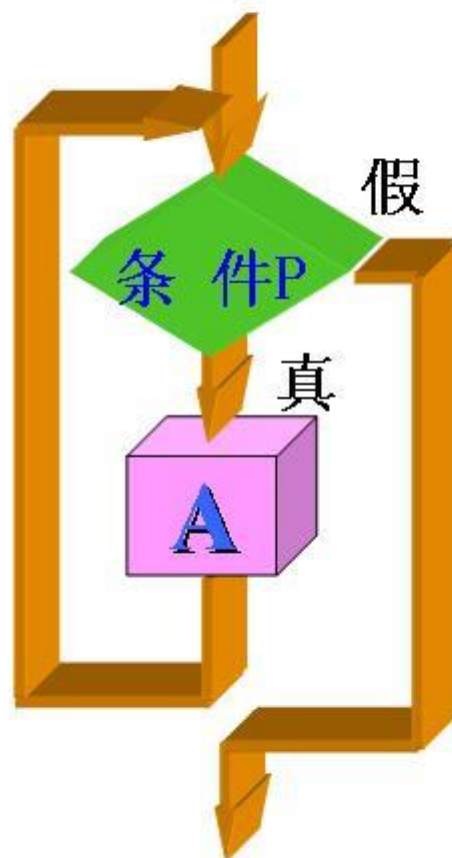
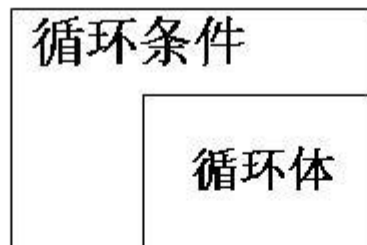
```
    }
```

```
    printf("a=%d,b=%d\n",a,b);
```

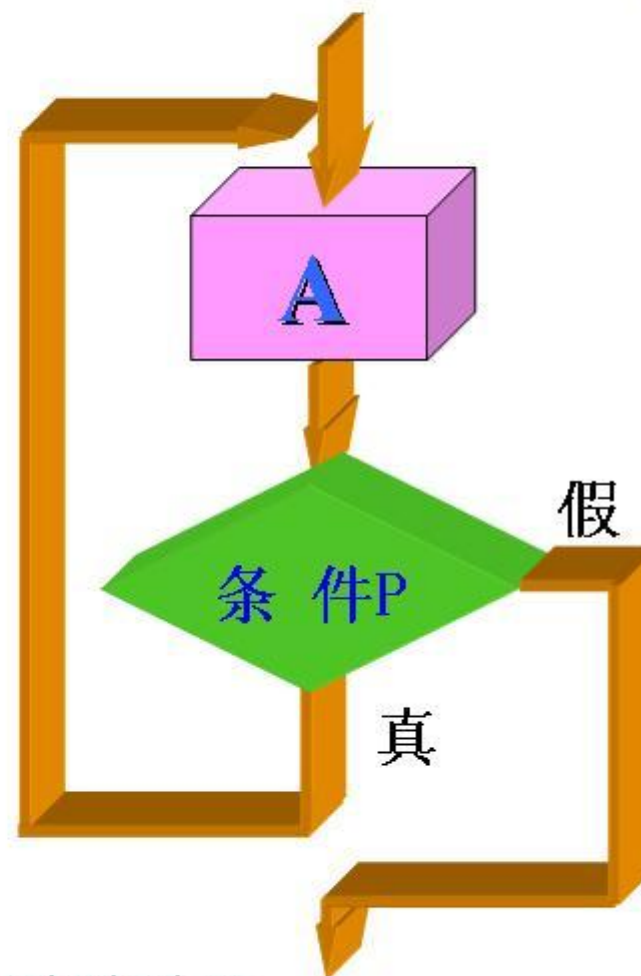
```
}
```


三大结构

- 循环结构



当型循环



直到型循环

三大结构



例：求**1**到**100**之间自然数的和。

```
#include "stdio.h"
```

```
main( )
```

```
{
```

```
    int i,sum;
```

```
    sum=0;
```

```
    for(i=1;i<=100;i++)
```

```
        sum=sum+i;
```

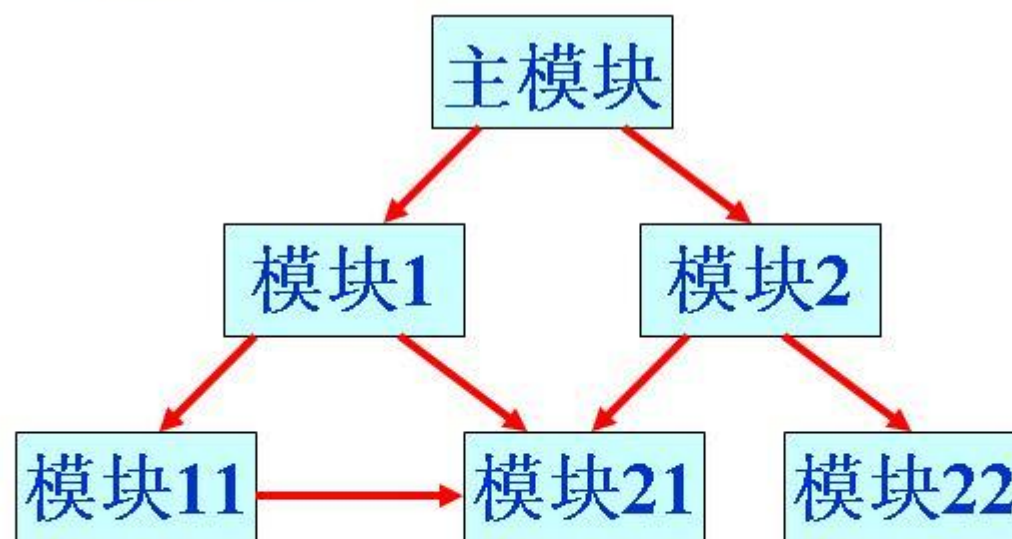
```
    printf("sum=%d\n",sum);
```

```
}
```



模块化程序结构

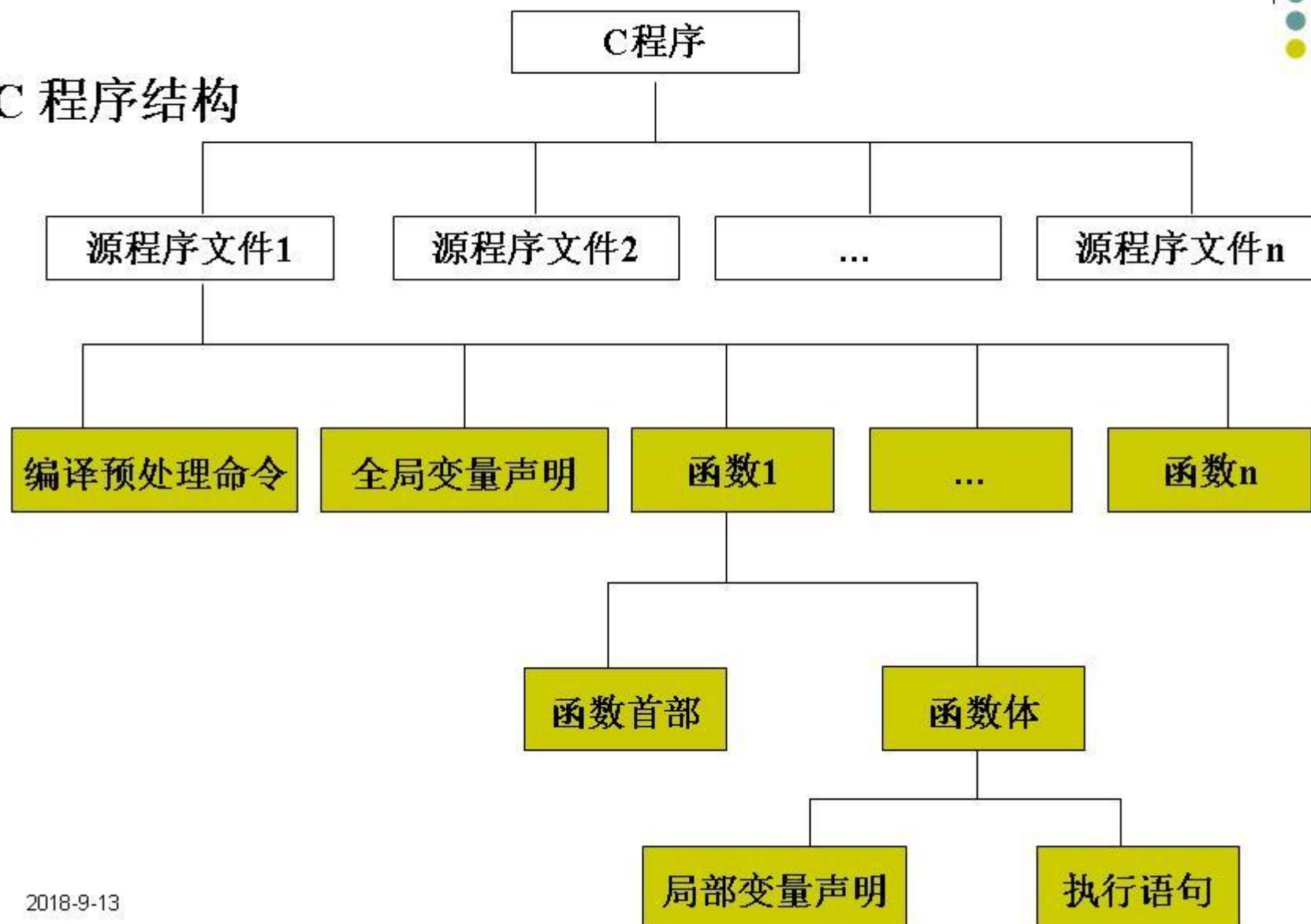
一个较大的应用程序应分为若干个程序模块，每一个模块用来实现一个特定的功能。所有的高级语言都有子程序概念，用子程序实现模块的功能。




2. 结构化程序设计方法



C 程序结构





```
void main()  
{ int n=3;  
  printf ("%d\n", sub1(n));  
}
```

程序输出结果：
9

函数调用可以嵌套

函数定义不可嵌套

```
sub1(int n)  
{ int i,a=0;  
  for (i=n; i>0; i--)  
    a+=sub2(i);  
  return a;  
}
```

```
sub2(int n)  
{  
  return n+1;  
}
```

1.1.2 面向对象方法



- 面向对象的软件工程仍采用结构化软件工程某些成熟思想和方法，在软件开发过程中仍采用分析、设计、编程、测试等技术，但在构造系统的思想方法上进行了改进。

问题域
面向对象分析
面向对象设计
面向对象编程
面向对象测试
维护

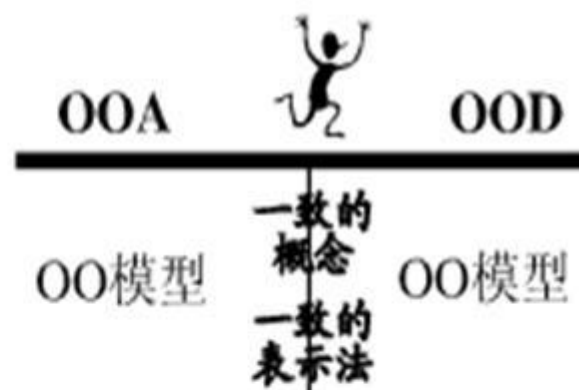
ATM例



- 采用一致的概念和表示法
——不存在分析与设计之间的鸿沟



传统方法分析与设计之间的鸿沟



面向对象的分析与设计之间不存在鸿沟



- 面向对象软件工程强调以问题域中的事物为中心来考虑问题，根据这些事物的本质特征，抽象地表示为对象，作为系统的基本构成单位。
- 因此，面向对象软件工程可以使软件系统直接地映射问题域，使软件系统保持问题域中事物及其相互关系的本来面貌。



- 面向对象起源于哲学观点。
- “面向对象”是一种认识客观世界的世界观，这种世界观将客观世界看成是由许多不同种类的对象构成的，每个对象有自己的内部状态和运动规律，不同对象之间的相互联系、相互作用就构成了完整的客观世界。
- 万事万物都是对象。





- 在客观世界中，有一些对象具有相同的特征，将他们称之为同类对象。于是出现了类的概念。



类的作用：(1) 分类：用于区分不同事物。

(2) 是创建对象的模板：产生对象。





“面向对象”运用到软件中

——面向对象软件工程（OOA/OOD/OOP）

- 利用这种“面向对象”认知的世界观来进行软件开发。
- 程序由类和对象组成。

```
class Tree {  
    .....  
    .....  
}
```

```
Tree tree1 = new Tree();  
Tree tree2 = new Tree();  
    .....
```

UML基本元素符号及其JAVA实现

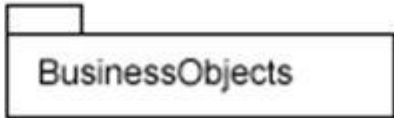


- 1、类（Classes）
- 类名、属性、方法
- 属性和操作之前可附加一个可见性修饰符。加号（+）表示具有公共可见性。减号（-）表示私有可见性。**#**号表示受保护的可见性。省略这些修饰符表示具有**package**（包）级别的可见性。
- 如果属性或操作具有下划线，表明它是静态的。

Java	UML
<pre>public class Employee { private int empID; public double calcSalary() { ... } }</pre>	<pre>classDiagram class Employee { -empID:int +calcSalary():double }</pre>

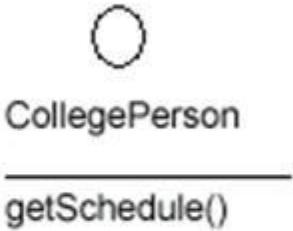


- **2、包（Package）**
- 包是一种常规用途的组合机制。主要用于对模型进行组织。

Java	UML
<pre>package BusinessObjects; public class Employee { }</pre>	



- 3、接口（Interface）
- 接口是一系列操作的集合，它指定了一个类所提供的服务。接口既可用下面的那个图标来表示（上面一个圆圈符号，圆圈符号下面是接口名，中间是直线，直线下面是方法名），也可由附加了<<interface>>的一个标准类来表示。

Java	UML
<pre>public interface CollegePerson { public Schedule getSchedule(); }</pre>	

UML关系及其JAVA实现



- 1、依赖（**Dependency**）
- 实体间“使用”关系。
- 对不在实例作用域内的一个类或对象的任何类型的引用。例如一个局部变量对通过方法调用而获得的一个对象的引用(如下所示)，或对一个类的静态方法的引用(同时不存在那个类的一个实例)
- 也可表示包和包间的关系。包中含有类，可根据包中的各个类之间的关系，表示出包和包的关系。

Java	UML
<pre>public class Employee { public void calcSalary(CalculatorStrategy { - } }</pre>	<pre>classDiagram Employee ..> Calculator</pre>



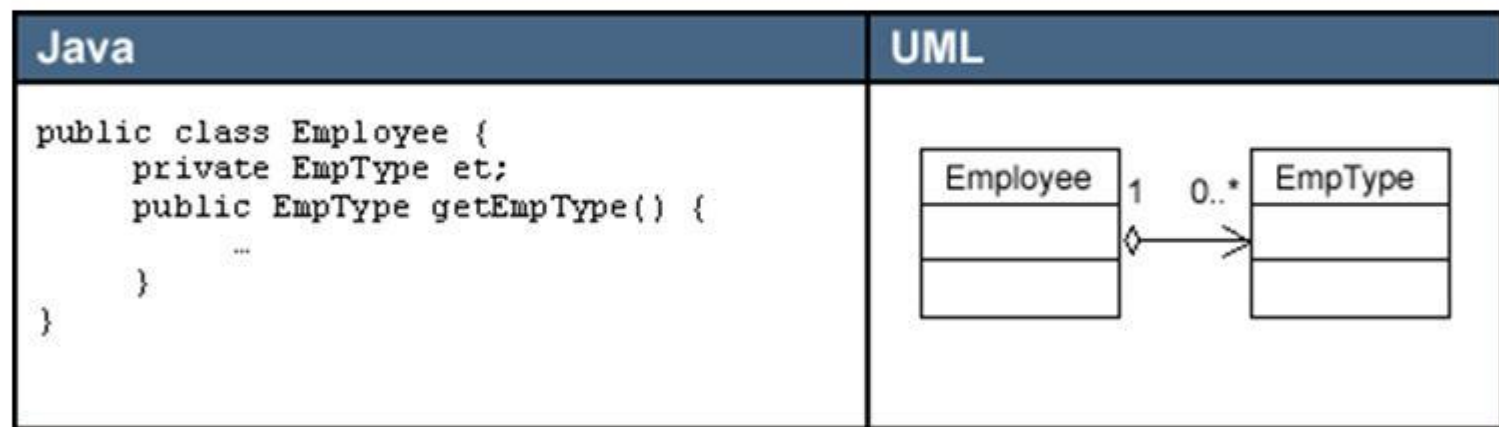
- **2、关联（Association）**
- 对象是相互连接的。一个实例作用域的变量。
- 箭头可选，指定导航。没有，双向导航。
- 可附加其他修饰符。多重性(Multiplicity)修饰符表示实例间的关系。在下图中，**Employee**可以有**0个或更多的TimeCard**对象；但是，每个**TimeCard**只从属于单独一个**Employee**。

Java	UML
<pre>public class Employee { private TimeCard _tc; public void maintainTimeCard() { ... } }</pre>	<pre>classDiagram Employee "1" --> "0..*" TimeCard</pre>



大雁与雁群的这种关系就可以称之为聚合

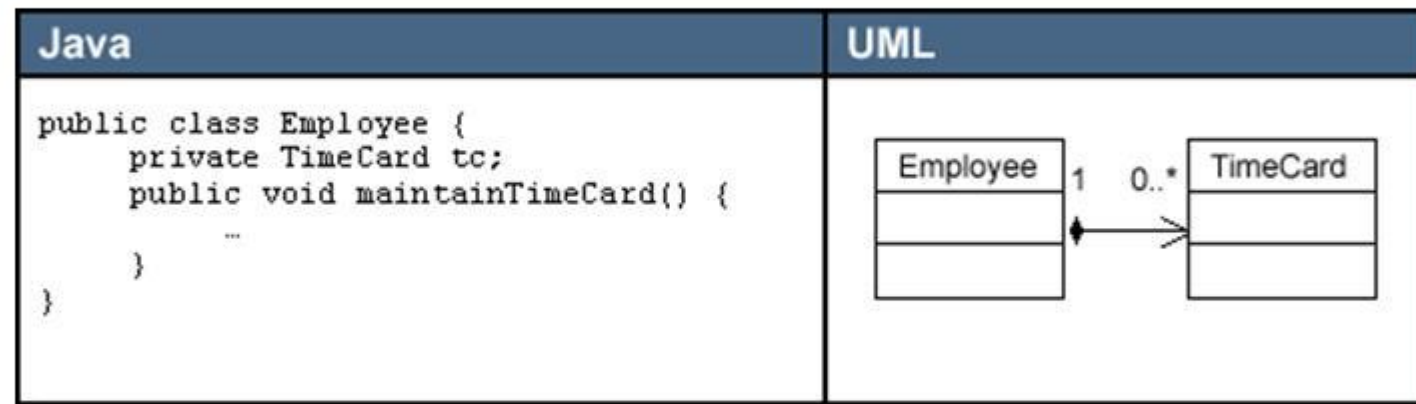
- 3、聚合（Aggregation）
- 关联的一种形式，代表两个类之间的整体/局部关系。聚合指整体在概念上处于比局部更高的一个级别，而关联指两个类在概念上位于相同的级别。聚合也转换成Java中的一个实例作用域变量。
- 关联和聚合的区别纯粹是概念上的，而且严格反映在语义上。聚合只能是一种单向关系。





大雁与雁翅的关系就叫做合成/组合

- **4、合成/组合 (Composition)**
- 聚合的一种特殊形式，指“局部”在“整体”内部的生存期职责。所以，虽然局部不一定要随整体的销毁而被销毁，但整体要么负责保持局部的存活状态，要么负责将其销毁。
- 局部不可与其他整体共享。但是，整体可将所有权转交给另一个对象，后者随即将承担生存期职责。





- **5、泛化/继承（Generalization）**
- 泛化表示一个更泛化的元素和一个更具体的元素之间的关系。
- 泛化是用于对继承进行建模的**UML**元素。在**Java**中，用**extends**关键字来直接表示这种关系。

Java	UML
<pre>public abstract class Employee { } public class Professor extends Employee { }</pre>	<pre>classDiagram Employee < -- Professor</pre> <p>The UML diagram shows two class boxes. The box on the left is labeled 'Employee' and the box on the right is labeled 'Professor'. A solid line with an open triangular arrowhead points from the 'Professor' box to the 'Employee' box, indicating that 'Professor' is a specialization of 'Employee'.</p>



- 6、实现（**Realization**）
- 实例关系指定两个实体之间的一个合同。换言之，一个实体定义一个合同，而另一个实体保证履行该合同。对**Java**应用程序进行建模时，实现关系可直接用**implements**关键字来表示。

Java	UML
<pre>public interface CollegePerson { } public class Professor implements CollegePers }</pre>	

软件体系结构



- 软件工程从传统的结构化软件工程进入到了现代的面向对象软件工程后，需要进一步研究整个软件系统的体系结构
- 寻求质量最好、建构最快、成本最低的构造过程



- 在引入了软件体系结构的软件开发之后，应用系统的构造过程变为面向对象分析、软件体系结构、面向对象设计、面向对象编程、面向对象测试和面向对象维护
- 可以认为软件体系结构架起了面向对象分析和面向对象设计之间的一座桥梁

1.2 软件设计与体系结构



- 1. 软件设计

- 软件设计形成一套文档，根据这些文档，程序员能够完整地设计出应用程序。
- 软件设计的结果可以使程序员不需要其他文档的帮助，就可以编写程序。
- 软件设计就像是建筑物和机械零件的图纸，建筑商和技术工人根据图纸就可以盖好相应的建筑物和加工好相应的零件。



● 2. 软件体系结构

- 结构化技术是以砖、瓦、预制梁等盖平房，而面向对象技术以整面墙、整间房、一层楼梯的预制件盖高楼大厦。
- 土木工程进入到了现代建筑学，怎样才能容易地构造体系结构？什么是合理的组件搭配？重要组件更改以后，如何保证整栋高楼不倒？不同的应用领域（学校、住宅、写字楼）分别需要什么样的组件？是否具有实用、美观、强度、造价合理的组件，从而使建造出来的建筑（即体系结构）更能满足用户的需求？



- 同样，软件工程也从传统的结构化软件工程进入到了现代的面向对象软件工程，需要进一步研究整个软件系统的体系结构，寻求质量最好、建构最快、成本最低的构造过程。



- 软件体系结构（**Architecture**）是设计抽象的进一步发展，满足了更方便地开发更大、更复杂的软件系统的需要。
- 随着软件系统规模越来越大、越来越复杂，对软件总体的系统结构设计和规格说明比起对计算的算法和数据结构的选择变得重要得多。

软件体系结构发展的阶段



- 第一个阶段是“无体系结构”设计阶段，以汇编语言进行小规模应用程序开发为特征。
- 第二个阶段是萌芽阶段，出现了程序结构设计主题，主要特征是使用控制流图和数据流图。
- 第三个阶段是初期阶段，出现了以**UML**为典型代表的从不同侧面描述系统的结构模型。
- 第四个阶段是高级阶段，以描述系统的高层抽象结构为中心，不关心具体的建模细节，该阶段以**Kruchten**提出的“**4+1**”模型为标志。



- 解决好软件的质量、复用性和可维护性问题，是研究软件体系结构的根本目的。

思考题



- **1.** 是否存在没有体系结构的软件？采用结构化技术开发的软件是否具有体系结构？
- **2.** 软件设计的含义
- **3.** 软件体系结构的发展阶段
- **4.** 研究软件体系结构的根本目的



谢谢