复习

第1章　软件工程学概述
第3章　需求分析
第5章　总体设计
第6章　详细设计
第7章　实现
第8章　维护
第9章　面向对象方法学引论
第10章　面向对象分析
第11章　面向对象设计

第1章　软件工程学概述

◆软件危机

- ●定义：指在计算机软件的开发和维护过程中所遇到的一系列严重问题。
- ●典型表现：
  - Ø开发成本和进度的估计常常很不准确；
  - Ø用户对“已完成的”软件系统不满意；　“闭门造车”；
  - Ø 软件质量不可靠；
  - Ø软件常常是不可维护的；
  - Ø软件成本的比例逐年上升；
  - Ø软件产品“供不应求”；
- ●产生软件危机的原因
  - Ø软件的规模加大、复杂性提高、性能增强;
  - Ø软件是逻辑产品,尚未完全认识其本质和特点;
  - Ø缺乏系统的开发、维护大型软件项目的技术手段和管理方法；
  - Ø用户和软件开发人员的理解鸿沟；
  - Ø错误的认识和作法

第1章　软件工程学概述

- 软件工程
  - 定义
    - 软件工程是指导计算机软件开发和维护的一门工程学科。采用工程的概念、原理、技术和方法来开发与维护软件，把经过时间考验而证明正确的管理技术和当前能够得到的最好的技术方法结合起来，以经济地开发出高质量的软件并有效地维护它。
  - 本质特性
    - 软件工程的中心课题是控制复杂性
    - 和谐地合作是开发软件的关键
    - ……

第1章　软件工程学概述

- 软件工程的7条基本原理

1.用分阶段的生命周期计划严格管理

2.坚持进行阶段评审

3.实行严格的产品控制

4.采用现代程序设计技术

5.结果应能清楚地审查

6.开发小组的人员应该少而精

7.承认不断改进软件工程实践的必要性

第1章　软件工程学概述

- 软件生命周期是软件产品或系统一系列相关活动的全周期。
- 软件生命周期的3个大阶段

1. 软件定义: 确定软件开发总目标；确定工程的可行性；导出实现策略及系统功能；估计资源和成本，并且制定工程进度表。
   - 问题定义、可行性研究、需求分析
2. 软件开发: 具体设计和实现在前一个时期定义的软件
   - 总体设计、详细设计、编码和单元测试、综合测试
3. 软件维护: 使软件持久地满足用户的需要

第1章　软件工程学概述

- 生命周期中各阶段的任务
  1. 问题定义：　“要解决的问题是什么?”；　确定用户要求解决的性质、工程的目标和规模。
  2. 可行性研究：　“对于上一个阶段所确定的问题有行得通的解决办法吗?”，经济可行性、技术可行性、法律可行性、不同的方案
  3. 需求分析：　“为了解决这个问题，目标系统必须做什么”，　确定系统必须具有的功能和性能

  ，系统要求的运行环境，并且预测系统发展的前景。
  4. 总体设计（概要设计）：　“概括地说，应该怎样实现目标系统?”；　设计出实现目标系统的几种可能的方案。推荐一个最佳方案。
  5. 详细设计：　“应该怎样具体地实现这个系统呢?”；　设计出程序的详细规格说明。
  6. 编码和单元测试：　写出正确的容易理解、容易维护的程序模块；　仔细测试编写出的每一个模块。
  7. 综合测试：　集成测试和验收测试，现场测试或平行运行
  8. 软件维护：　使系统持久地满足用户的需要。包括：　改正性维护，适应性维护，完善性维护，预防性维护。

第1章　软件工程学概述

生命周期模型规定了把生命周期划分成哪些阶段及各个阶段的执行顺序，因此，也称为过程模型。

◆典型的过程模型

1.瀑布模型(Waterfall model)

2.快速原型开发模型(Rapid Prototyping model)

3.增量模型(Incremental model)

4.螺旋模型(Spiral model)

5.喷泉模型

6.其它模型

- 极限编程XP(eXtreme Programming)
- RUP(Rational Unified Process)
- 建造—修补模型(Build-and-fix model)

第1章　软件工程学概述

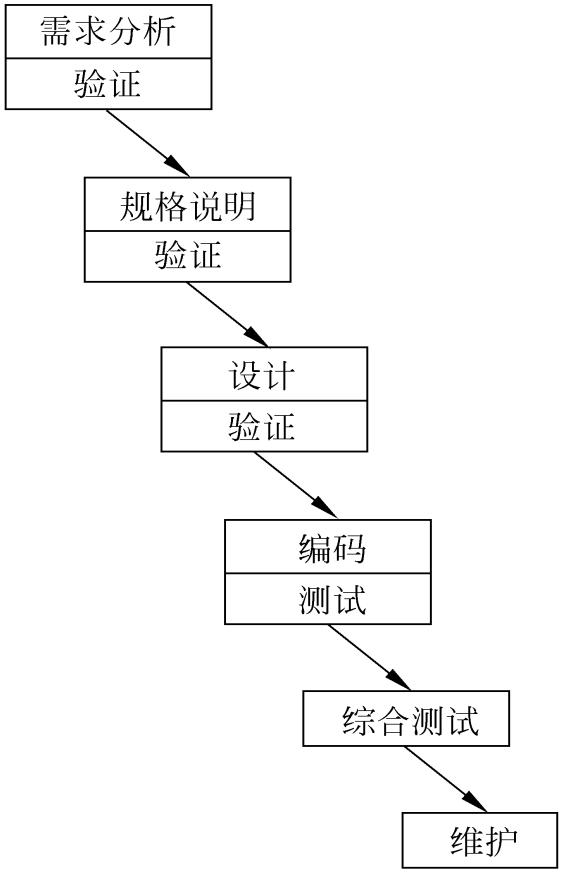
- 瀑布模型

特点

1. 阶段间具有顺序性和依赖性
2. 推迟实现的观点
   - 清楚地区分逻辑设计与物理设计，尽可能推迟程序的物理实现。
3. 质量保证的观点(文档驱动)
   - 每个阶段都必须完成规定的文档
   - 每个阶段结束前都要对所完成的文档进行评审

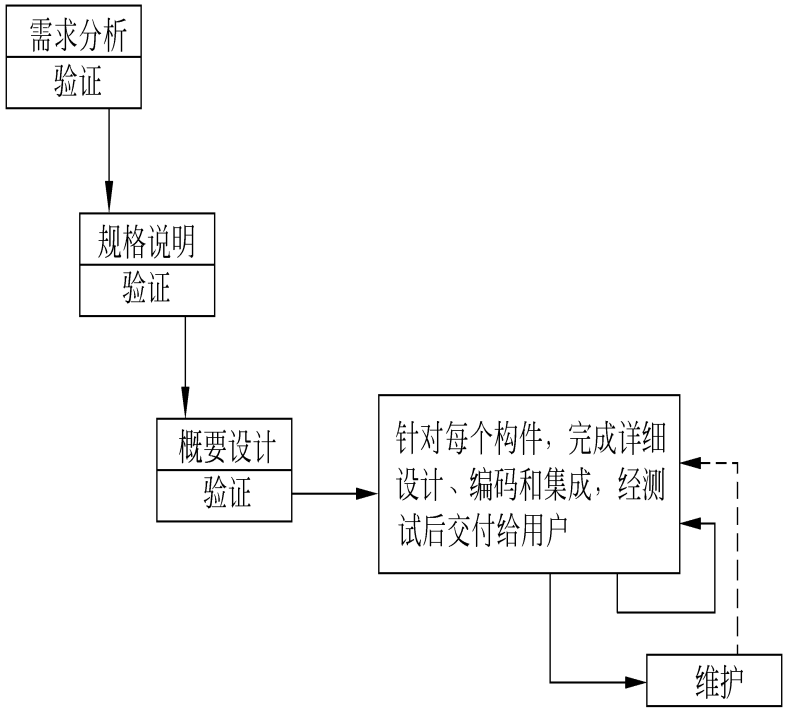
缺点

1. 开发过程一般不能逆转，否则代价太大。
2. 规格说明很难理解：“我知道这是按我的要求做的，但不是我想要的样子。”
3. 软件的实际情况必须到项目开发的后期客户才能看到。



第1章　软件工程学概述

- 增量模型
  - 把软件产品作为一系列增量构件来设计、编码、集成和测试。
  - 优点
    - 每个阶段交付一个可用的产品。
    - 减少一个全新产品给客户带来的心理上的影响。
    - 分阶段地交付产品不需要大的资金支出。
    - 需求经常变化，增量模型的灵活性使其具有更加优越的适用性。
  - 缺点
    - 需要一个开放的结构，方便构件的加入。



第1章　软件工程学概述

生命周期模型	优点	缺点
瀑布模型	文档驱动的有序方法	交付产品可能不符合客户的要求
快速原型模型	确保交付的产品符合客户的要求	还没有证明无懈可击
增量模型	增大投资的早期回报	要求开放的结构，可能退化为建造-修补模型
螺旋模型	结合上述所有模型的特性	只能用于大型的内部软件产品，开发者必须精通风险分析和风险排除

第3章　需求分析

- 需求分析的任务
  1. 准确地回答“系统必须做什么?”
  2. “分析软件需求和书写软件需求规格说明书”
- 软件需求
  - 用户解决问题或达到目标所需要的条件或能力；
  - 系统或系统部件要满足合同、标准、规范或其他正式规定文档所需具有的条件或能力
  - 反映上述两个定义中所描述的条件或能力的文档说明
  - 需求层次：业务需求→用户需求→功能与非功能需求
- 具体任务
  - 确定对系统的综合要求

    功能需求、性能需求、可靠性和可用性需求、出错处理需求、接口需求、约束、逆向需求、扩展需求
  - 分析系统的数据要求
  - 导出系统的逻辑模型
  - 修正系统开发计划

第3章　需求分析

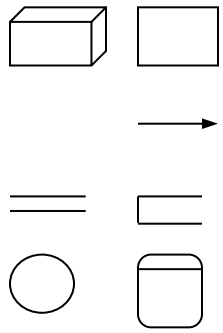
- 数据流图
  - 数据流图（DFD）符号

    数据源点/数据终点

    数据流

    数据存储

    加工/处理

  - 根据描述画数据流图
- 状态图
  - 通过描绘系统的状态及引起系统状态转换的事件，来表示系统的行为。
    - 状态
    - 事件
    - 行为
  - 画状态图

第5章 总体设计

- 总体设计的任务
  1. “概括地说，系统应该如何实现？”
  2. 系统划分:即确定组成系统的程序、文件、数据库、人工过程和文档等
  3. 设计软件的结构：即确定每个程序是由哪些模块组成，以及这些模块相互间的关系。

第5章 总体设计

- 设计原理

1. 模块化
   - 模块化的根据

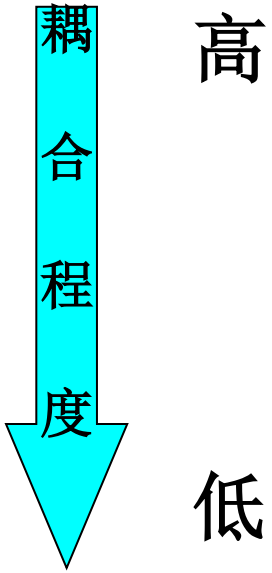
     C（P1＋P2）＞ C（P1）＋ C（P2）

     E（P1＋P2）＞ E（P1）＋ E（P2)
2. 抽象
   - 抽象就是抽出事物的本质特性而暂不考虑它们的细节
3. 逐步求精
   - 为了能集中精力解决主要问题而尽量推迟对问题细节的考虑。
4. 信息隐藏和局部化
   - 信息隐藏原理：应该这样设计和确定模块，使得一个模块内包含的信息(过程和数据)对于不需要这些信息的模块来说，是不能访问的。
   - 局部化: 把一些关系密切的软件元素物理地放得彼此靠近。
5. 模块独立
   - 每个模块完成一个相对独立的子功能，并且与其它模块间的接口简单。
   - 模块独立性的衡量标准
     - 模块内聚（Cohension）：模块内各元素交互的程度
     - 模块耦合（Coupling）：模块间交互程度

第5章 总体设计

◆ 耦合

1. 内容耦合
2. 共用耦合
3. 控制耦合
4. 印记（特征）耦合
5. 数据耦合

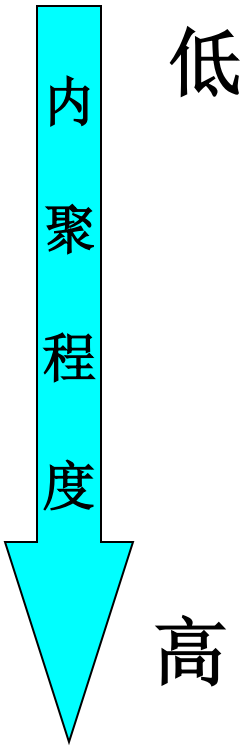


- 各种耦合的含义
- 设计时力争做到低耦合。应该采取的设计原则是：尽量使用数据耦合，少用控制耦合和特征耦合，限制公用耦合的范围，完全不用内容耦合。

第5章 总体设计

◆ 内聚

1. **偶然性内聚**
2. **逻辑性内聚**
3. **时间性内聚**
4. **过程性内聚**
5. **通信性内聚**
6. **顺序内聚**
7. **功能性内聚**
8. **信息性内聚**

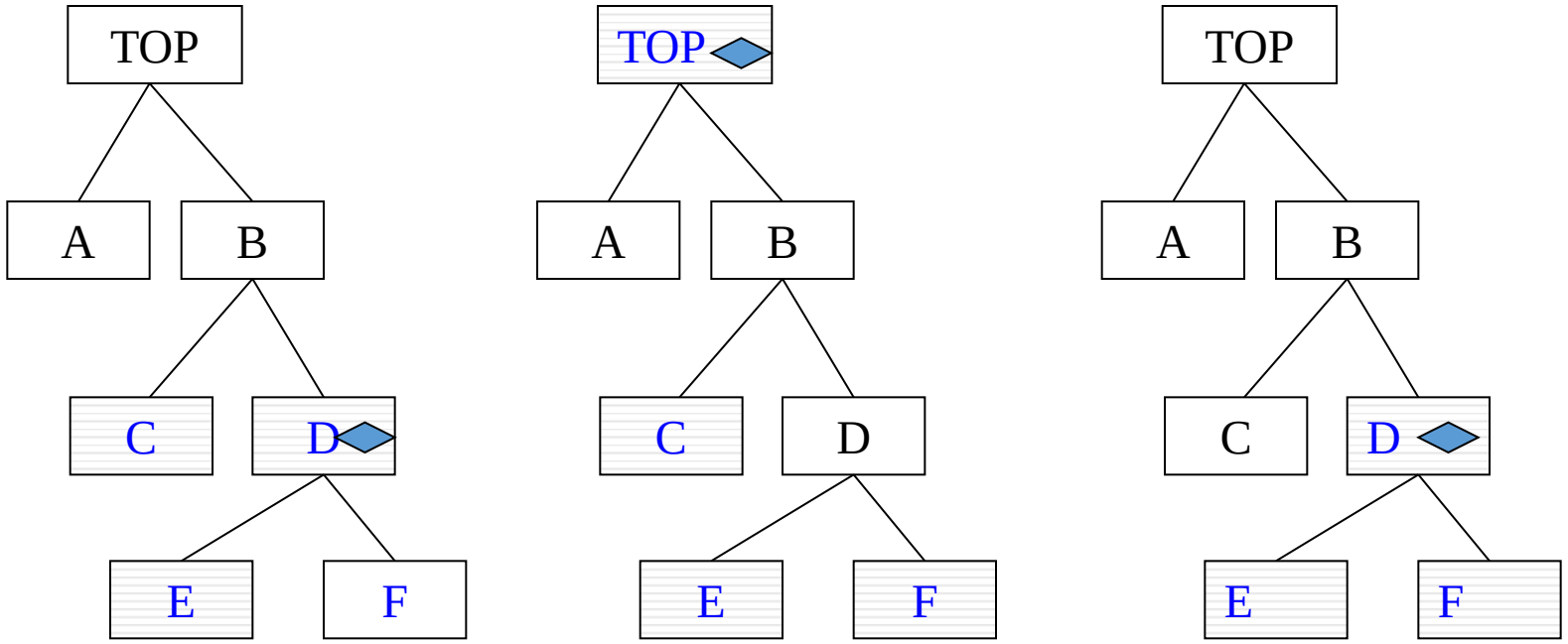


- 各种内聚的含义
- 设计时力争做到高内聚，并且能够辨认出低内聚的模块，通过修改设计提高模块的内聚程度。

第5章 总体设计

- 启发规则
  1. 改进软件结构提高模块独立性
  2. 模块规模应该适中
  3. 深度、宽度、扇出和扇入都应适当
  4. 模块的作用域应该在控制域之内
     - 模块的作用域：受该模块内一个判定影响的所有模块的集合。
     - 模块的控制域：模块本身以及所有直接或间接从属于它的模块的集合。
     - 所有受判定影响的模块应该都从属于做出判定的那个模块，最好局限于做出判定的那个模块本身及它的直属下级模块。
  5. 力争降低模块接口的复杂程度
  6. 设计单入口单出口的模块
  7. 模块功能应该可以预测

第5章 总体设计

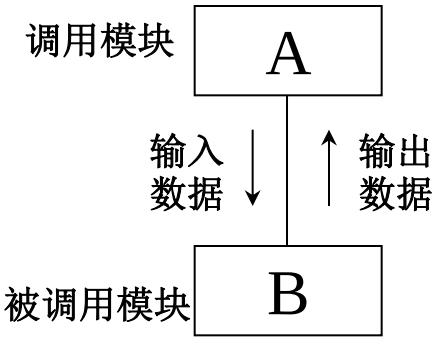


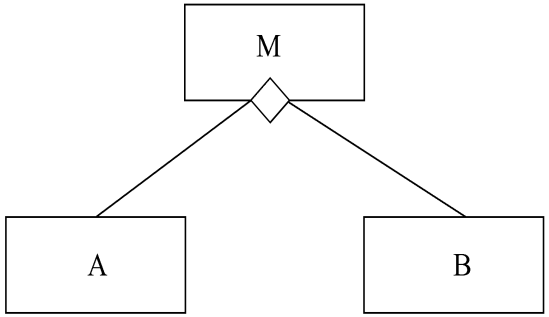
模块的判定作用范围

(a) 差的结构图； (b) 不理想的结构图； (c) 理想的结构图

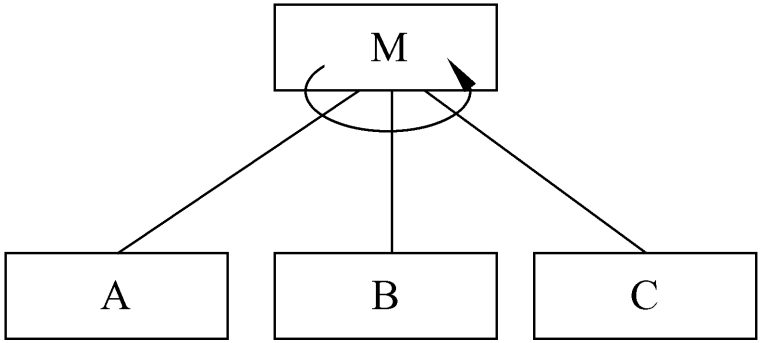
第5章 总体设计

- 结构图
  - 描述软件系统的模块层次结构，清楚地反映出程序中各模块之间的调用关系和数据传递。





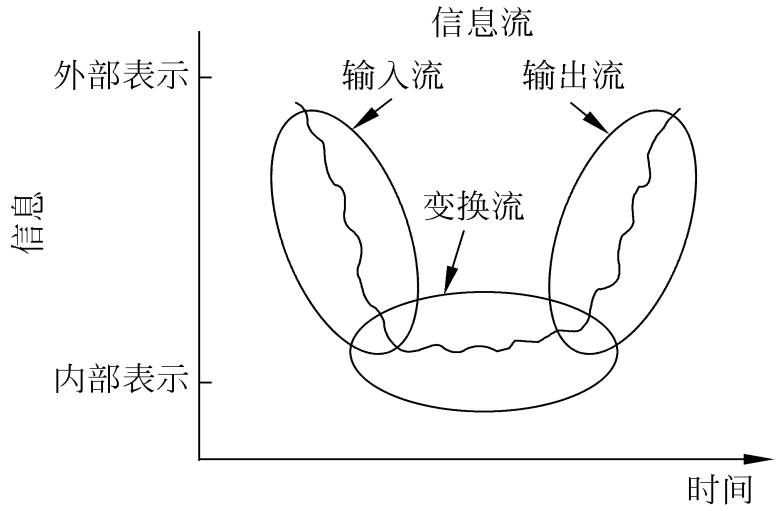
选择调用



循环调用

第5章 总体设计

- 面向数据流的设计方法
  - 基本思想： DFD → 结构图
  - 两种信息流类型
    - 变换流：掌握从变换流到初始结构图的转换
    - 事务流

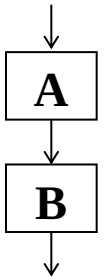


第6章 详细设计

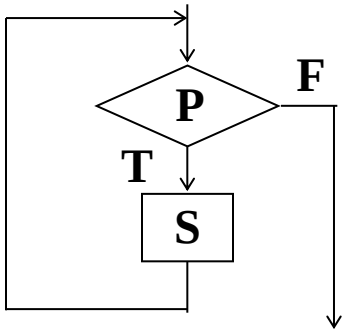
- 详细设计的任务
  - 不是具体地编写程序，而是设计程序的“蓝图”，确定每个模块的处理过程。
- 结构程序设计
  - 结构化定理
    - 任何单入口单出口的程序都可以由“顺序”、“选择”和“循环”三种基本结构实现。
  - 过程设计的工具（重点是画图）
    - 程序流程图
    - 盒图
    - PAD图
    - 判定表

第6章 详细设计

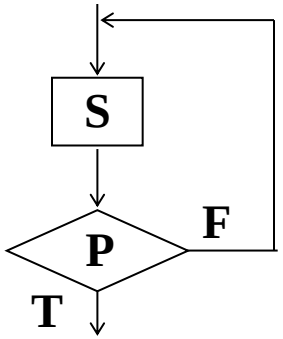
- 程序流程图的基本符号



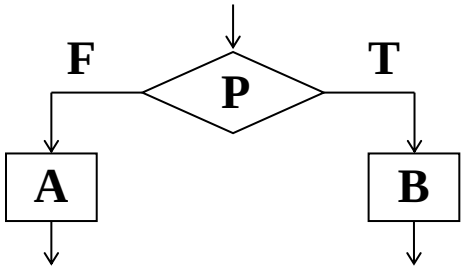
(1) 顺序结构



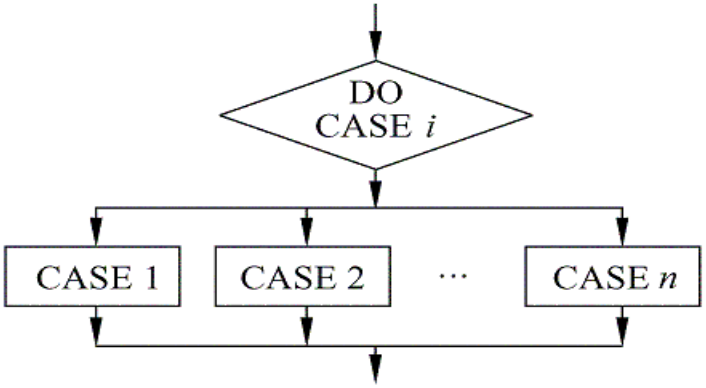
(3) 先判定型循环结构



(4) 后判定型循环结构



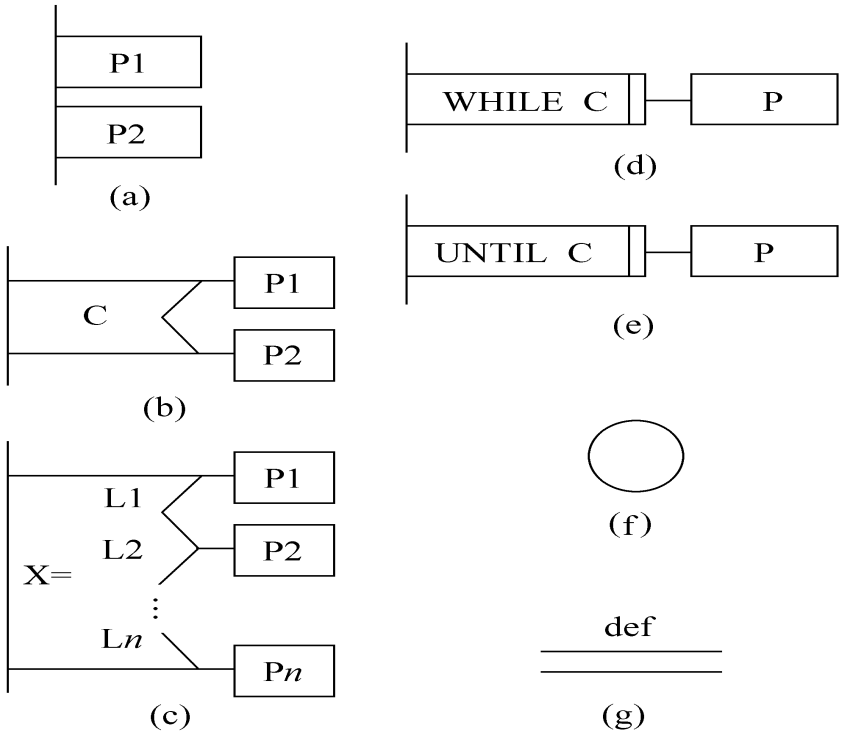
(2) 选择结构



(5) 多情况选择(case structure)

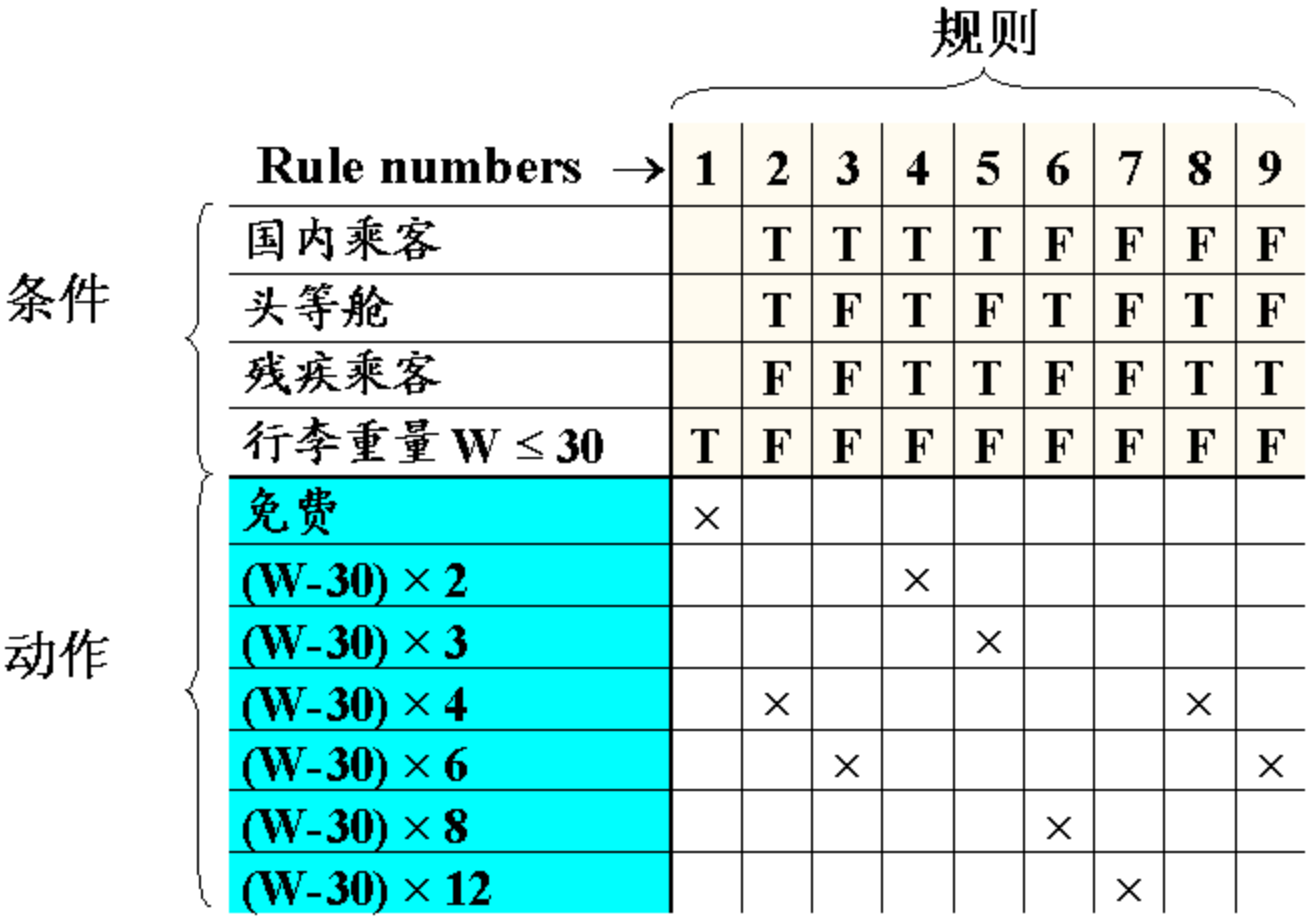
第6章 详细设计

- PAD图的基本符号



第6章 详细设计

- 判定表



		规则								
	Rule numbers →	1	2	3	4	5	6	7	8	9
条件	国内乘客		T	T	T	T	F	F	F	F
	头等舱		T	F	T	F	T	F	T	F
	残疾乘客		F	F	T	T	F	F	T	T
	行李重量 W ≤ 30	T	F	F	F	F	F	F	F	F
动作	免费	×								
	(W-30) × 2				×					
	(W-30) × 3					×				
	(W-30) × 4		×						×	
	(W-30) × 6			×						×
	(W-30) × 8						×			
	(W-30) × 12							×		

用判定表表示计算行李费的算法

第7章 实现

- 编码风格
  - 编码风格的作用就是使代码容易读；
  - 风格良好的代码更容易阅读和理解，错误更少；

1. 使用一致和有意义的标识符名
2. 用缩进显示程序结构
3. 用加括号的方式排除二义性
4. 避免大量使用循环嵌套和条件嵌套
5. 当心运算符的副作用
6. 把数定义称常量
7. 利用sizeof()计算对象的大小
8. 清晰的代码，而非最巧妙的代码
9. 程序的注释
   - 序言性注释和功能性注释
   - 对一段程序注释，而不是每一个语句
10. 使用数据结束标记（EOF、BOF），不要指定数据的数目来判断文件的结束。

第7章 实现

- 测试
  - 测试的目的就是在软件投入生产性运行之前，尽可能多地发现软件中的错误。
  - 调试的目的是诊断并改正错误。
  - 对软件规格说明、设计和编码的最后复审。
  - 开发总工作量的40%以上，极端情况下，其他开发步骤总成本的3倍到5倍。
  - 测试是为了发现程序中的错误而执行程序的过程。
  - 好的测试方案是极可能发现迄今为止尚未发现的错误的测试方案。
  - 成功的测试是发现了至今为止尚未发现的错误的测试。
  - 测试只能查找出程序中的错误，不能证明程序中没有错误。
  - Pareto原理：80%的错误很可能是20%的模块造成的。
  - 从“小规模”测试逐步到“大规模”测试。
  - 穷举测试是不可能的。
  - 为了达到最佳的测试效果，应该由独立的第三方从事测试工作。

第7章 实现

- 测试方法
  - 黑盒测试：又称功能测试或数据驱动测试
  - 白盒测试：又称结构测试或逻辑驱动测试
- 测试步骤
  - 模块测试（单元测试）
    - 在这个测试步骤中所发现的往往是编码和详细设计的错误
  - 子系统测试
    - 模块放在一起形成一个子系统来测试
    - 着重测试模块的接口
  - 系统测试
    - 经过测试的子系统装配成一个完整的系统来测试
    - 发现的往往是软件设计中的错误，也可能发现需求说明中的错误

集成测试

  - 验收测试（确认测试）
    - 它的目标是验证软件的有效性（如果软件的功能和性能如同用户所合理期待的那样，软件就是有效的）
    - 用户积极参与，可能主要使用实际数据进行测试
    - 发现的往往是系统需求说明书中的错误
  - 平行运行

第7章 实现

- 回归测试
  - 回归测试是指重新执行已经做过的测试的某个子集，以保证变化（程序改错、新模块加入等）没有带来非预期的副作用。
- (α) Alpha测试
  - Alpha测试由用户在开发者的场所进行，并且在开发者对用户的“指导”下进行测试。开发者负责记录发现的错误和使用中遇到的问题。总之，Alpha测试是在受控的环境中进行的。
- (β) Beta测试
  - 软件的多个用户在一个或多个用户的实际使用环境下进行的测试。开发者通常不在测试现场，是在开发者无法控制的环境下进行的软件现场应用。

第7章 实现

- 集成测试方法
  - 非渐增式集成
    - 先分别测试每个模块，再把所有模块按设计要求放在一起结合成所要的程序。
    - 先进行单元测试，再进行集成测试
  - 渐增式集成

    将单元测试与集成测试结合在一起，把下一个要测试的模块同已经测试好的那些模块结合起来进行测试，测试完以后再把下一个应该测试的模块结合进来测试。
    - 自顶向下（Top-Down）集成
    - 自底往上（Bottom-Up）集成
    - 三明治式（Sandwich）集成

第7章 实现

- 几种集成测试方法的优缺点

方法	优点	不足
非渐增式集成		**没有错误隔离手段 主要设计错误发现迟**
自顶向下集成	**错误隔离 较早发现主要设计错误**	**可复用模块得不到充分测试**
自底往上集成	**错误隔离 可复用模块得到充分测试**	**主要设计错误发现迟**
三明治式集成	**错误隔离 较早发现主要设计错误 可复用模块得到充分测试**	

第7章 实现

- 白盒测试技术：逻辑覆盖
  - 定义：以程序内部的逻辑结构为基础的设计测试用例的技术
  - 类型
    1. 语句覆盖：选择足够多的测试数据，使被测程序中每个语句至少执行一次。
    2. 判定覆盖：判定覆盖又叫分支覆盖，选择足够多的测试数据使每个判定的每个分支都至少执行一次。
    3. 条件覆盖：选择足够多的测试数据使每个判定表达式中的每个条件都取到各种可能的结果
    4. 判定／条件覆盖：选取足够多的测试数据，使得判定表达式中的每个条件都取到各种可能的值，而且每个判定表达式也都取到各种可能的结果。
    5. 条件组合覆盖：选取足够多的测试数据，使得每个判定表达式中条件的各种可能组合都至少出现一次。
  - 设计测试用例（会做）

第7章 实现

- 黑盒测试技术
  1. 等价划分
     - 把程序的输入域划分成若干个数据类，每类中的一个典型值在测试中的作用与这一类中所有其他值的作用相同。据此导出测试用例。
     - 设计测试用例

       (1) 设计一个新的测试方案以尽可能多地覆盖尚未被覆盖的有效等价类，重复这一步骤直到所有有效等价类都被覆盖为止；

       (2) 设计一个新的测试方案，使它覆盖一个而且只覆盖一个尚未被覆盖的无效等价类，重复这一步骤直到所有无效等价类都被覆盖为止。
  2. 边界值分析
     - 着重测试输入等价类和输出等价类的边界，选取的测试数据应该刚好等于、刚刚小于和刚刚大于边界值。
- 会用等价划分和边界值分析法设计测试用例

第8章 维护

- 所谓软件维护就是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。保证软件在一个相当长的时期能够正常运行。
- 60%以上, 这个百分比还在持续上升。
- 维护类型
  - 改正性维护

    诊断和改正错误的过程。17%～21%
  - 适应性维护

    为了适应环境的变化进行的修改软件的活动。 18%～25%
  - 完善性维护

    增加新功能或修改已有功能。 50%～66%
  - 预防性维护

    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。4%左右

软件的可维护性

- 维护人员理解、改正、改动或改进这个软件的难易程度。
- 提高可维护性是支配软件工程方法学所有步骤的关键目标。
- 决定软件可维护性的因素
  1）可理解性
  2）可测试性
  3）可修改性
  4）可移植性
  5）可重用性
    - 软件中使用的可重用构件越多，软件的可靠性越高，改正性维护需求越少。
    - 软件中使用的可重用构件越多，适应性和完善性维护也就越容易。

第9章　面向对象方法学引论

- 面向对象方法学的4个要点
  1)客观世界是由各种对象（Object)组成。面向对象的软件系统是由对象组成的。
  2)对象组成对象类(Class)。类是具有相同属性和行为的对象的集合。每个对象类定义了一组数据和一组方法。
  3)按照子类(派生类)与父类(基类)的关系，对象类组成一个层次结构的系统(类等级)。子类继承(inheritance)父类的数据和方法。
  4)对象彼此之间仅能通过传递消息(Message)互相联系。

OO=objects+classes+inheritance+communication with messages
面向对象 = 对象 + 类 + 继承 + 消息通信

第9章　面向对象方法学引论

特性	面向对象方法	传统方法
思维方法	与人类习惯的思维方法一致，解空间与问题空间一致 ❖ 以对象为核心，数据以及数据上的操作封装为一个统一体--对象 ❖ 软件系统由对象组成，以对象间的消息模拟实体间的联系。 ❖ 抽象思维，归纳思维，演绎思维 ❖ 开发围绕建立问题领域的对象模型，是一个逐步深化的渐进过程	解空间与问题空间不一致 ❖ 以算法为核心，数据和过程分离。 ❖ 软件系统由模块组成，模块间通过调用来集成系统。 ❖ 自顶向下按部就班。 总存在用错误的数据调用正确的模块，或用正确的数据调用错误的模块的危险。
稳定性	较好 ❖功能需求变化仅需要作一些局部性的修改 ❖可派生子类以实现功能扩充或修改	较差 ❖基于功能分解，需求变化大多针对功能 ❖功能变化引起软件结构的整体修改
可重用性	较好 ❖通过对象实例或派生类 ❖方便修改和扩充，且不影响原有类的使用	较差 ❖标准函数库不是自含的和独立的 ❖模块重用，则相应的数据也必须重用。
开发大型软件	较易 ❖可分解成相互独立的小产品	较难 ❖分而不解
可维护性	较好 ❖稳定性好，易修改，易理解。	较差 ❖稳定性较差，较难修改，较难理解。

第9章　面向对象方法学引论

- 面向对象的一些概念
  - 对象：封装了数据结构及可以施加在这些数据结构上的操作（服务或方法）的封装体。
  - 类：具有相同数据和相同操作的一组相似对象的集合
  - 消息：要求某个对象执行在定义它的那个类中所定义的某个操作的规格说明。
  - 方法：对象所能执行的操作，也就是类中所定义的服务。（如C++的成员函数）
  - 属性：类中所定义的数据（如C++的数据成员）
  - 封装：表示对象状态的数据和实现操作的代码与局部数据，都被封装在黑盒子里面，不能从外面直接访问或修改这些数据和代码。
  - 继承：是子类自动地共享基类中定义的数据和方法的机制。
  - 多态性：同一方法，不同的子类有不同的实现。
  - 函数重载：是指在同一作用域内的若干个参数特征不同的函数可以使用相同的函数名字。

第10章 面向对象分析

- 面向对象分析基本顺序
  1) 寻找类－&－对象；
  2) 识别结构；
  3) 识别主题；
  4) 定义属性；
  5) 建立动态模型；
  6) 建立功能模型；
  7) 定义服务。

第9，10章　面向对象方法学

- 面向对象建模
  - 三种模型
    - 对象模型：它是对模拟客观世界实体的对象以及对象彼此间的关系的映射，描述系统的数据结构。
    - 动态模型：描述系统的控制结构。
    - 功能模型：描述系统的功能
  - 建模图形工具
    - 类图：类图描述类及类与类之间的静态关系（关联、聚集、泛化等）。（根据描述画图）
    - 类的状态图：通过描绘对象的状态及引起对象状态转换的事件，来表示对象的行为。（根据事件跟踪图画状态图）
    - 用例图：描述的是外部行为者所理解的系统功能。系统、行为者、用例及用例之间的关系。（根据描述画图）
    - 事件跟踪图：（根据描述画图）

第11章　面向对象设计

- 面向对象设计的任务
  - 系统设计: 确定实现系统的策略和目标系统的高层结构;
  - 对象设计: 确定解空间中的类、关联、接口形式及实现服务的算法。

第11章　面向对象设计

- 面向对象设计的准则
  - 模块化、抽象、信息隐藏
  - 耦合

    (1) 交互耦合

    对象之间的耦合通过消息连接来实现，则这种耦合就是交互耦合。应该尽量减少消息中包含的参数个数，降低参数的复杂程度。减少对象发送(或接收)的消息数。

    (2) 继承耦合

    继承是一般化类与特殊类之间耦合的一种形式。设计应该使特殊类尽量多继承并使用其一般化类的属性和服务。
  - 内聚

    （1） 服务内聚

    一个服务应该完成一个且仅完成一个功能。

    （2） 类内聚

    类的属性和服务应该全都是完成该类对象的任务所必需的，其中不包含无用的属性或服务。

    （3） 一般-特殊内聚

    一般-特殊结构应该是对相应的领域知识的正确抽取。

第11章　面向对象设计

- 重用
  - 重用有两方面的含义：一是尽量使用已有的类，二是如果确实需要创建新类，则在设计这些新类的协议时，应该考虑将来的可重复使用性。
  - 类构件的重用方式

    (1) 实例重用
    - 按照需要创建类的实例
    - 用几个简单的对象作为类的成员创建出一个更复杂的类。

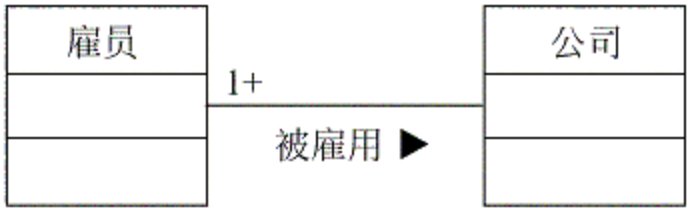
    (2) 继承重用
    - 为提高继承重用的效果，关键是设计一个合理的、具有一定深度的类构件继承层次结构。

    (3) 多态重用
    - 利用多态性不仅可以使对象的对外接口更加一般化，从而降低了消息连接的复杂程度，而且还提供了一种简便可靠的软构件组合机制。
    - 转换接口：类构件在重用时都必须重新定义的服务的集合。
    - 扩充接口：如果在派生类中没有给出扩充接口的新算法，则将继承父类中的算法。

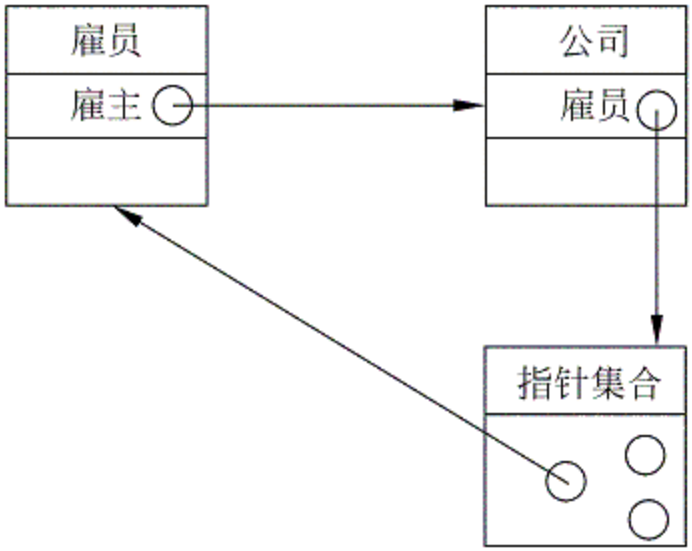
第11章　面向对象设计

- 设计关联
  - 实现关联的具体策略
    - 单向遍历
    - 双向遍历
      - 用属性实现两个方向的关联
      - 用独立的关联对象实现双向关联

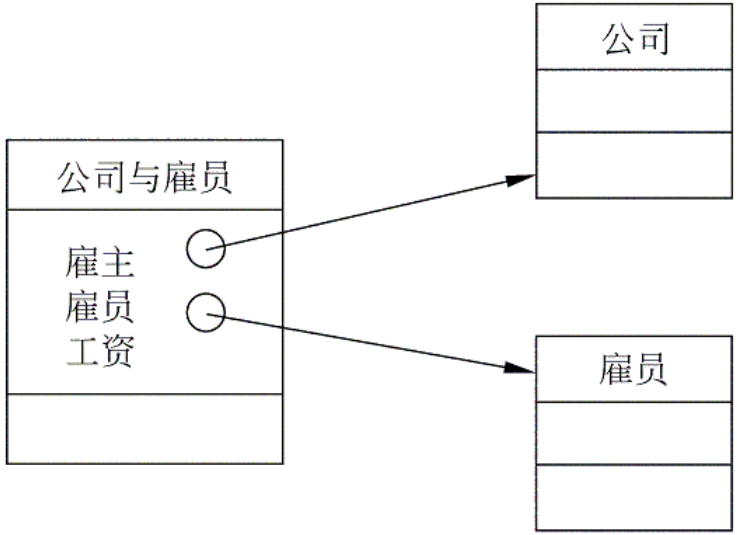
第11章　面向对象设计



1，用属性实现一个方向的关联



2，用属性实现两个方向的关联



3，用独立的关联对象实现双向关联

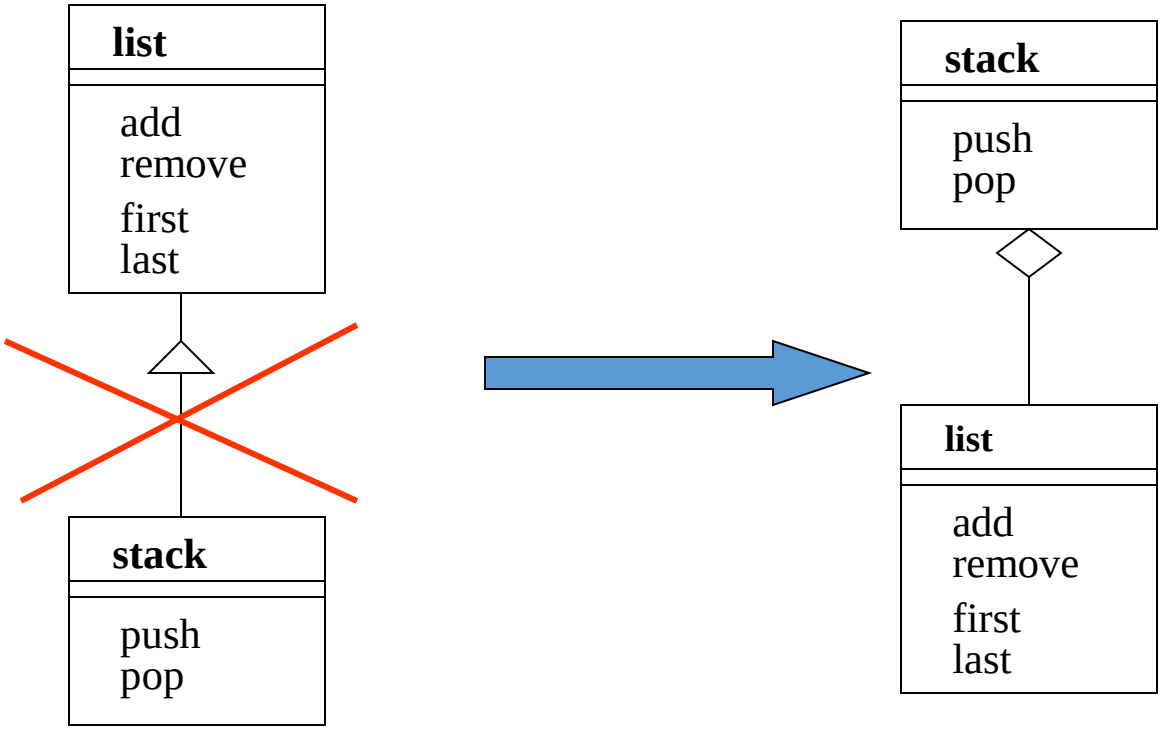
第11章　面向对象设计

- 设计优化

1. 提高效率的几项技术
   - (1) 增加冗余关联以提高访问效率
   - (2) 调整查询次序
   - (3) 保留派生属性
2. 调整继承关系
   - (1) 抽象与具体
   - (2) 为提高继承程度而修改类定义
   - (3) 利用委托实现行为共享

     仅当存在真实的一般-特殊关系(即子类确实是父类的一种特殊形式)时，利用继承机制实现行为共享才是合理的。

利用委托实现行为共享


list
add
remove
first
last
stack
push
pop
stack
push
pop
list
add
remove
first
last