面向对象开发的理论基础

主讲：华俊昌

CONTENT

目录

01

Part one

结构化方法

结构性的编程思想

20世纪60年代编程思想发生了一场革命性的变化，结构性的编程方法出现了。结构化程序设计思想曾为解决“软件危机”立下过汗马功劳，它在一定程度上解决了软件的可靠性、可理解性、可维护性等问题；

C语言里面有几个能表现出结构化思想的地方：分支（if），循环（while，for），结构体(struct)。

概述

- 结构化程序设计思想

体现了人们抽象思维和复杂问题分解的基本原则与要求

结构化技术的三个方面：

- 结构化分析（SA）

基于功能分解的分析方法，即使用数据流程图、决策表等工具建立符合用户需求的结构化说明书

- 结构化设计（SD）

面向数据流的设计方法，从而提出满足系统需求的最佳软件结构，完成软件层次图或者软件结构图

- 结构化编程（SP）

一种编程典范。它采用子程序、程式码区块（英语：block structures）、for循环以及while循环等结构，来取代传统的 goto。希望借此来改善计算机程序的明晰性、品质以及开发时间，并且避免写出面条式代码。

结构化方法的基本原理

结构化分析是以系统中数据的加工处理过程分析为主要内容的分析方法。

结构化设计是以模块功能及其处理过程设计为主要内容进行详细设计的 一种设计方法。

其概念最早由E.W.Dijikstra在1965年提出，它是软件发展的一个重要的里程碑。

结构化开发方法也称为**面向过程的方法**或**传统软件工程开发方法**，它的观点是采用自顶向下、逐步求精的程序设计方法。

使用三种基本控制结构构造程序，任何程序都可由顺序、选择、循环三种基本控制结构构造；

详细描述处理过程常用三种工具：图形、表格和语言；

使用的手段主要有数据流图、数据字典、层次方框图、流程图、结构化语言等。

结构化程序设计的一般步骤是：**分析业务流程、分析数据信息的加工处理过程；画出数据流图；建立数据字典；提出系统的总体逻辑方案；细化数据流图；确定模块的接口；为每个模块确定采用的算法和数据结构；根据E-R图设计数据库、根据模块算法编程等。**

结构化开发

结构化开发的主要过程：问题定义、可行性论证及软件计划、需求分析、总体设计、详细设计等。

每个阶段使用的工具如图所示：

阶段	拟解决的关键性问题	工具	交付成果
问题定义	要解决的问题是什么		
可行性论证及软件计划	有行得通的解决办法吗		可行性分析报告
需求分析	系统必须做什么	数据流图、数据字典	需求规格说明书
总体设计	概括地说，应该怎样做	系统结构图、层次方框图	概要设计说明书
详细设计	具体怎样做	HIPO图、处理流程图	详细设计说明书

结构化开发方法举例

结构化开发方法举例-书店借书系统

用到的分析与设计工具

- 数据流图
- 数据字典
- 系统功能结构层次图与HIPO图、处理流程图
- E-R图

数据流图（Data Flow Diagram ,DFD图）

它是描述数据加工处理过程的工具，有四种基本符号如下图所示。

基本符号的含义：

- 矩形方框表示数据的源点或终点，是系统的外部实体.
- 圆形表示变换数据的处理。
- 两条平行横线代表数据存储。
- 箭头表示数据流，即特定数据的流动方向。

数据存储和数据流都是数据，仅仅所处的状态不同。数据存储是处于静止状态的数据，数据流是处于运动中的数据。

数据流图

画数据流图的基本原则：自顶向下、逐层细化、完善求精。

具体步骤:

（1）绘顶层数据流图。找出对整个系统而言的输入、输出数据，确定外部实体，它们决定了系统与外界的接口。

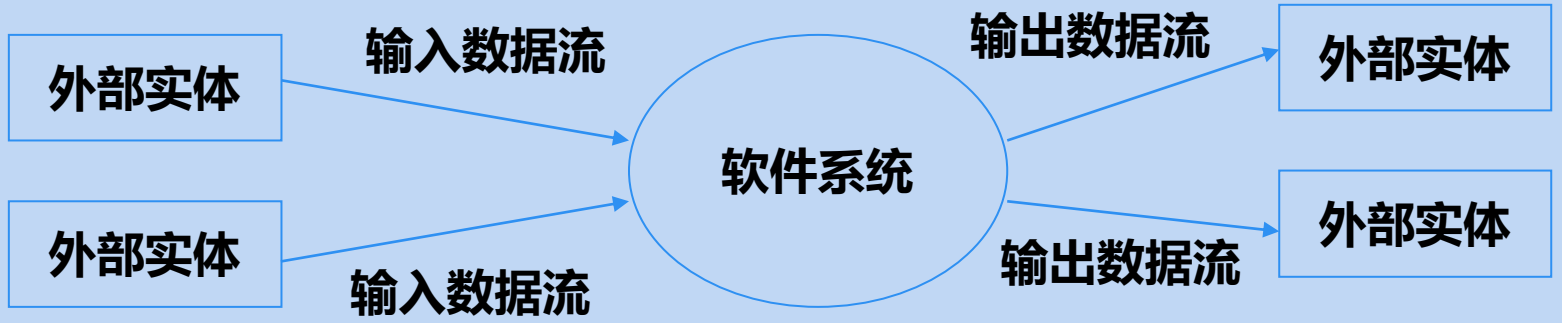
（2）为数据流命名，为加工命名。

（3）检查核对。

（4）核对无误后，进行分解，画处理的内部。

在（2）至（4）步之间反复迭代，直到处理无法进一步分解为止。

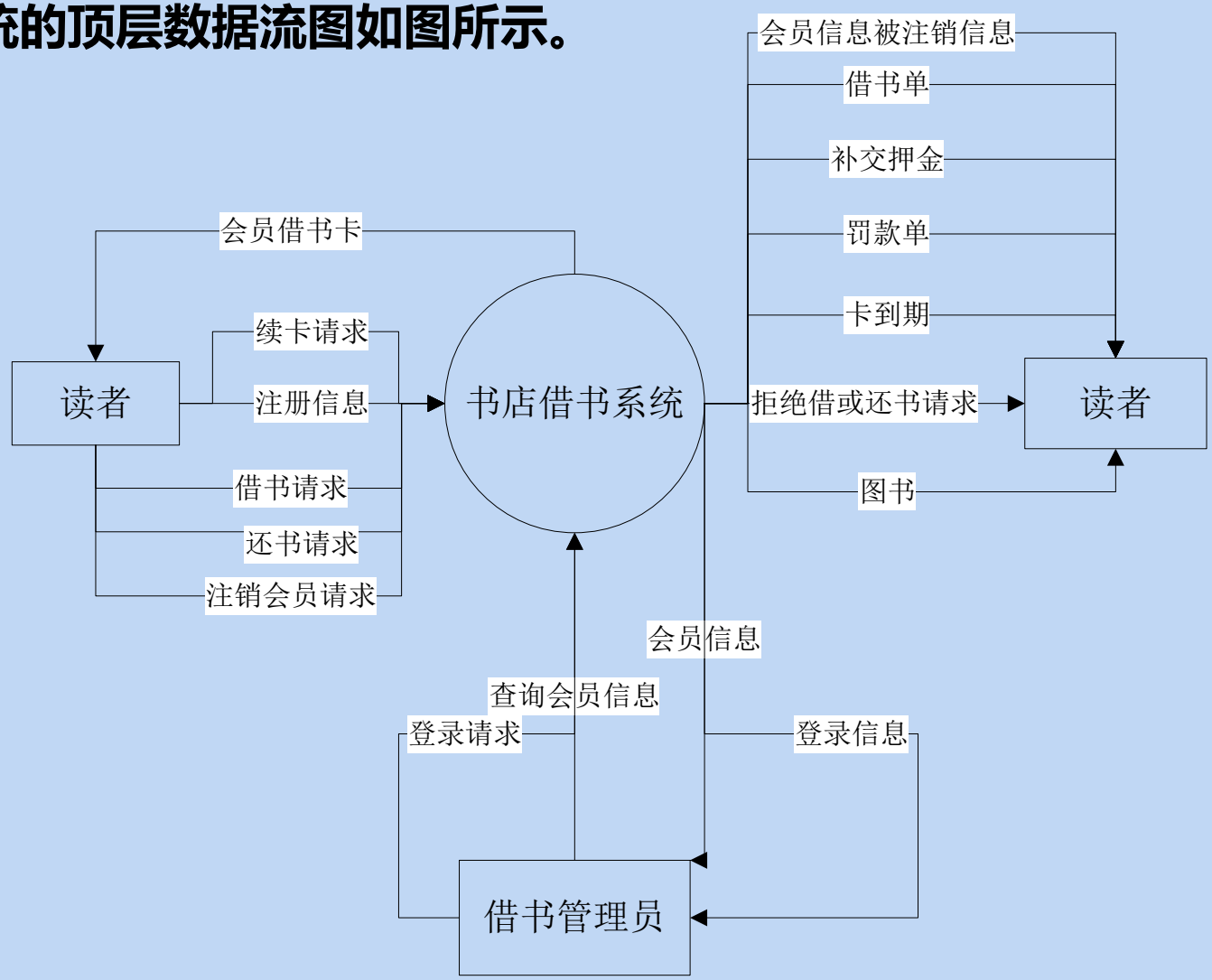
顶层数据流图只要求表示对整个系统而言的输入输出数据，如图所示。



顶层数据流图的画法

数据流图

例如，书店借书系统的顶层数据流图如图所示。



数据字典

数据字典是对所有与系统相关的数据元素的一个有组织的列表，精确、严格地定义各个数据元素，使得用户及开发人员对于输入、输出、存储和处理形成共同的理解。

数据字典由对下列6类元素的定义组成：数据流、数据流分量、数据结构、数据存储、处理逻辑、外部实体。数据字典是对数据流图的详细描述。

例如:

借阅制度表 = 读者类别+允许借阅册数+罚款规定+丢失图书罚款规定

读者类别 = [金卡 | 银卡 | 铜卡]

又如:

酒店客房预订请求 = 客人数据+住宿期限+客房类别

客人数据 = 客人姓名+地址+身份证号码+[护照号码] +支付方式

身份证号码 = 15{十进制数字}18

护照号码 = 字母 + 8{十进制数字}8

字母 = "A"..."Z"

十进制数字 = "0"..."9"

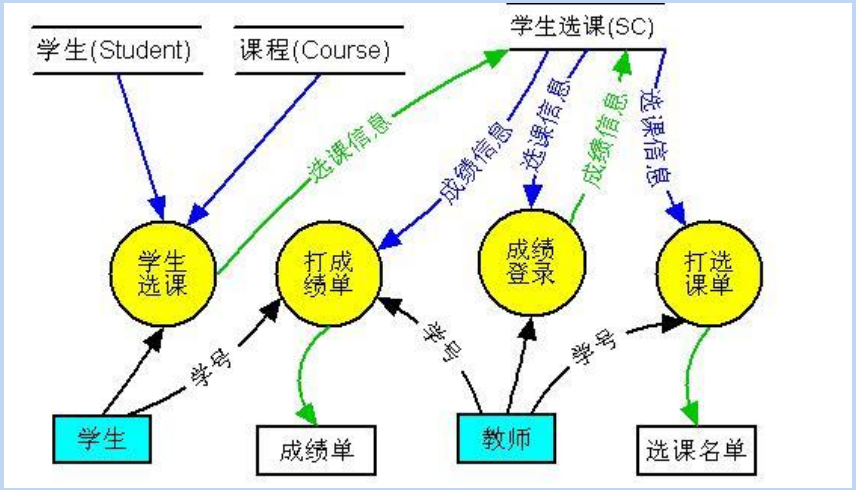
数据字典

数据字典应用举例

以**学生选课**数据字典为例简要说明

①数据项：以“学号”为例
数据项名：学号
数据项含义：唯一标识每一个学生
别名：学生编号
数据类型：字符型
长度：8
取值范围：00000～99999
取值含义：前2位为入学年号，后3位为顺序编号
与其他数据项的逻辑关系：（无）

② **数据结构**：以“学生”为例
数据结构名：学生
含义说明：是学籍管理子系统的主体数据结构，定义了一个学生的有关信息
组成：学号，姓名，性别，年龄，所在系



③ **数据流**：以“选课信息”为例
数据流名：选课信息
说明：学生所选课程信息
数据流来源：“学生选课”处理
数据流去向：“学生选课”存储
组成：学号，课程号
平均流量：每天10个
高峰期流量：每天100个

数据字典应用举例

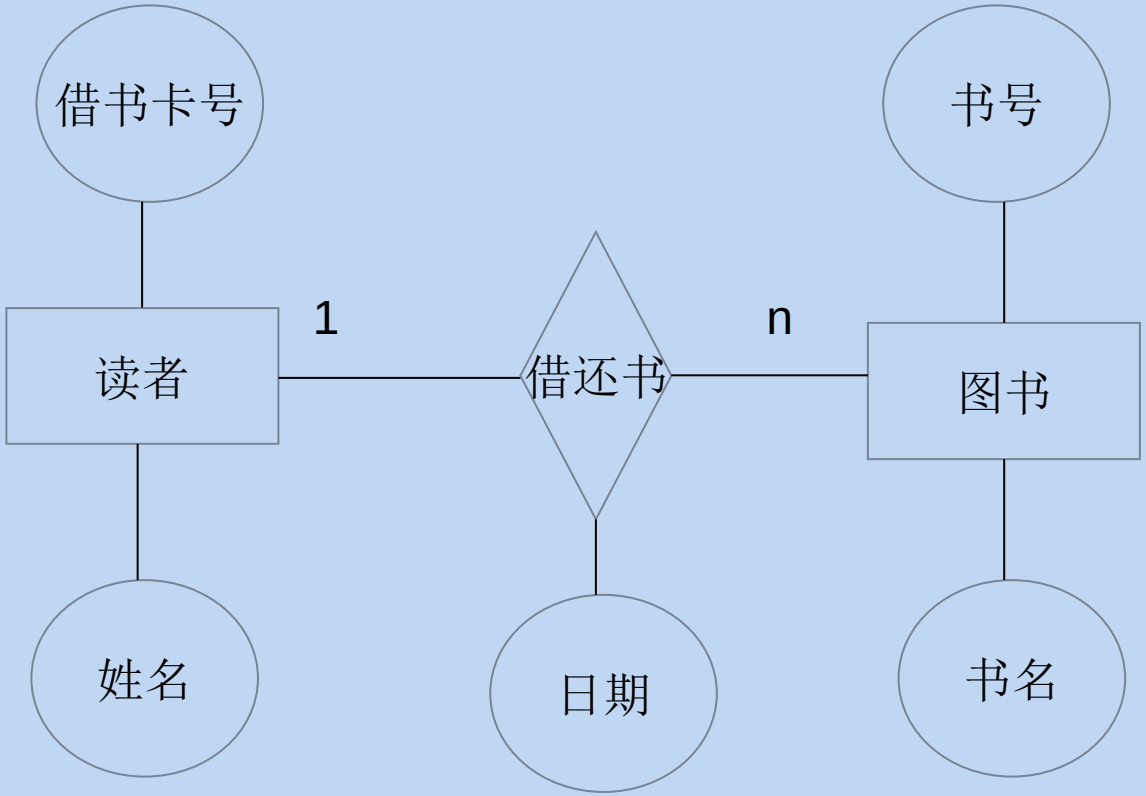
④ **数据存储**：以“学生选课”为例
数据存储名：学生选课
说明：记录学生所选课程的成绩
编号：（无）
流入的数据流：选课信息，成绩信息
流出的数据流：选课信息，成绩信息
组成：学号，课程号，成绩
数据量：50000个记录
存取方式：随机存取

⑤ **处理过程**：以“学生选课”为例
处理过程名：学生选课
说明：学生从可选修的课程中选出课程
输入数据流：学生，课程
输出数据流：学生选课
处理：每学期学生都可以从公布的选修课程中选修自己愿意选修的课程，选课时有些选修课有先修课程的要求，还要保证选修课的上课时间不能与该生必修课时间相冲突，每个学生四年内的选修课门数不能超过8门。

E-R图（实体-关系图）

数据字典应用举例

例如:图书馆“读者借还书”的E-R图。



层次图与HIPO图

数据字典应用举例

例如:图书馆“读者借还书”的E-R图。　层次图用来描绘软件的层次结构，图中的一个矩形框代表一个模块，方框间的连线表示调用关系。它适于在自顶向下设计软件的过程中使用，描述模块的划分。

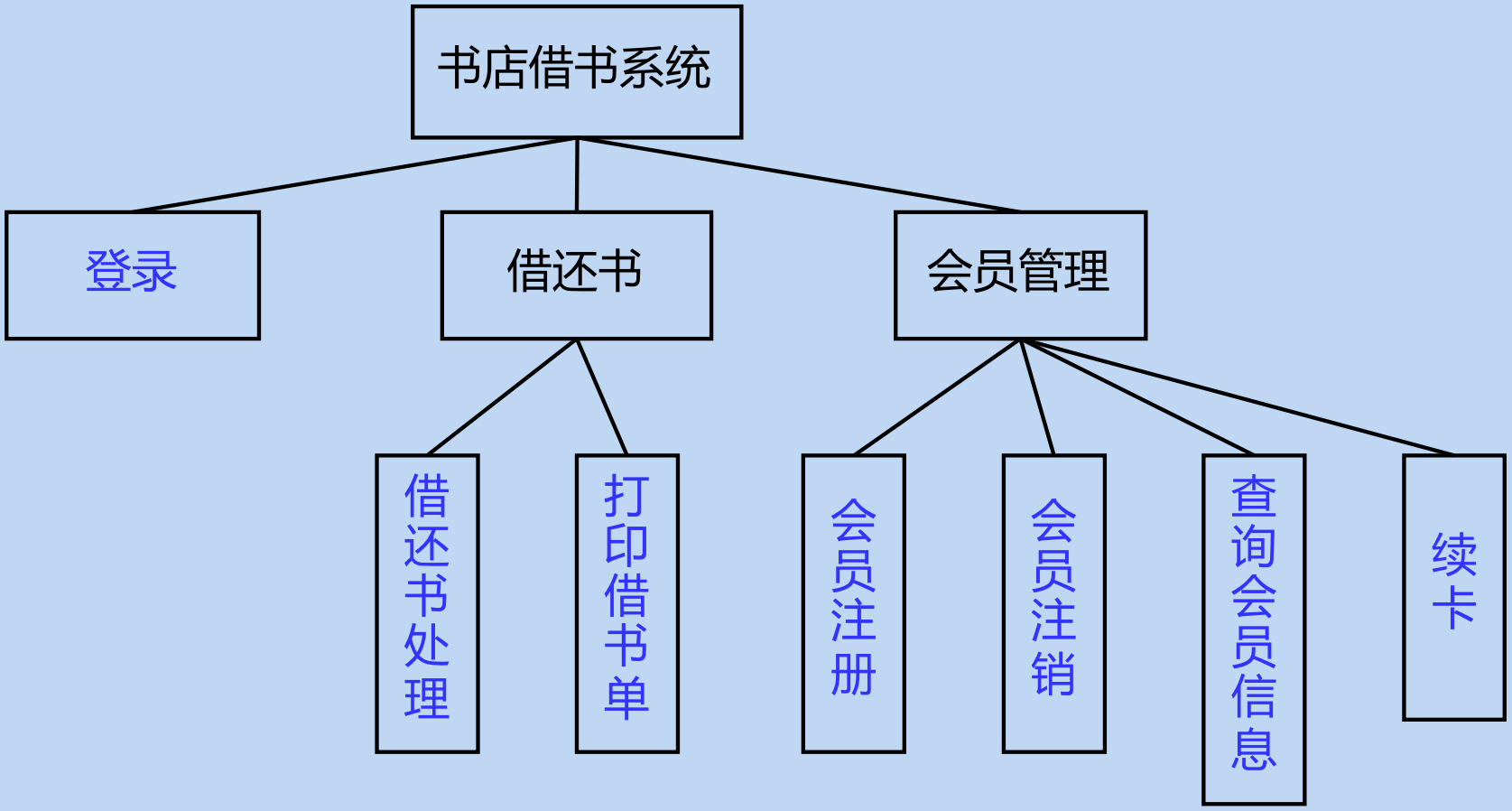
在第1层数据流图中对“书店借书系统”划分模块得到对应的系统功能结构的层次图,如图所示。

HIPO图是“层次图加输入/处理/输出图”的英文缩写，为了能使HIPO图更清晰,首先要对层次图中的各个模块编号, 加了编号后的层次图,称为H图.图中体现了编号的规则.

“2.1.1 借书处理”功能的IPO图和功能处理流程图，据此可以编程。

层次图应用举例

书店借书系统功能结构的层次图

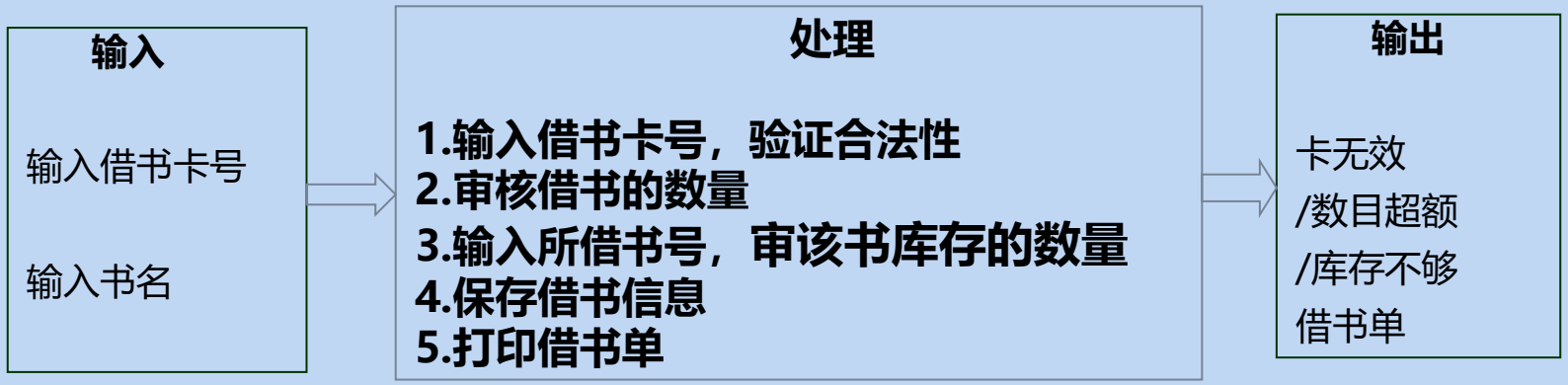


IPO图

数据字典应用举例

IPO是输入、处理、输出的简称，IPO图能方便地描绘输入数据、数据的处理和输出数据的关系。

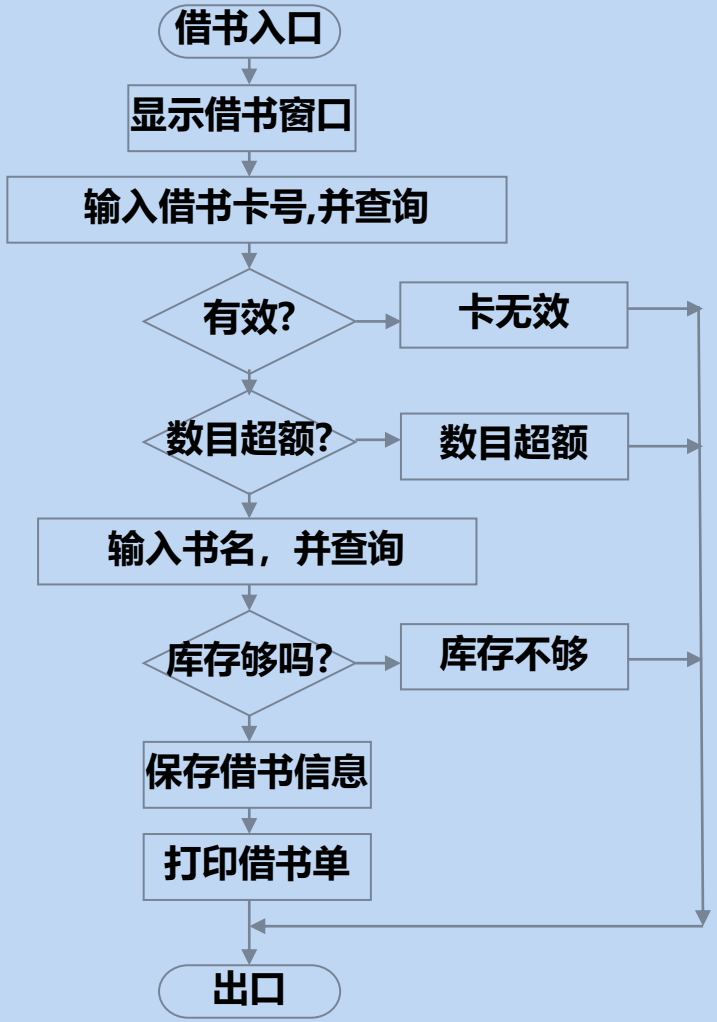
例如：书店借书系统中的”借书处理”功能模块的详细设计--IPO图。



IPO图

功能模块详细设计的IPO表及功能处理流程图

借书处理功能的HIPO表、功能处理流程图。据此可以编程。



IPO表	
系统：书店借书系统	作者：
模块：借书处理	日期：
编号：2.1.1	
被调用：借还书处理	调用：
输入： **输入借书卡号、书名、数量**	输出：**卡无效、数目超额、库存不够、借书单**
处理： 1.输入借书卡号，验证合法性 2.审核借书的数量 3.输入所借书号，审该书库存的数量 4.保存借书信息 5.打印借书单	
局部数据元素	注释

结构化开发方法----结构化分析与设计过程小结

结构化分析是以系统中数据的加工处理过程分析为主要内容的分析方法。主要工具是**数据流图、数据字典**。

结构化设计是以模块功能及其处理过程设计为主要内容进行详细设计的一种设计方法。主要工具是**系统功能结构层次图、IPO图/表、处理流程图。**

阶段	工具
结构化需求分析	数据流图、数据字典
总体设计	系统功能模块结构层次图
详细设计	HIPO图、功能处理流程图

02

Part two

面向对象方法

结构化方法的缺陷

- 以算法为核心，数据和代码分离，反映了计算机的观点，数据和操作不易保持一致性。
- 软件系统的结构紧密依赖于系统所要完成的功能，功能需求的变化易引起软件结构的修改。
- 所使用的标准函数缺乏“柔性”，不能适应不同应用场合的不同需要。
- 不易组织人员开发大型软件，开发出来的软件也很难维护。

面向对象思想的由来

传统的软件开发方法 — 没有直接而全面地反映问题的本质

直接面对问题域中客观存在的事物来进行软件开发 — 面向对象

面向对象方法的发展历史

- 初始阶段
  - 60年代末挪威奥斯陆大学和挪威计算中心共同研制的Simula语言是面向对象发展历史上的第一个里程碑，后来的一些著名面向对象编程语言（如Smalltalk, C++, Eiffel）都受到Simula的启发。
  - 80年代，Xerox研究中心推出了Smalltalk语言和环境，它具备了面向对象语言的继承和封装的主要特征，使面向对象程序设计方法趋于完善，掀起了面向对象研究的高潮。
- 发展阶段
  - 从80年代中期到90年代，面向对象语言十分热门，大批比较实用的面向对象编程语言（Object Oriented Programming Language, OOPL）涌现出来，如C++、Object Pascal、Eiffel、Actor 等，特别是C++语言已成为目前应用最广泛的OOPL。
  - 面向对象编程语言的繁荣是面向对象方法走向实用的重要标志，也是面向对象方法在计算机学术界、产业界和教育界日益受到重视的推动力。

面向对象方法的发展历史

- 成熟阶段
  - 在C++语言十分热门的时候，人们开始了对面向对象分析（Object Oriented Anlysis, OOA）的研究，进而延伸到面向对象设计（Object Oriented Design, OOD）。特别是90年代以后，许多专家都在尝试用不同的方法进行面向对象的分析与设计，这些方法各有所长，力图解决复杂软件的开发问题。
  - 在这段时期，面向对象的分析和设计技术逐渐走向实用，最终形成了从分析、设计到编程、测试与维护一整套的软件工程体系。

面向对象方法的发展情况

面向对象方法发展到软件工程包括三个方面: OOP（面向对象程序设计）、OOA（面向对象的分析）、OOD（面向对象的设计）

① 面向对象分析（OOA）

于现实世界客观存在的事物出发，运用人类的自然思维方式，强调直接以现实世界的事物为中心来思考、认识问题。

②面向对象设计（OOD）

从客观事物的本质特点出发，把它们抽象表示为系统的类，作为系统的基本构成单元，使得该系统的组件可以直接反映客观世界中事物及其相互关系的本来面貌。

③面向对象编程（OOP）

是一种计算机编程架构。其本质是以建立模型体现出来的抽象思维过程和面向对象的方法，模型是用来反映现实世界中事物特征的。

面向对象方法概述

面向对象方法按照人类的自然思维方式，以概念为核心，面对客观世界建立软件系统模型。

- 对象、类、继承、封装、消息等基本符合人类的自然思维方式。
- 有利于对业务领域和系统需求的理解。
- 有利于人员交流。

面向对象方法对需求变化有较好的适应性

- 面向对象的封装机制使开发人员可以把最稳定的部分（即对象）作为构筑系统的基本单位，而把容易发生变化的部分（即属性与操作）封装在对象之内。
- 对象之间通过接口联系，使得需求变化的影响尽可能地限制在对象内部。

面向对象方法概述

面向对象方法支持软件复用

- 对象具有封装性和信息隐蔽等特性，使其容易实现软件复用。
- 对象类可以派生出新类，类可以产生实例对象，从而实现了对象类数据结构和操作代码的软构件复用。
- 面向对象程序设计语言的开发环境一般预定义了系统动态连接库，提供了大量公用程序代码，避免重复编写，提高了开发效率和质量。

面向对象的软件系统可维护性好

- 系统由对象构成，对象是一个包含属性和操作两方面的独立单元，对象之间通过消息联系。
- 系统出错时容易定位和修改，不至于牵一发而动全身。

面向对象方法学优点

- **优点：**

1. 与人类习惯的思维方法一致

传统的程序设计技术是面向过程的设计方法，这种方法以算法为核心，把数据和过程作为相互独立的部分，数据代表问题空间中的客体，程序代码则用于处理这些数据。

2. 稳定性好

传统的软件开发方法以算法为核心，开发过程基于功能分析和功能分解。用传统方法所建立起来的软件系统的结构紧密依赖于系统所要完成的功能，当功能需求发生变化时将引起软件结构的整体修改。事实上，用户需求变化大部分是针对功能的，因此，这样的软件系统是不稳定的。

面向对象方法学优点

- **优点：**

3. 可重用性好

用已有的零部件装配新的产品，是典型的重用技术，例如，可以用已有的预制件建筑一幢结构和外形都不同于从前的新大楼。重用是提高生产率的最主要的方法。

4. 较易开发大型软件产品

在开发大型软件产品时，组织开发人员的方法不恰当往往是出现问题的主要原因。用面向对象方法学开发软件时，构成软件系统的每个对象就像一个微型程序，有自己的数据、操作、功能和用途，因此，可以把一个大型软件产品分解成一系列本质上相互独立的小产品来处理，这就不仅降低了开发的技术难度，而且也使得对开发工作的管理变得容易多了。这就是为什么对于大型软件产品来说，面向对象范型优于结构化范型的原因之一。

5. 可维护性好

用传统方法和面向过程语言开发出来的软件很难维护，是长期困扰人们的一个严重问题，是软件危机的突出表现。

面向对象方法与结构化方法比较

（1）面向对象分析与设计方法小结

反复迭代完善需求。

对已有的需求进行整理，列出需求列表。

与用户交流得到有效的需求列表。

画出初始用例模型，表达系统的主要功能及主要业务流程。

完善需求列表，完善用例模型。

反复迭代进行系统分析。

识别系统中的对象及其关系，画初始类模

确定类的职责、属性和方法。

表示出主要业务过程的动态模型。

由动态模型反复映射，完善类模型。

反复迭代进行系统设计。

确定整个系统的拓扑结构（部署图）

修订类模型。

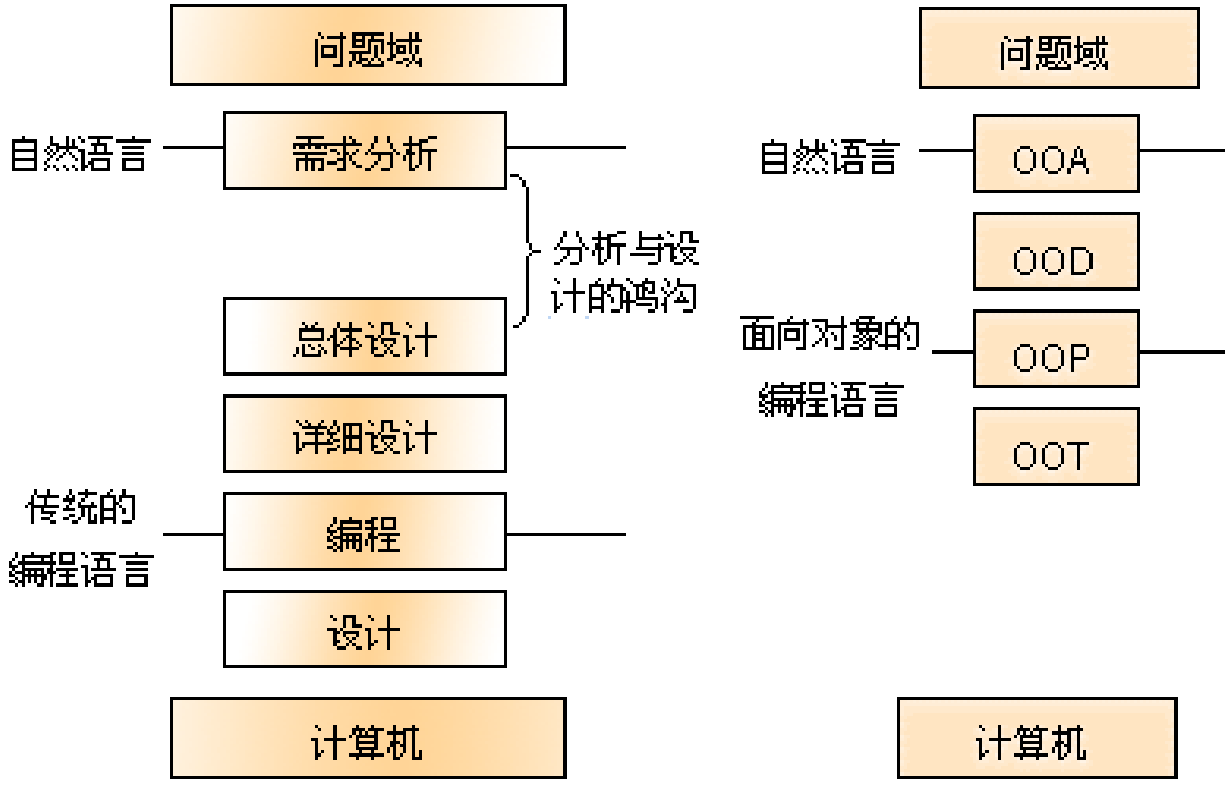
相应修订动态模型。

完成反映程序模块的包图。

完成反映程序软件构成的组件图。

设计界面，设计数据库。

面向过程方法　　面向对象方法



面向对象方法与结构化方法比较

（2）结构化分析与设计方法小结。

获取完整的需求。

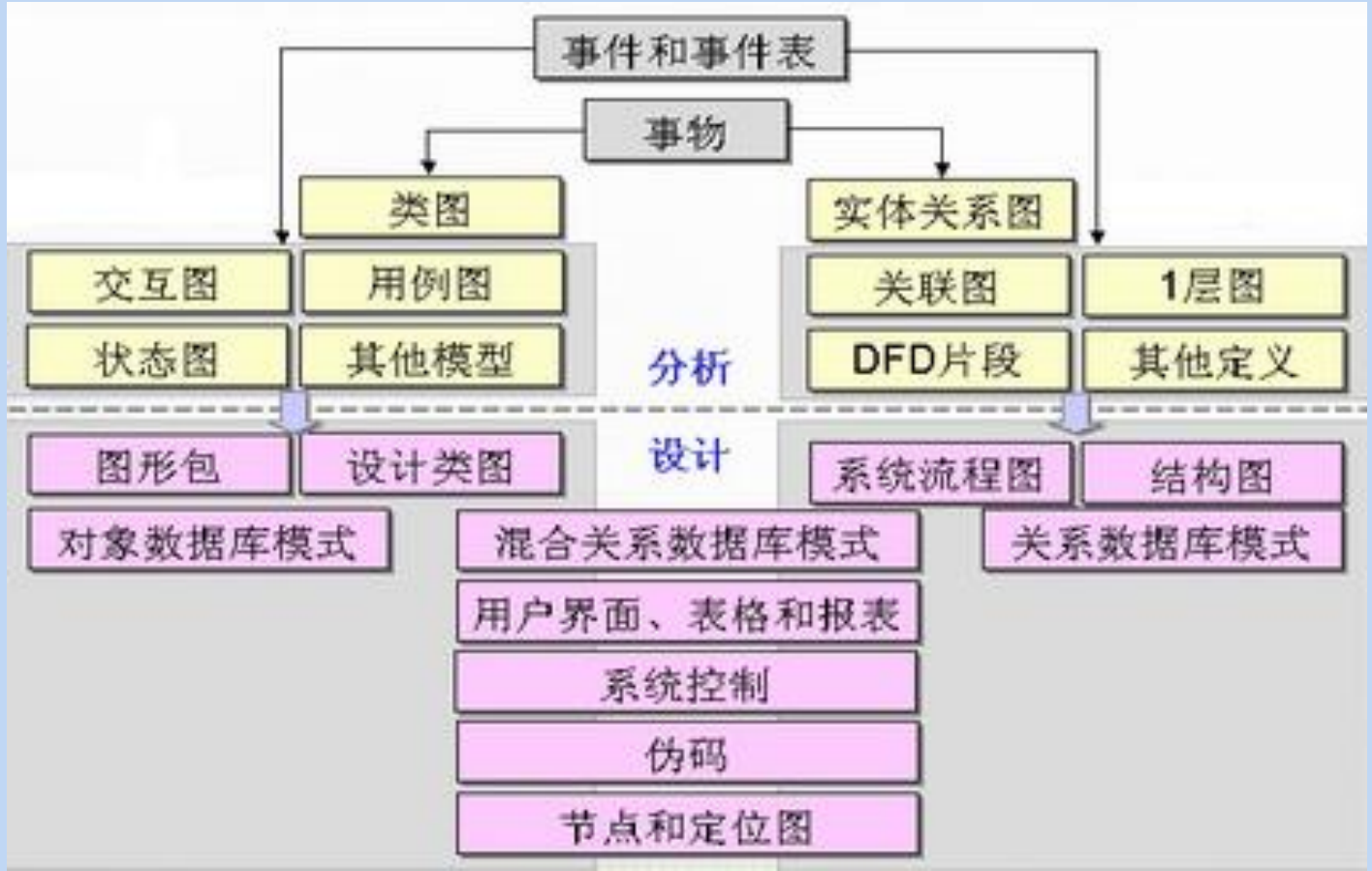
自顶向下、逐层分解，画出数据流图。

书写数据字典。

映射出系统的层次结构，进行系统结构(模块及其接口)设计。

逐层细分，细化出每个处理。

设计界面，设计数据库。



面向对象方法与结构化方法比较

两种方法的简单比较	
面向对象开发方法	结构化开发方法
用例图	数据流图 系统功能结构图
分析: 类图+顺序图	
设计: 类图+顺序图(描述用例功能的实现流程—系列对象方法的调用.) 类的详细设计(含方法设计)	功能处理流程图
编码(类代码)	编码(函数代码)

小结

结构化方法和面向对象的方法是当前两种主流的软件开发方法。

面向对象的开发方法**根据现实问题直接抽象出对象，分析对象的行为和与行为相关的数据，对象间通过传递消息进行通信，从问题出发，模拟现实问题，建立系统模型，易于理解和实现。**

结构化开发方法**有一套成熟的理论基础，“自顶向下、逐步求精”，在获取完整的需求之后才能开始系统的分析和设计。**

思考题

- 怎么用结构化的方法和面向对象方法来处理下面的问题
- 请问1999.6.1-2008.4.5之间有多少天。

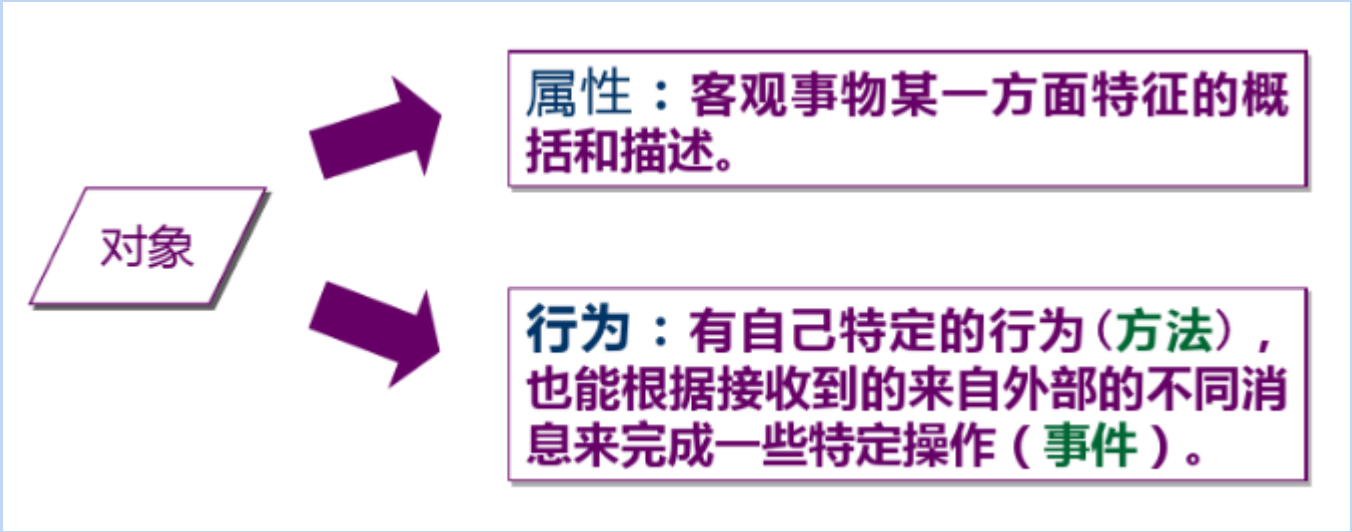
03

Part three

面向对象中的基本实体

对象

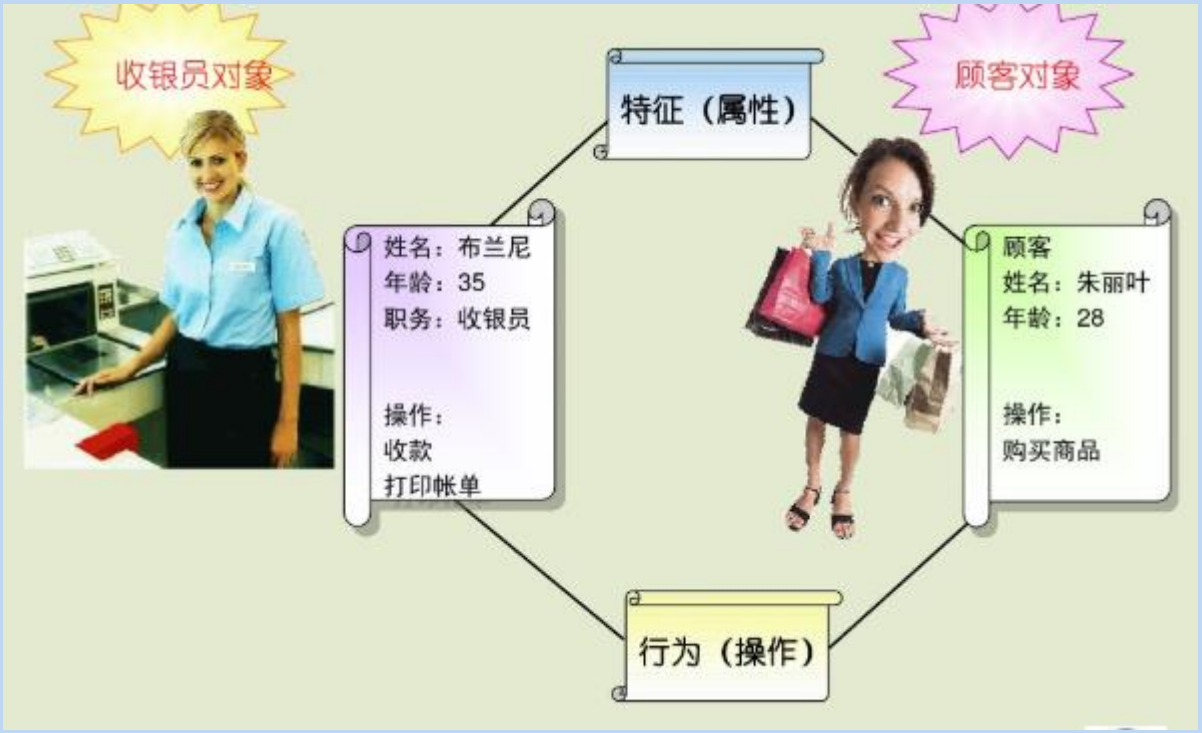
- 对象（Object）
  - 系统中用来描述客观事物的一个实体，它是构成系统的一个基本单位，由一组属性和对这组属性进行操作的一组服务组成。
- 对象的两个基本要素:属性和服务
  - 属性是用来描述对象静态特征的一个数据项
  - 服务是用来描述对象动态特征（行为）的一个操作序列。
- 对象是属性和服务的结合体，对象的属性值只能由这个对象的服务来读取和修改。



对象

对象具有的特征

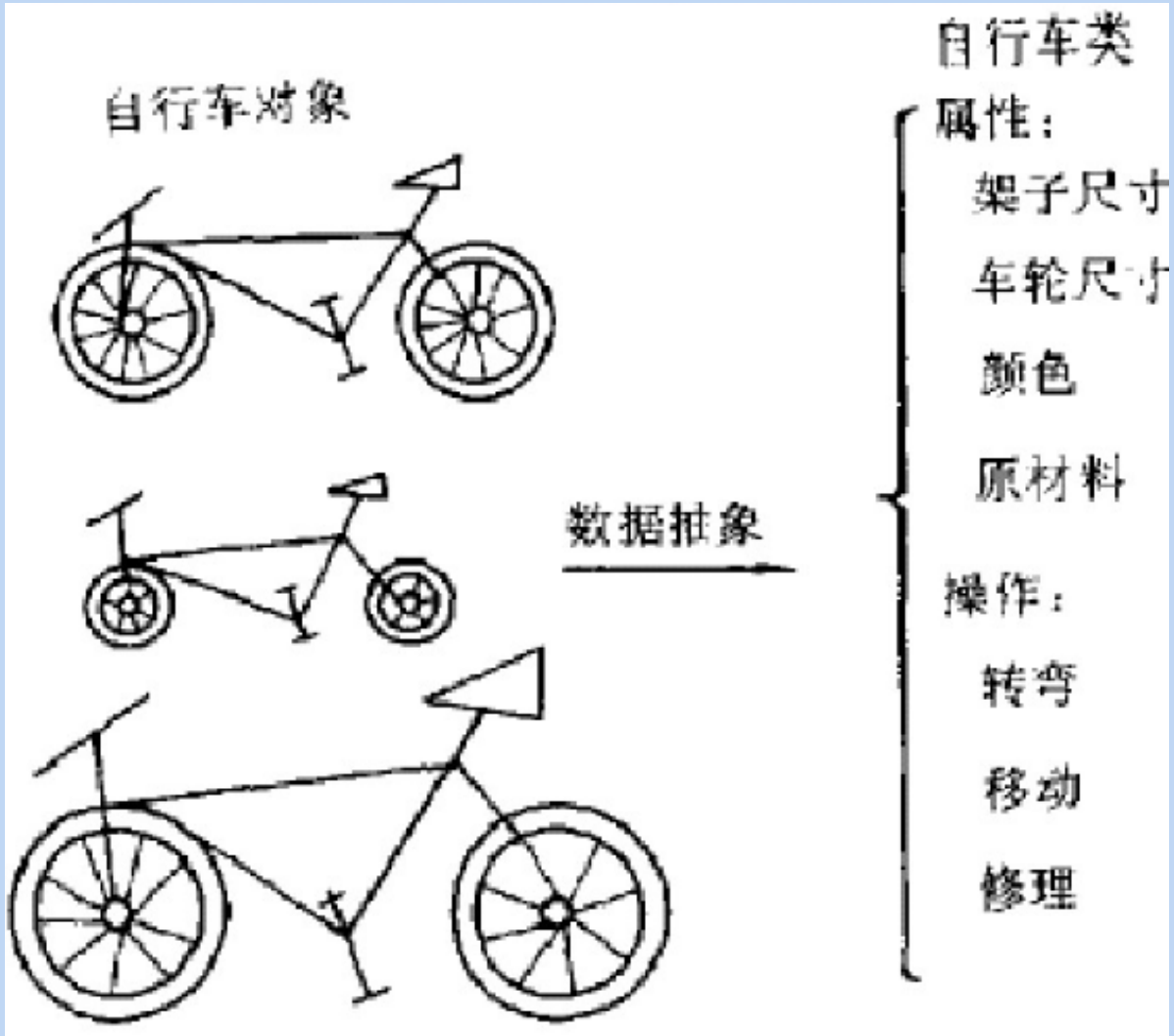
- 万物皆为对象
- 每个对象都有自己的惟一标识
- 对象具有属性和行为
- 对象具有状态
- 对象之间依靠消息实现通信



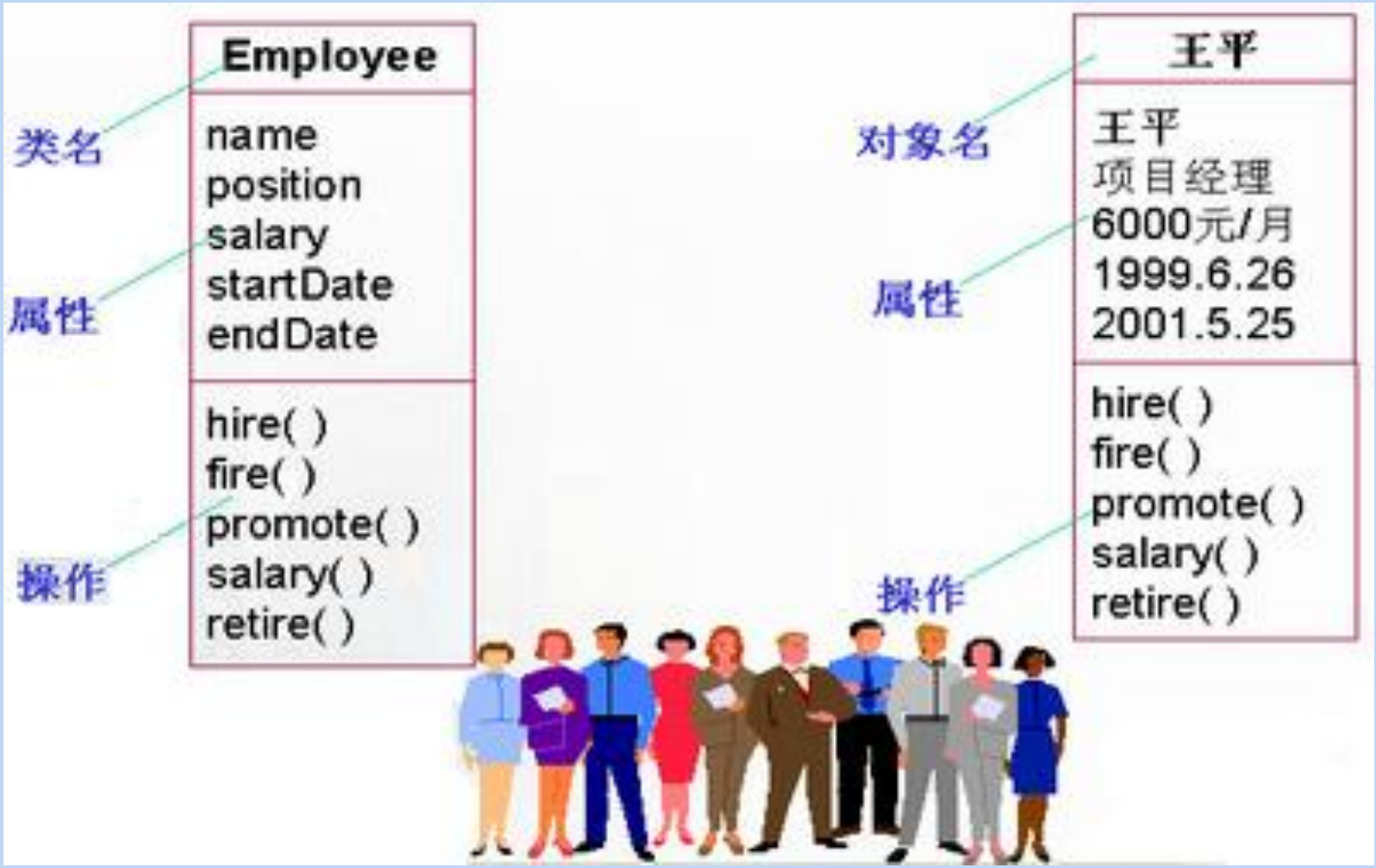
类

- 类（Class）
  - 具有相同属性和服务的一组对象的集合，它为属于该类的全部对象提供了统一的抽象描述，其内部包括属性和服务两个主要部分。
- 类代表一个抽象的概念或事物，对象是在客观世界中实际存在的类的实例。
- 类体现了人们认识事物的基本思维方法

类与对象比较



类与对象比较

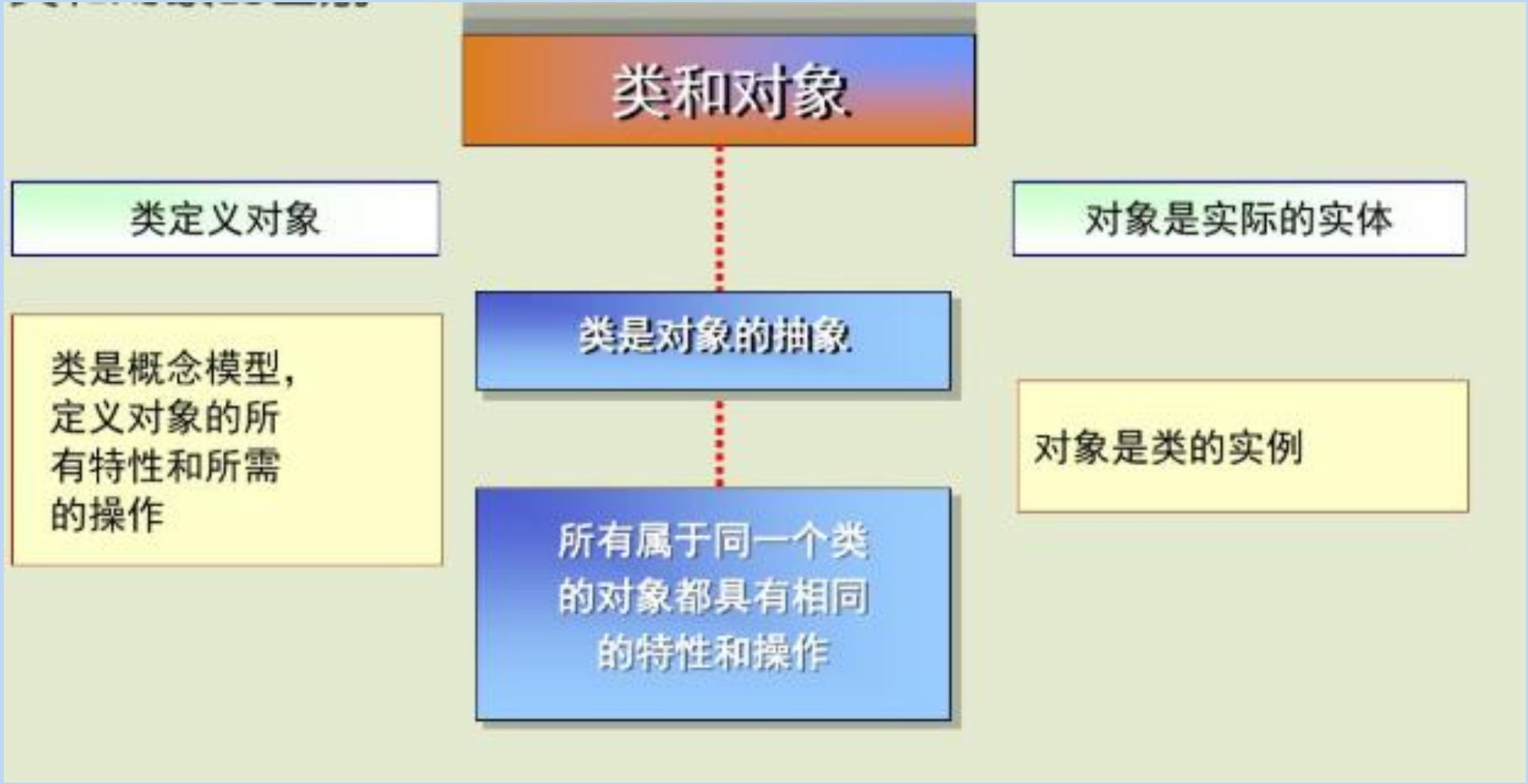

Employee
name
position
salary
startDate
endDate
hire( )
fire( )
promote( )
salary( )
retire( )
类名
属性
操作
王平
王平
项目经理
6000元/月
1999.6.26
2001.5.25
hire( )
fire( )
promote( )
salary( )
retire( )
对象名
属性
操作

类与对象比较

类与对象的比较

- 同类对象具有相同的属性和服务，是指它们的定义形式相同，而不是说每个对象的属性值都相同。
- 类是一组客观对象的抽象表示，它将该组对象所共同具有的结构特征（属性）和行为特征（操作）集中起来加以描述说明。
- 类和对象是抽象和具体的关系，组成类的所有对象成为该类的实例。
- 类是静态的，类的存在、语义和关系在程序执行前就已经定义好了。
- 对象是动态的，对象在程序执行时可以被创建和删除。
- 在面向对象的系统分析和设计中，并不需要逐个对对象进行说明，而是着重描述一批对象共性的类。

类与对象比较


类和对象
类定义对象
对象是实际的实体
类是对象的抽象
类是概念模型，
定义对象的所
有特性和所需
的操作
对象是类的实例
所有属于同一个类
的对象都具有相同
的特性和操作

消息

- 消息（Message）
  - 消息是对象发出的服务请求，一般包含提供服务的对象标识，服务标识、输入信息和应答信息等信息。
  - 一个对象向另一个对象发消息请求某项服务，接收消息的对象响应该消息，激发所要求的服务操作，并把操作结果返回给请求服务的对象。
- 采用消息（而不是函数调用）这个术语更接近人们日常思维。

举例：使用电视机时，用户通过按钮或遥控器发出转换频道的消息，电视机变换对电视台的接收信号频率，并将结果显示给用户。在这里，用户发出的信息包括：

接受者——电视机；

要求的服务——转换频道；

输入信息——转换后的频道序号；

应答信息——转换后频道的节目。

访问说明符/可见性控制

- 访问说明符其实就是设置一个访问权限，只有设置好这个访问权限以后才能更好的封装我们的一些变量或方法。所以学好这个说明符对我们后续学习Java的封装功能很有帮助。
- 访问说明符有哪些?
  - public：公开的。可以被任何类访问，也可以被不同包里面的类访问。
  - private：保密的。该成员只能被当前类访问，其他类是不可以的。不同包也不可以访问
  - protected：受保护的。同包里的类可以访问，不同包里面只有子类才可以访问，继承就可以访问了。
  - default：默认的。同包里面的类可以访问。
- 访问范围由大到小
  - public：任何地方均可访问
  - protected：同一包和子类可见
  - 默认：同一包中可见
  - private：仅该类部可见

04

Part Four

面向对象的构造

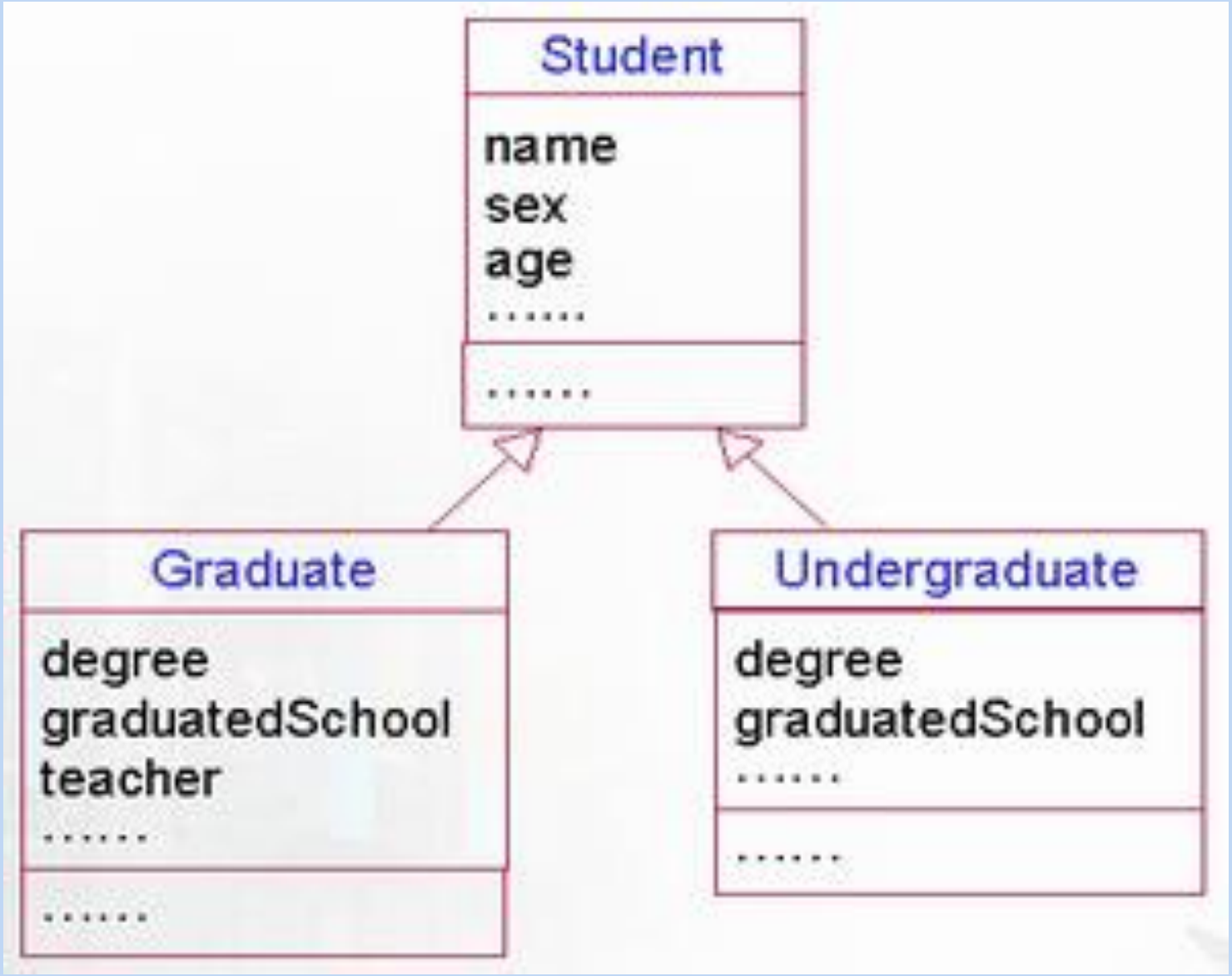
封装

- 封装（Encapsulation）
  - 具有相同属性和服务的一组对象的集合，它为属于该类的全部对象提供了统一的抽象描述，其内部包括属性和服务两个主要部分。
- 封装使对象形成两个部分：接口和实现
  - 对用户来说，接口是可见的，实现是不可见的。
- 封装可以保护对象，避免用户误用，也可以保护客户端，其实现过程的改变不会影响到相应客户端的改变。
- 与封装密切相关的概念是可见性，它是指对象的属性和服务允许对象外部存取和引用的程度。

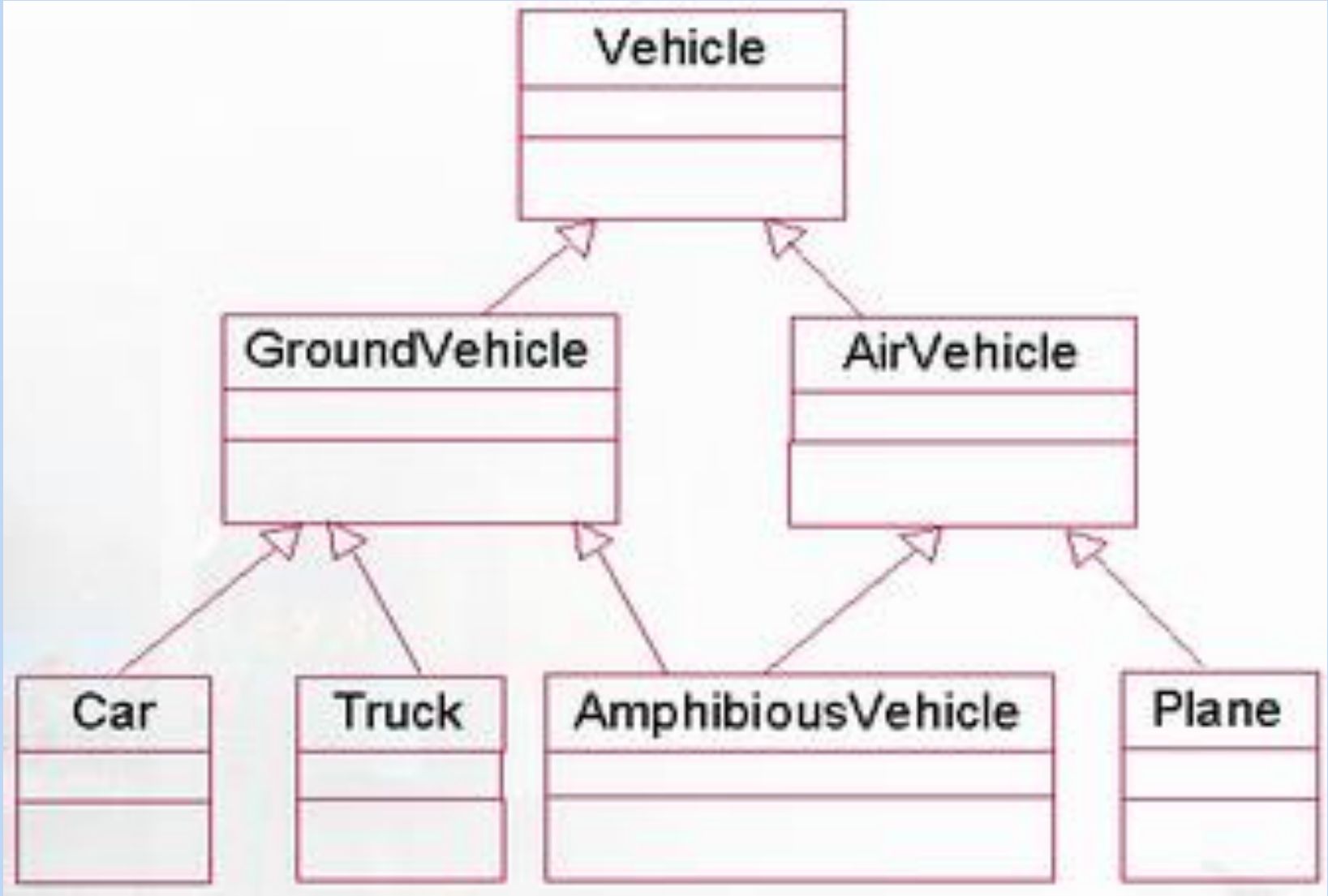
继承

- 继承（Inheritance）
  - 子类可以自动拥有父类的全部属性和服务。
- 继承简化了人们对现实世界的认识和描述，在定义子类时不必重复定义那些已在父类中定义过的属性和服务，只要说明它是某个父类的子类，并定义自己特有的属性和服务即可。
- 与父类/子类等价的其他术语
  - 一般类/特殊类、超类/子类、基类/派生类……
- 继承可分为单继承和多继承
  - 单继承是指子类只从一个父类继承
  - 多继承是指子类从多个父类继承

单继承

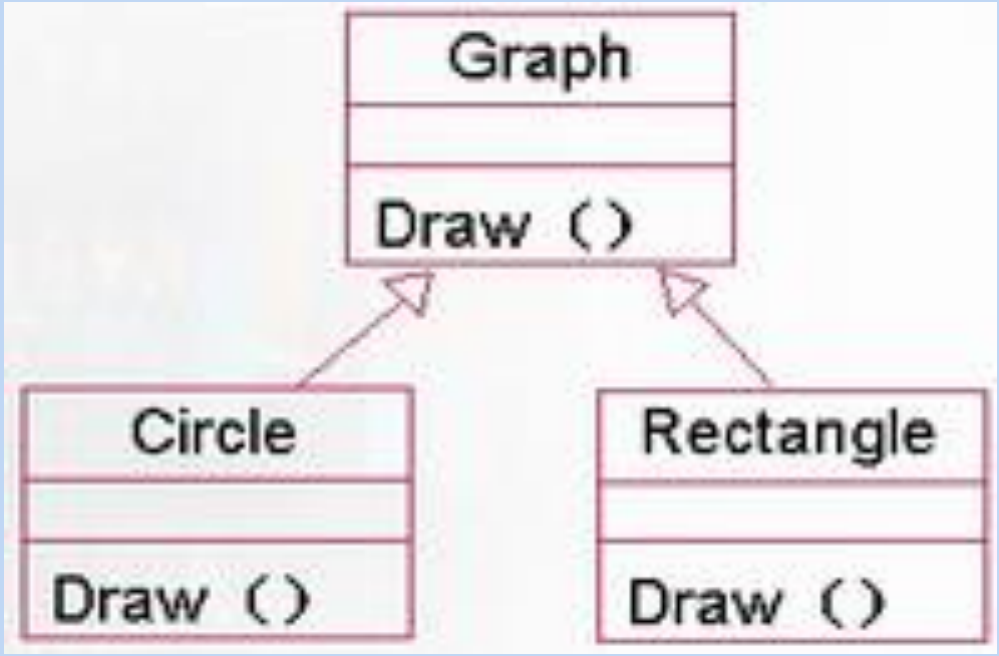

Student
name
sex
age
.....
.....
Graduate
degree
graduatedSchool
teacher
.....
.....
Undergraduate
degree
graduatedSchool
.....
.....

多继承


Vehicle
GroundVehicle
AirVehicle
Car
Truck
AmphibiousVehicle
Plane

多态性

- 多态性（Polymorphism）
  - 父类的某个方法被子类重写时，可以各自产生自己的功能行为。
- 多态性机制不但为软件的结构设计提供了灵活性，减少信息冗余，而且提高了软件的可复用性和可扩展性。



抽象类和抽象方法

- 用关键字abstract修饰的类称为abstract类（抽象类）。如：

```
abstract class A {
…
}
```

- abstract修饰的方法称为abstract方法（抽象方法）。如：

```
abstract int min(int x,int y);
```

- 抽象类的特点
  - abstract类中可以有abstract方法（抽象方法）也可以有非abstract方法。
  - abstract类，我们不能使用new运算符创建该类的对象。

抽象类与多态

abstract类只关心操作，但不关心这些操作具体实现的细节，可以使程序的设计者把主要精力放在程序的设计上，而不必拘泥于细节的实现上。

使用多态进行程序设计的核心技术之一是使用上转型对象，即将abstract类声明对象作为其子类的上转型对象，那么这个上转型对象就可以调用子类重写的方法。

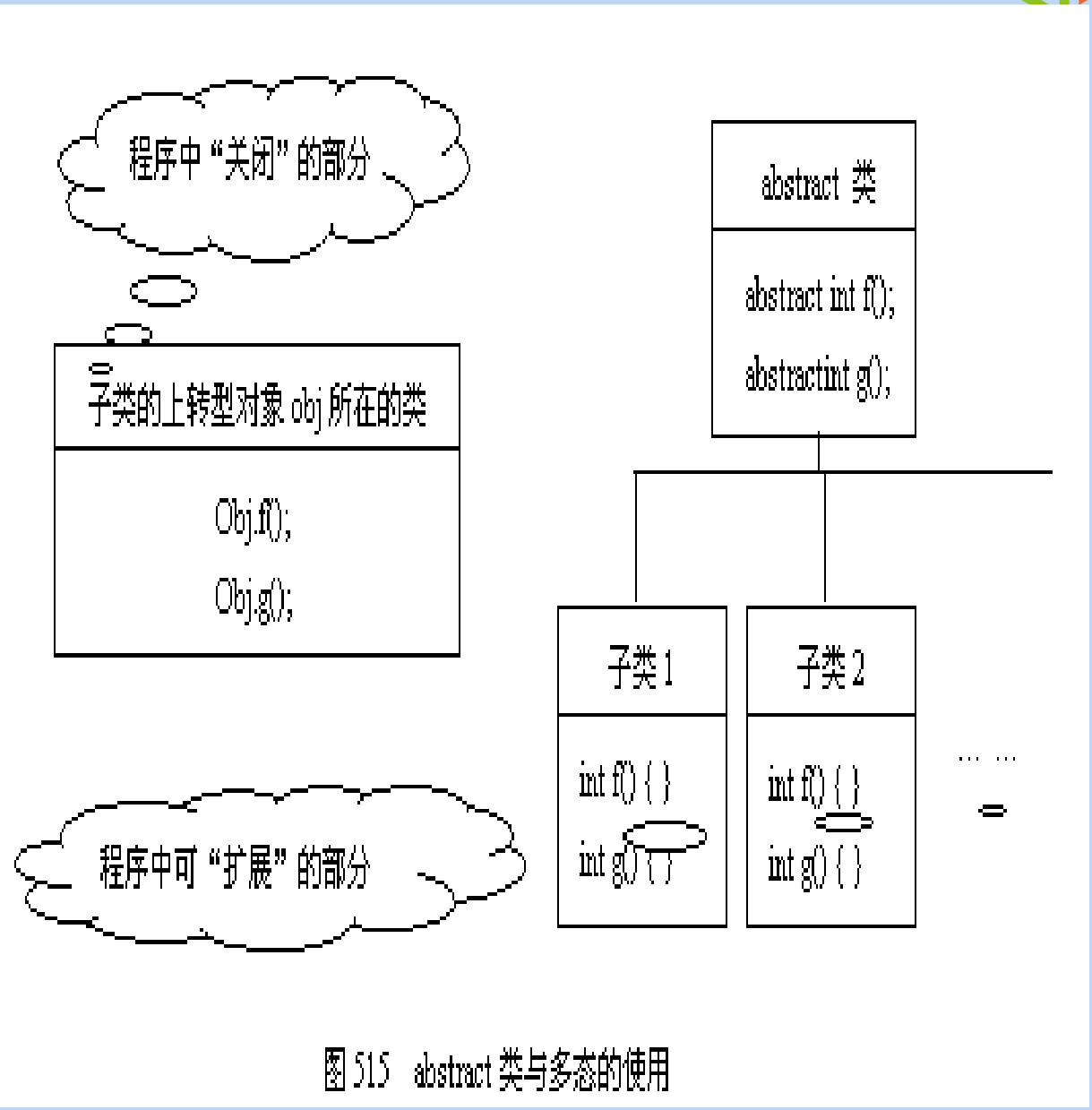


图 5.15 abstract 类与多态的使用

接口

- 为了克服Java单继承的缺点，Java使用了接口，一个类可以实现多个接口。
- 使用关键字interface来定义一个接口。接口的定义和类的定义很相似，分为接口的声明和接口体。
- 接口可以增加很多类都需要具有的功能，不同的类可以实现相同的接口，同一个类也可以实现多个接口。
- 接口只关心操作，并不关心操作的具体实现
- 接口的思想在于它可以增加很多类都需要具有的功能，而且实现相同的接口类不一定有继承关系。

接口的声明与使用

- 1．接口声明
- 接口通过使用关键字interface来声明，格式：
- interface 接口的名字
- 2．接口体
- 接口体中包含常量定义和方法定义两部分。
- 3．接口的使用
- 一个类通过使用关键字implements声明自己实现一个或多个接口。
- 4．通过import语句引入包中的接口
- import java.io.*;

接口与多态

- 可以通过在接口中声明若干个abstract方法，表明这些方法的重要性，方法体的内容细节由实现接口的类去完成。使用接口进行程序设计的核心思想是使用接口回调，即接口变量存放实现该接口的类的对象的引用，从而接口变量就可以回调类实现的接口方法

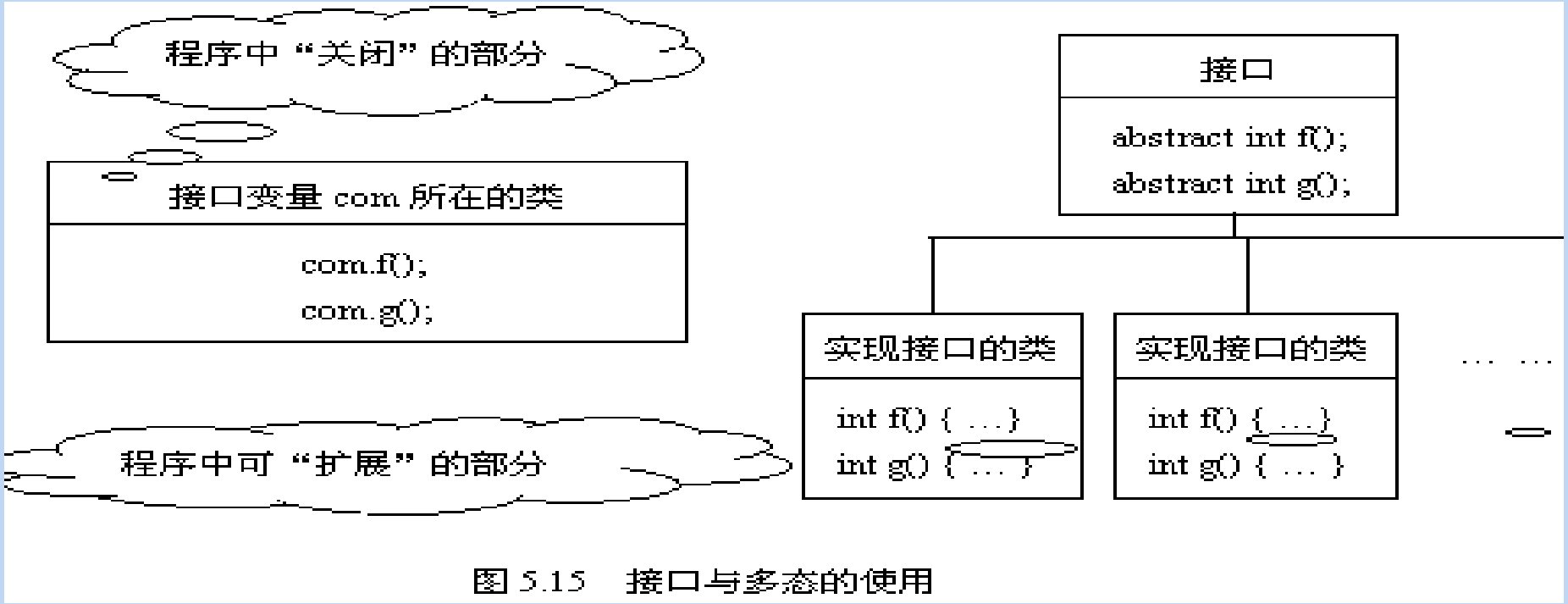


图 5.15　接口与多态的使用

抽象类与接口的比较

- 1．abstract类和接口都可以有abstract方法。
- 2．接口中只可以有常量,不能有变量；而abstract类中即可以有常量也可以有变量。
- 3．abstract类中也可以有非abstract方法,接口不可以。

05

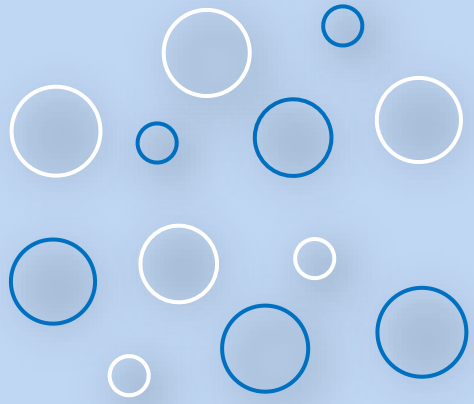
Part Five

面向对象的设计原则

软件设计中存在的问题

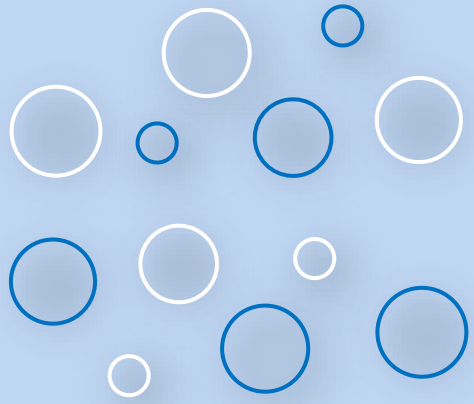
软件设计中存在的问题

- 过于僵硬（Rigidity）
  - 很难加入新功能
- 过于脆弱（Fragility）
  - 很难修改
- 复用率低（Immobility）
  - 高层模块无法重用
- 黏度过高（Viscosity）
  - 破坏原始设计框架



什么是好的设计?

- 一个好的系统设计应该有如下性质：
  - 可扩展性
  - 灵活性
  - 可插入性



设计目标与设计原则

设计目标

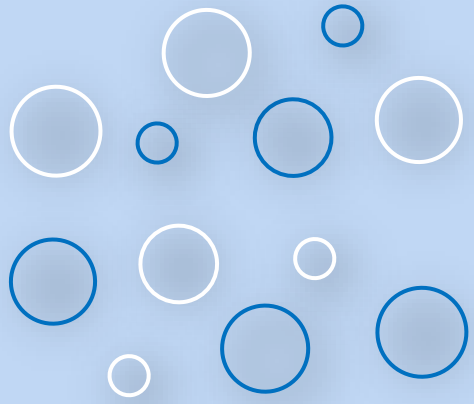
- 可扩展性（Extensibility）
  - 容易添加新的功能
- 灵活性（Flexibility）
  - 代码修改平稳地发生
- 可插入性（Pluggability）
  - 容易将一个类抽出去，同时将另一个有同样接口的类加入进来

面向对象设计原则

- OCP：开-闭原则:
  - 对可变性封装
- SRP：单一职责原则
  - 如何划分职责
- LSP：里氏代换原则
  - 如何进行继承
- DIP：依赖倒转原则
  - 针对接口编程
- ISP：接口隔离原则
  - 恰当的划分角色和接口
- CRP：合成复用原则
  - 尽量使用合成/聚合而不使用继承复用
- LoD：迪米特原则
  - 不要跟陌生人说话
- 其他设计原则

目标与原则的关系

- 可扩展性（Extensibility）
  - “开 - 闭”原则、里氏替换原则、依赖倒转原则、合成/聚合复用原则
- 灵活性（Flexibility）
  - “开 - 闭”原则、Demeter法则、接口隔离原则
- 可插入性（Pluggability）
  - “开 - 闭”原则、里氏替换原则、依赖倒转原则、合成/聚合复用原则



开放 - 封闭原则
Open-Closed Principle

OCP定义

- 软件实体（类，模块，函数，等等）应该尽可能允许扩展，同时尽可能避免被更改。
- SOFTWARE ENTITIES (CLASSES, MODULES, FUNCTIONS, ETC.) SHOULD BE OPEN FOR EXTENSION, BUT CLOSED FOR MODIFICATION.

- Bertrand Meyer 1988

OCP特征

- 可扩展（对扩展是开放的）
  - 模块的行为功能可以被扩展，在应用需求改变或需要满足新的应用需求时，我们可以让模块以不同的方式工作
- 不可更改（对更改是封闭的）
  - 这些模块的源代码是不可改动的
  - 任何人都不许修改模块的源代码
- 自相矛盾?

太玄经

知固而不知革，物失其则
知革而不知固，物失其均

- **固：close for modification**
- **革：open for extension**

太玄经

- 一个系统对修改关闭，就是”固”
- 而一个系统对扩展开放，就是”革”
- 一个系统不可拓展，就会“物则失则”，或者说系统无法发展
- 而一个系统动则需要修改，便会“物失其均”，也就是失去其重心

OCP关键

- OCP的关键是

  抽象！
- 由抽象可以预见所有可能的扩展（闭）:模块可以操作一个抽象体，由于模块依赖于一个固定的抽象体，因此它对修改是封闭的(closed for modification)
- 由抽象可以随时导出新的类（开）:同时，通过从这个抽象体派生，又可扩展此模块的行为和功能(open for extension)
- 符合OCP原则的程序只通过增加代码来变化而不是通过更改现有代码来变化，因此这样的程序就不会引起象非开放-封闭的程序那样对变化的连锁反应

OCP实例

- 思考：如何在程序中模拟用手去开门和关门?
- 行为：
  - 开门
  - 关门
  - 判断门的状态

```
public class Door1{
    private boolean _isOpen=false;
    public boolean isOpen(){
        return _isOpen;
    }
    public void open(){
        _isOpen = true;
    }
    public void close(){
        _isOpen = false;
    }
}
```

```
public class Hand1 {
    public Door1 door;
    void act() {
        if (door.isOpen())
            door.close();
        else
            door.open();
    }
}
```

主程序

```
public class Main1 {
    public static void main(String[] args) {
        Hand1 myHand = new Hand1();
        myHand.door = new Door1();
        myHand.act();
    }
}
```

新的问题

- 需要手去开关抽屉，冰箱……？
- 我们只好去修改程序……

抽屉

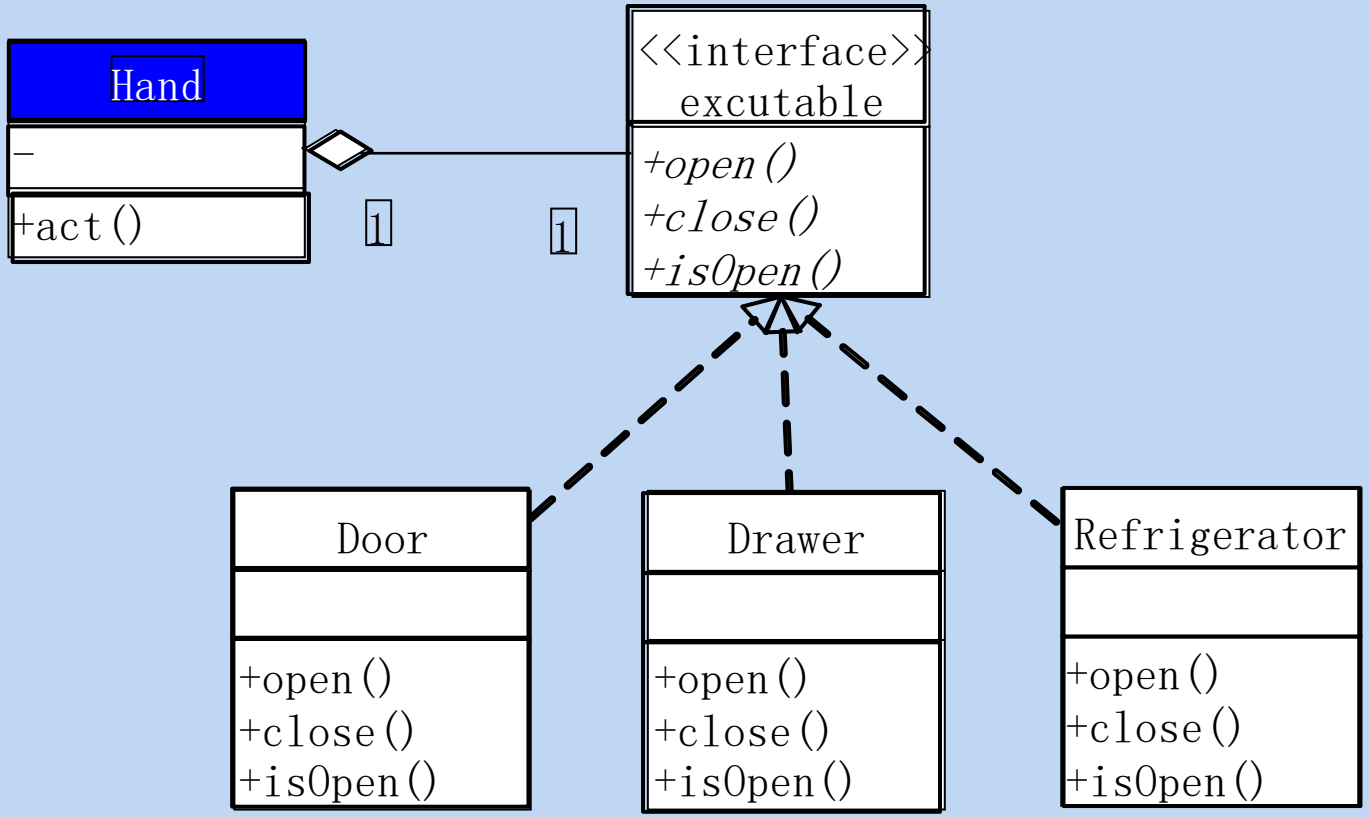
```
public class Drawer1 {
    private boolean _isOpen=false;
    public boolean isOpen(){
        return _isOpen;
    }
    public void open(){
        _isOpen = true;
    }
    public void close(){
        _isOpen = false;
    }
}
```

```
public class Hand1 {
    public Door1 door;
    public Drawer1 drawer;
    public int item=1;
    void act() {
        switch (item){
            case 1:
                if (door.isOpen())
                    door.close();
                else
                    door.open();
                break;
            case 2:
                if (drawer.isOpen())
                    drawer.close();
                else
                    drawer.open();
                break;
        }
    }
}
```

主程序

```
public class Main1 {
    public static void main(String[] args) {
        Hand1 myHand = new Hand1();
        myHand.door = new Door1();
        myHand.item = 1;
        myHand.act();
    }
}
```

符合OCP的设计


Hand
-
+act()
1
1
<<interface>>
excutable
+open()
+close()
+isOpen()
Door
+open()
+close()
+isOpen()
Drawer
+open()
+close()
+isOpen()
Refrigerator
+open()
+close()
+isOpen()

“可开关”的接口

```
public interface Excutable {
    public boolean isOpen();

    public void open();

    public void close();
}
```

门（新的）

```
public class Door2 implements Excutable {
    private boolean _isOpen = false;
    public boolean isOpen() {
        return _isOpen;
    }
    public void open() {
        _isOpen = true;
    }
    public void close() {
        _isOpen = false;
    }
}
```

抽屉（新的）

```
public class Drawer2 implements Excutable {
    private boolean _isOpen = false;
    public boolean isOpen() {
        return _isOpen;
    }
    public void open() {
        _isOpen = true;
    }
    public void close() {
        _isOpen = false;
    }
}
```

手（新的）

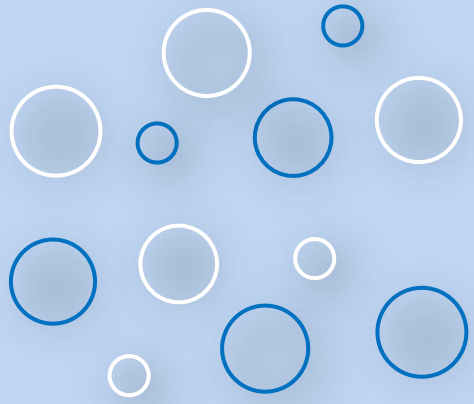
```
public class Hand2 {
    public Excutable item;

    void act() {
        if (item.isOpen())
            item.close();
        else
            item.open();
    }
}
```

主程序（新的）

```
public class Main2 {
    public static void main(String[] args) {
        Hand2 myHand = new Hand2();
        myHand.item = new Door2();
        myHand.act();
    }
}
```

选择性封闭

- 通常情况下，没有任何一个大的程序能够做到100%的封闭
- 一般来讲，无论模块是多么的“封闭”，都会存在一些无法对之封闭的变化
- 既然不可能完全封闭，因此就必须选择性地对待这个问题
- 也就是说，设计者必须对于他（她）设计的模块应该对何种变化封闭做出选择



单一职责原则

Single Responsibility Principle

SRP定义

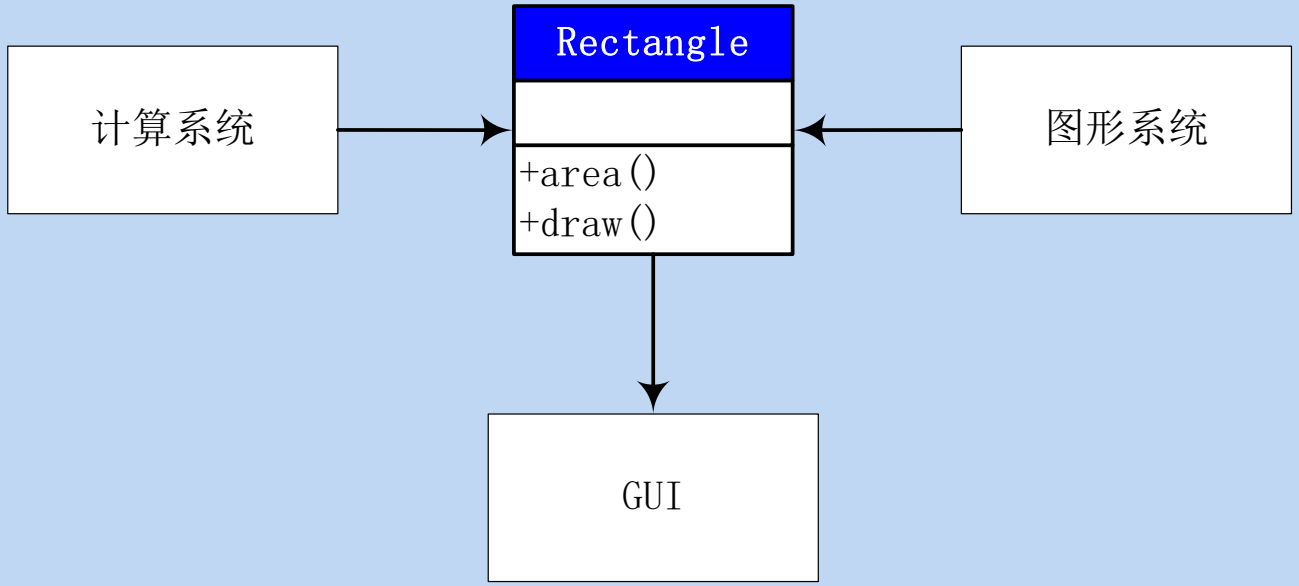
- 就一个类而言，应该仅有一个引起它变化的原因

SRP实例

- 一个GUI系统的设计
- 要求：
  - 可以计算矩形的面积
  - 并在屏幕上显示

脆弱的设计

- 几何计算系统调用Rectangle计算面积
- GUI系统调用Rectangle绘制在屏幕上



出现的问题

- 编译几何计算系统时还需要编译进图形代码…
- 需要更换显示系统时还需要重新测试所有几何计算系统…
- 解决方法：将计算和绘制的职责分别放入CulRectangle和GraphRectangle中

感谢您的观看