

《面向对象设计与构造》课程

Lec1-对象与对象化编程(上)

2018

OO课程组

北京航空航天大学

内容提要

- 课程介绍
- 过程式程序回顾
- 为什么引入对象
- 对象化程序的构成
- 对象是什么
- 作业

课程介绍



- 三个关键词
 - 设计(design)与构造(build)==》工程化开发
 - 面向对象==》系统化的思维方式
- 讨论：软件与程序的区别
- 讨论：如何说明你所写程序有多好？
- 课程目标：掌握以工程化方法来开发高质量复杂软件系统的能力
 - 工程化方法：综合分析软件功能和性能约束，综合考虑相应约束进行设计和实现，并能使用测试和逻辑分析等手段进行综合验证和优化
 - 高质量：能够使用技术手段来表明/论证所开发的软件质量是否满足要求

数据结构设计
与应用能力

算法设计与
优化能力

基于约束的需
求分析能力

基于需求的规
格化设计能力

线程安全
设计能力

关注性能的实
现与测试能力

基于规格的程序
正确性论证能力

微型团队
协同能力

体系化的课程



“昆仑课程”
(二下春季)



“训练营课程”
(一下暑期)



“补给站课程” (二下暑期)

“昆仑课程” 知识点设置

对象与类

OO程序构造

层次化设计

对象运行机制

线程交互

线程安全设计

过程规格设计

类规格设计

设计原则

自动化测试

基于规格的程序
正确性证明

程序的模型化
表示

“昆仑课程” 核心规则



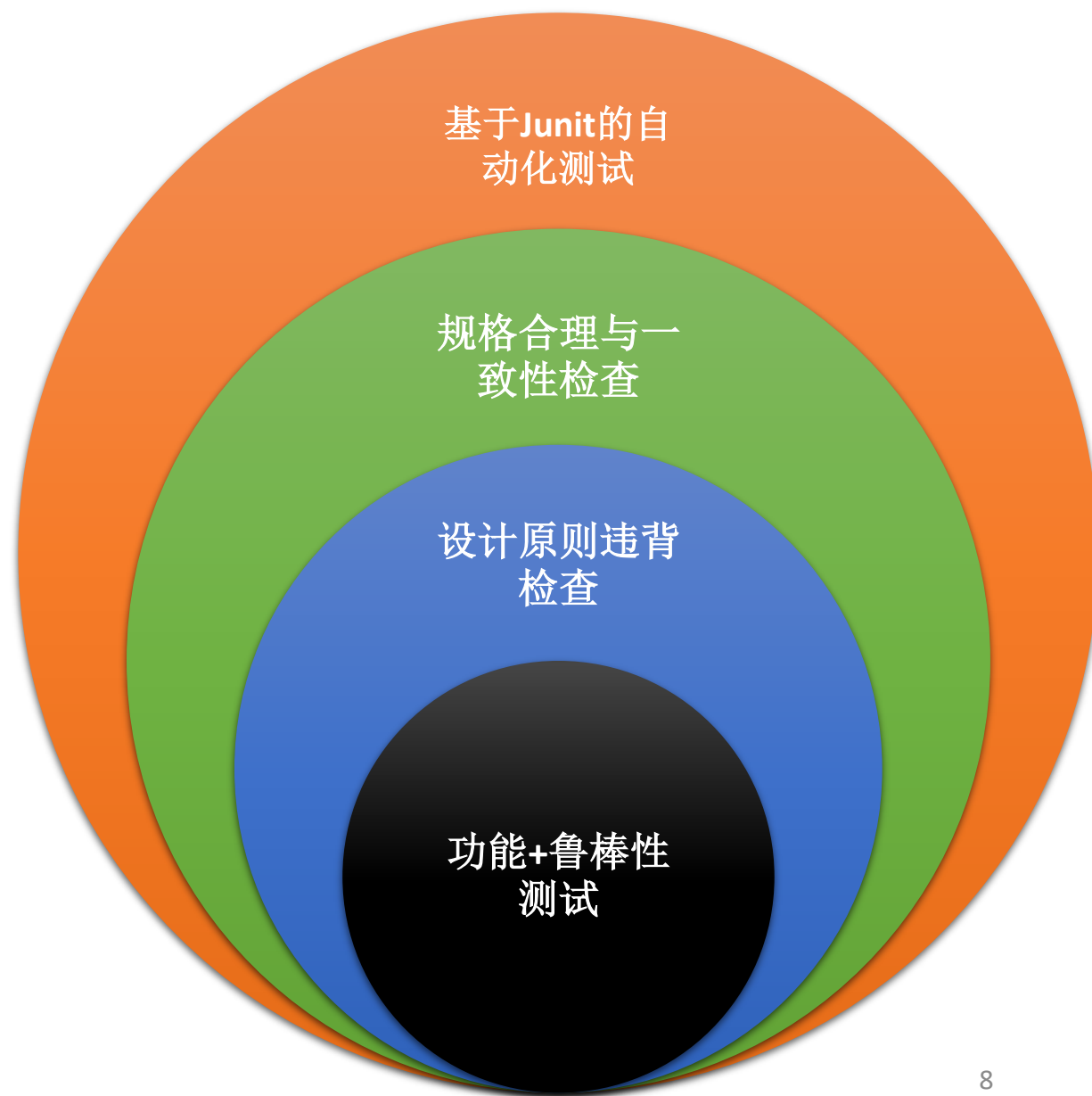
- **32(授课)+16(实验)**学时，3学分，必修课
- 内容分4个模块，每个模块包括4次授课和2次实验
 - 3次/周介绍新内容
 - 每次一个程序作业、每次一个测试作业
 - 1次课程作业问题分析
 - 对各自的程序问题和测试问题进行总结分析，撰写技术博客
 - 2次实验围绕单元教学内容进行实践训练和分析
 - 每次实验当堂完成实验和在系统中完成相应报告
- 平衡与综合的测试
 - 基准公共测试
 - 竞争性互测

“昆仑课程” 核心规则

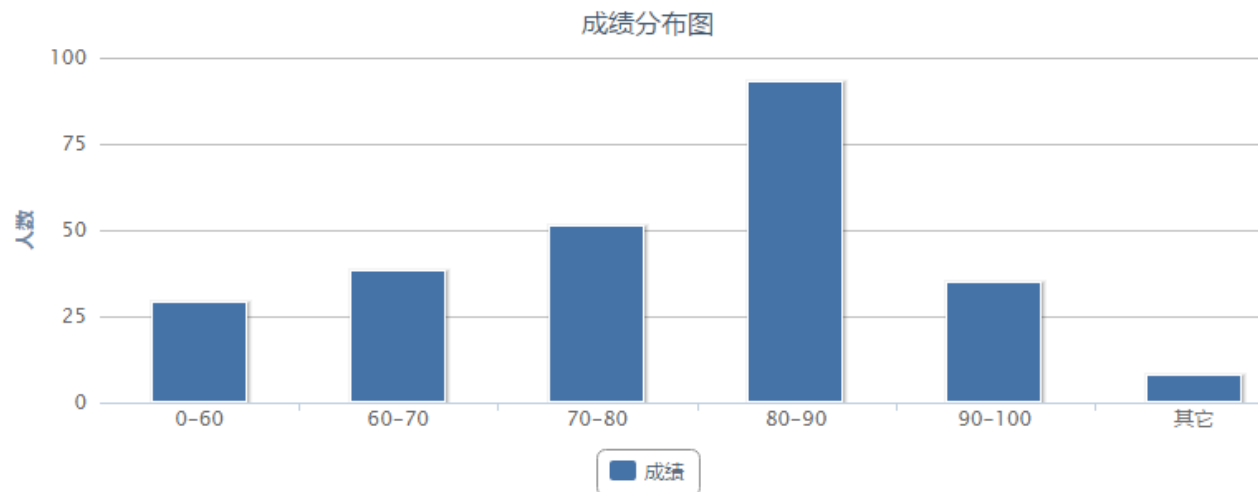
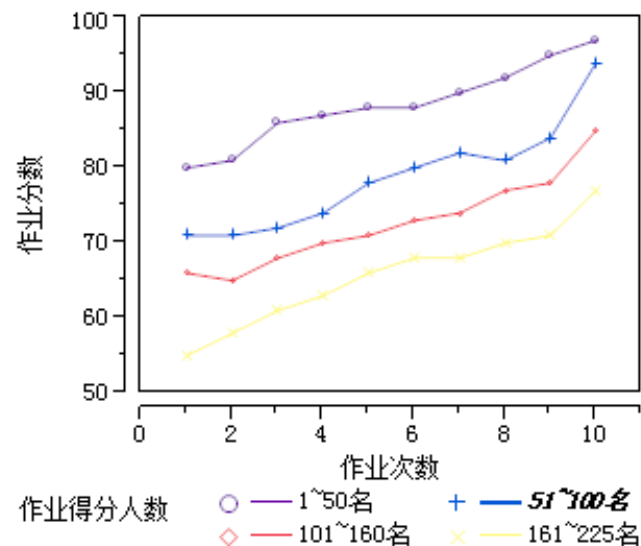
- 采用Java语言
 - OO概念支持、类型安全、（编译器）强大的静态检查能力、跨平台
- 如果不会Java语言
 - 使用C++/C#，会有一定的副作用
 - 学习使用Java
- 成绩评定：综合排序
 - 作业成绩：70%
 - 作业完成质量和测试
 - 实验成绩：30%
 - 完成度和质量

强调测试的课程作业

- Common Testing
 - 针对作业的功能和性能要求，精心设计的测试用例
- Double Blind Testing
 - 你不知道谁将测试你的程序
 - 你不知道你测试的是谁的程序
- Comprehensive Scoring
 - 作业提交情况
 - 通过的公共测试用例数
 - 被发现的bug数
 - 发现的bug数



“昆仑课程”的能力目标



缺考：8人

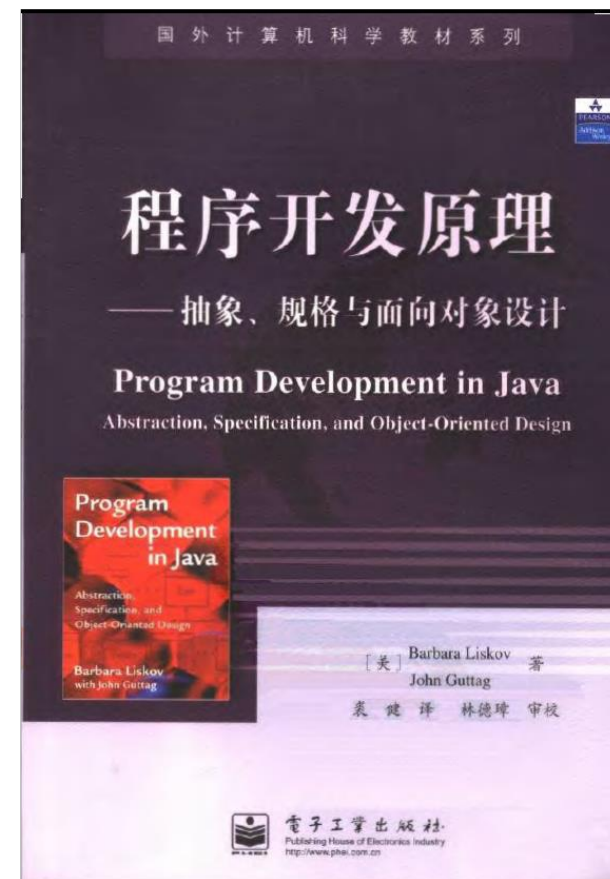
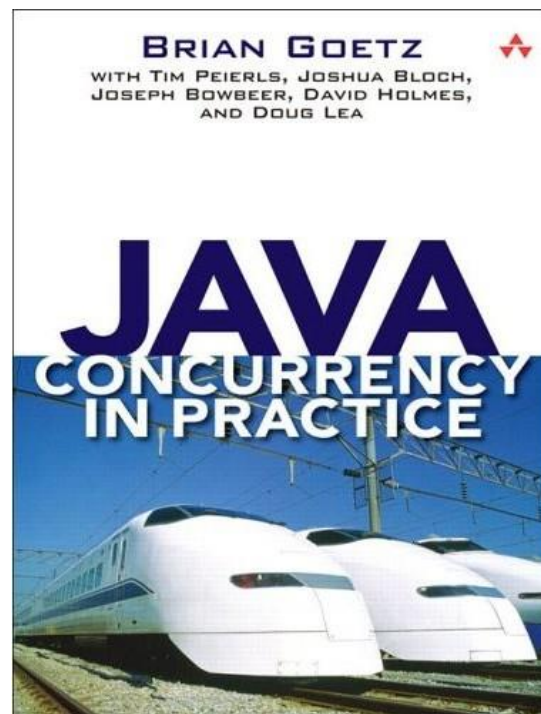
“昆仑课程” 参考材料

- 教材

- 程序开发原理—抽象、规格与面向对象设计
(Barbara Liskov, John Guttag, *Program Development in Java: Abstraction, Specification, and Object-Oriented Design*)
- Java Concurrency in Practice

- 互联网

- 百科
- Jdk guideline
- Stackoverflow.com
- 技术博客



“昆仑课程” 核心规则

- 采用竞争性学习模型(competitive learning)
 - 竞争性任务：互测、bug申诉、积分排位
 - 竞争关系：基于排位区域的互测关系
 - 成绩规则：你会及时了解自己的排位，最后一名也可能是80分！
- 拥抱竞争(embrace competition)
 - 无法回避，终身相随
 - 拥抱→理解→掌控
- 鼓励合作，但严禁抄袭
 - 通过博客分享经验

2018年的七大新举措

- 引入OJ(Online Judge)，进一步减少测试中的人为因素和个人PC的配置差异
- 引入Git，掌握规范化的主流开发过程和代码管理
- 引入关于软件输入划分的分类树，正向引导你的测试思维和进一步规范bug报告
- 引入基于分类树的互测bug报告限制机制，每个叶子节点最多只能有一个bug报告
- 引入关于项目指导书的MOOC，助教结合课程知识点和训练要点帮助大家解读指导书（需求），提升大家的需求理解能力
- 引入基于云的开发环境，让同学们与世界前沿的开发模式保持同步
 - 华为云、Docker、IDE in Cloud、与git的集成、...
- 引入企业助教，专门点评和推介同学们所撰写的总结博客(cnblogs)
 - 上海七牛创始人许式伟携公司资深工程师
 - 老师们也会在全国性的教学群中推介优秀博客

“昆仑课程” 核心规则



- 作业抄袭？
 - 作业deadline后一个小时内发布抄袭检测结果
 - 确认抄袭就相当于作业无效，累积发现五次无效作业，取消作业成绩！
 - 五次以内的每次无效作业，自动获得当次作业中去除基准分的全年级最差分
- 有效作业规则
 - 可编译、可运行、至少通过一个正常功能型的公共测试用例
- 通过率？
 - “补给站”已准备好欢迎有困难的同学！
 - 及格准则：抄袭次数=0 && 无效作业次数≤3 && 整体的公测通过率≥50% && 无故旷课次数=0 && 实验完成次数≥5
- 作业提交自动从你的git库中提取代码，不需要你手动上传
 - 确保每次代码更新都要提交到git库

“昆仑课程” 核心规则

- 程序测试与结果提交注意事项
 - 公共测试会在OJ中自动完成，可以查看结果
 - 竞争性测试可以使用任何合理的测试手段
 - 在OJ中输入你的测试用例
 - 在系统中提交bug
 - 如果分类树中相应的分支已经报告了bug，但是确信自己所发现的bug和那个分支确实有不同之处，则需要扩展分类树，然后把相应的bug报告出来
 - 分类树的分支扩展数 ≤ 2
 - 错误分类：Crash(触发异常，程序崩溃，权重4)；Wrong(错误输出结果，权重2)；Incomplete(不完整或部分正确的输出结果，权重1)
 - 扩展了分类树的bug一旦被确认，测试者会额外获得4分加分，但被测者不会被额外扣分。如果是无效分类扩展，则测试者获得1分减分，被测者不受影响。

“昆仑课程” 核心规则

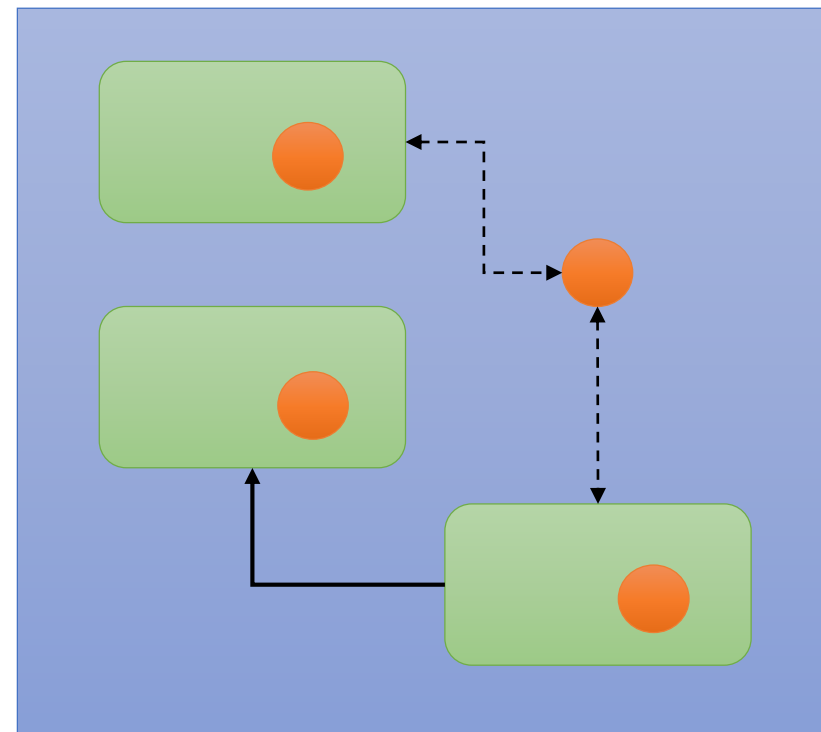
- 采用积分排位制
- 一次作业的积分
 - 基准分（根据作业难度设定，无效作业不能获得该基准分）(+)
 - 通过的公共测试用例数及加权(+)
 - 你的程序被发现了多少bug(-)
 - 你发现了别人程序的多少bug(+)
- 如何开展有效的测试？
 - 阅读程序代码，进行逻辑分析
 - 根据输入规格和分类树，设计相应的输入来运行程序
 - 关注：正常值、边界值、异常值
- 如何有效申诉？
 - 及时提起申诉
 - 有理有据
 - 及时回复交流，态度诚恳

“昆仑课程” 核心规则

- 作业提交时间(有可能会根据具体情况作出调整)
 - 周三19:00前完成作业，系统会自动从你的git库提取代码
 - 周四8:00前完成测试任务分配
 - 周四22:00前提交互测结果
- 没按时提交？
 - 如果git库中没有代码，或者提取的代码不符合有效性规则：无效作业！
- 存活秘笈
 - No.1 合理安排时间，克服拖延症。
 - No.2 对自己程序做充分测试，并记录下所用测试数据，再去测试别人。
 - No.3 不放弃，坚持下去。
 - No.4 走正道。

过程式程序回顾

- 结构化
 - 功能结构：模块、函数
 - 数据结构：类型、变量(全局、局部)
 - 组合结构（交互机制）：函数调用、变量共享
- 面向过程(procedure)
 - 是一种自然的思维方式：按照“自然过程/业务流程”来设计程序
 - 过程分解/业务分解
 - 提取公共过程



过程式程序回顾

- 模块表现形式
 - 物理意义上的模块：exe、lib、dll文件等
 - 逻辑意义上的模块：多个相关函数的集合体（.c文件+.h文件）
- 函数
 - 具有一定计算能力、相对独立的编程单位
 - 公共功能函数：围绕数据结构实施所需的计算和处理，如字符串处理、栈和队列处理函数等
 - 特定功能函数：直接源自于软件功能分解得到的函数，如学生注册、输入/输出函数等

过程式程序回顾

- 公共功能函数 vs 特定功能函数
 - 功能实现
 - 一般化 vs 特定程序功能
 - 调用场景
 - 不确定 vs 确定
 - 易变性
 - 不随程序功能变化而变化 vs 随程序功能变化而变化
 - 重用性
 - 高 vs 低

过程式程序回顾

- 函数调用
 - 形参与实参的匹配
 - 返回值的处理
- 变量
 - 全局变量：多个函数要使用 and 处理的变量，如电梯系统的电梯状态变量。
 - 局部变量：一个函数内部要处理的数据表示。
 - 临时变量：便于代码编写的一些临时变量，如循环变量、中间计算结果存储等

三者之间有什么关系？

过程式程序回顾

- 丰富的类型
 - 原子类型
 - 结构类型(struct)
 - 联合类型(union)
- 丰富的数据组织与使用方式
 - 数组、列表
 - 树与图
 - 指针

为什么引入对象

- 编码视角
 - 为什么多个函数需要共享访问数据（变量）？
 - 这些函数之间具有逻辑“聚合”的特性
 - 如何处理一个函数需要使用之前运行所产生的一些中间数据？
 - 增加全局变量
 - 或者，使用外部存储
 - 如何管理逻辑相关的函数+变量？
 - 聚合在一个文件中

为什么引入对象

- 程序设计视角
 - 需要一种手段来封装逻辑相关的函数和数据
 - 需要整合了处理的类型层次
 - 尽可能避免使用全局变量
- 程序思维视角
 - 处理流程
 - 数据及其状态变化的管理
 - 按照数据的层次化抽象

面向对象程序的构成

- 类
 - 属性(数据)、操作及其实现
 - 作用域
- 接口
 - 抽象操作
- 关系
 - 继承：类型层次+重用
 - 关联：数据聚合+调用
 - 实现：为多种数据抽象提供统一接口
- 入口类
 - 提供入口函数main（静态函数）

包

模块层次
作用域

面向对象程序的构成

- 过程式程序与面向对象程序的特点对比

过程式

- 强调函数分解
- 程序由函数组成
- 运行时由函数和数据表示
- 函数之间共享全局数据
- 函数之间传递数据

面向对象式

- 强调数据抽象
- 程序由类组成
- 运行时由对象表示
- 数据得到隐藏和保护
- 对象之间通过消息交互

面向对象程序的构成

- 以类作为基本的编程单位
- 类封装了数据和函数
- 类之间协作完成程序的功能
 - 有哪些协作方式？

```
public class Num {  
    // class providing useful numeric routines  
  
    public static int gcd (int n, int d) {  
        // REQUIRES: n and d to be greater than zero  
        // the gcd is computed by repeated subtraction  
        while (n != d)  
            if (n > d) n = n - d; else d = d - n;  
        return n;  
    }  
  
    public static boolean isPrime(int p) {  
        // implementation goes here  
    }  
}
```

对象与类

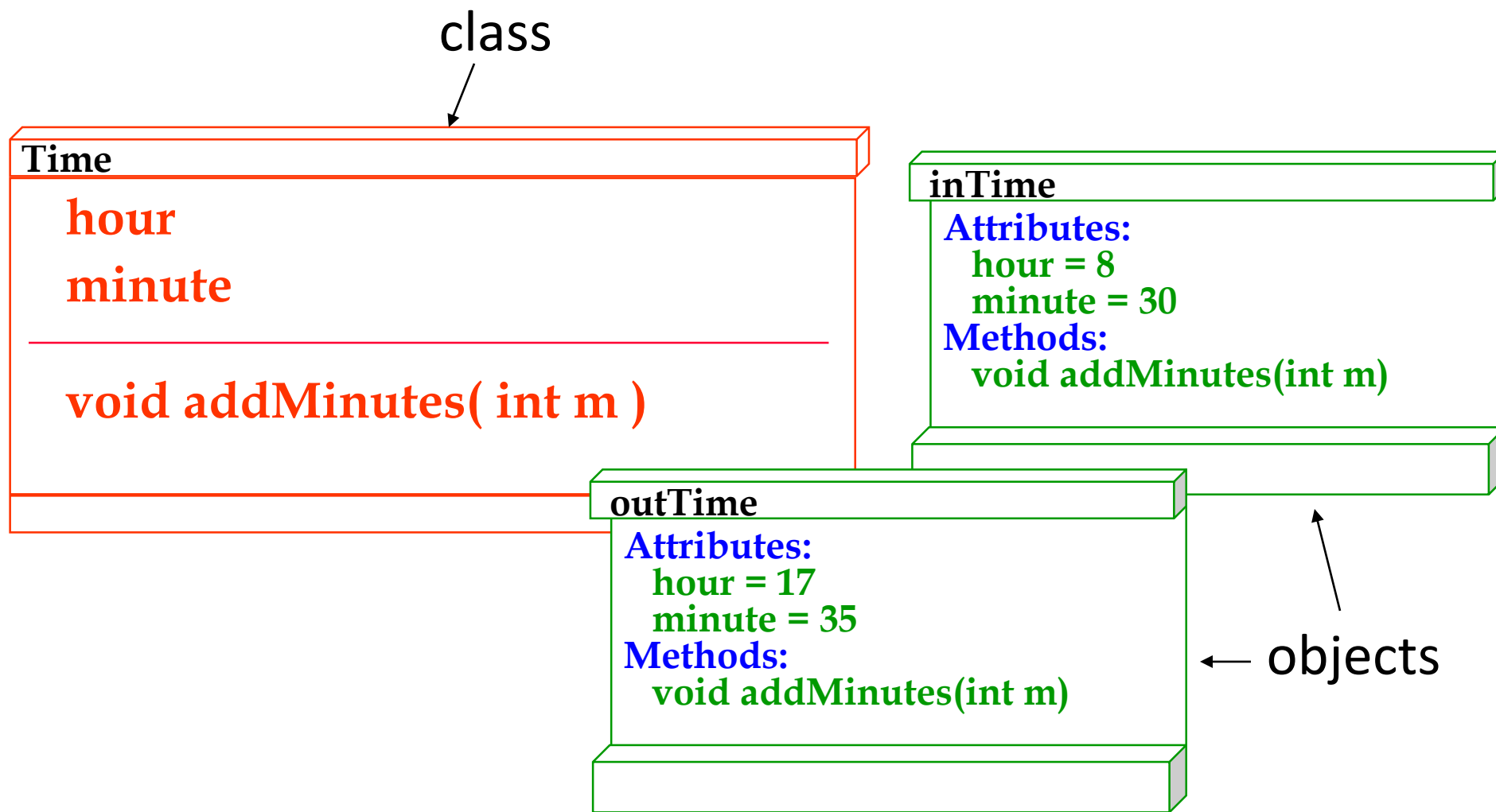
- 在面向对象程序中，我们称对象是类的实例化结果
 - 对象是运行时概念
 - 类是规格概念
- 类是通过关键词class定义的一个程序单位
 - `public class A {...}`
- 对象是方法中定义的变量（类型为某个类）
 - `A a = new A(...);`
- 一个对象可以通过多个变量来引用
 - `A b = a;`



对象与类

```
class Time {  
    private int hour, minute;  
  
    public Time (int h, int m) {  
        hour = h;  
        minute = m;  
    }  
  
    public void addMinutes (int m) {  
        int totalMinutes =  
            ((60*hour) + minute + m) % (24*60);  
        if (totalMinutes<0)  
            totalMinutes = totalMinutes + (24*60);  
        hour = totalMinutes / 60;  
        minute = totalMinutes % 60;  
    }  
}
```

对象与类




对象与类

- 类是一个用于构造对象的模板
 - 规定了对象拥有的数据及其类型
 - 规定了对象能够执行的动作
 - 规定了对象状态的变化空间
- 如果在程序中定义了一个类A，就意味着可以在另一个类B中构造A的变量以实现B的方法/功能
- 每个类都应该提供相应的构造器，用来在构造对象时初始化和设置对象的初始状态

对象与类---构造器

```
class Time {  
    private int hour, minute;  
  
    public Time (int h, int m) {  
        hour = h;  
        minute = m;  
    }  
  
    public void addMinutes (int m) {  
        int totalMinutes =  
            ((60*hour) + minute + m) % (24*60);  
        if (totalMinutes<0)  
            totalMinutes = totalMinutes + (24*60);  
        hour = totalMinutes / 60;  
        minute = totalMinutes % 60;  
    }  
}
```

constructor for Time



对象与类

- 对象是一个具有计算能力的实体
 - 封装(*Encapsulate*) 其状态，对外部屏蔽细节
 - 状态由对象所有属性变量的取值联合确定
 - 例如Time类中的hour和minute属性，(22,10)表示晚上时间状态，(11,30)则表示白天时间状态
 - 能够在相应状态上执行动作即方法 (*method*)
 - 在不同状态下执行方法的效果可能会不同
 - 通过消息传递机制与其他对象交互(*message passing*)
 - 消息: *object.method(p1,p2,...,pm)*
 - 和函数调用存在本质上的不同(后面会解释)

对象与类---构造对象

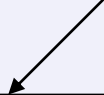
```
class Time {  
    private int hour, minute;  
  
    public Time (int h, int m) {  
        hour = h;  
        minute = m;  
    }  
    ...  
}
```

```
Time inToWork = new Time(8, 30);  
Time outFromWork = new Time(17, 35);
```

对象与类---执行方法

```
class Time {  
    private int hour, minute;  
    public Time (int h, int m) {  
        hour = h;  
        minute = m;  
    }
```

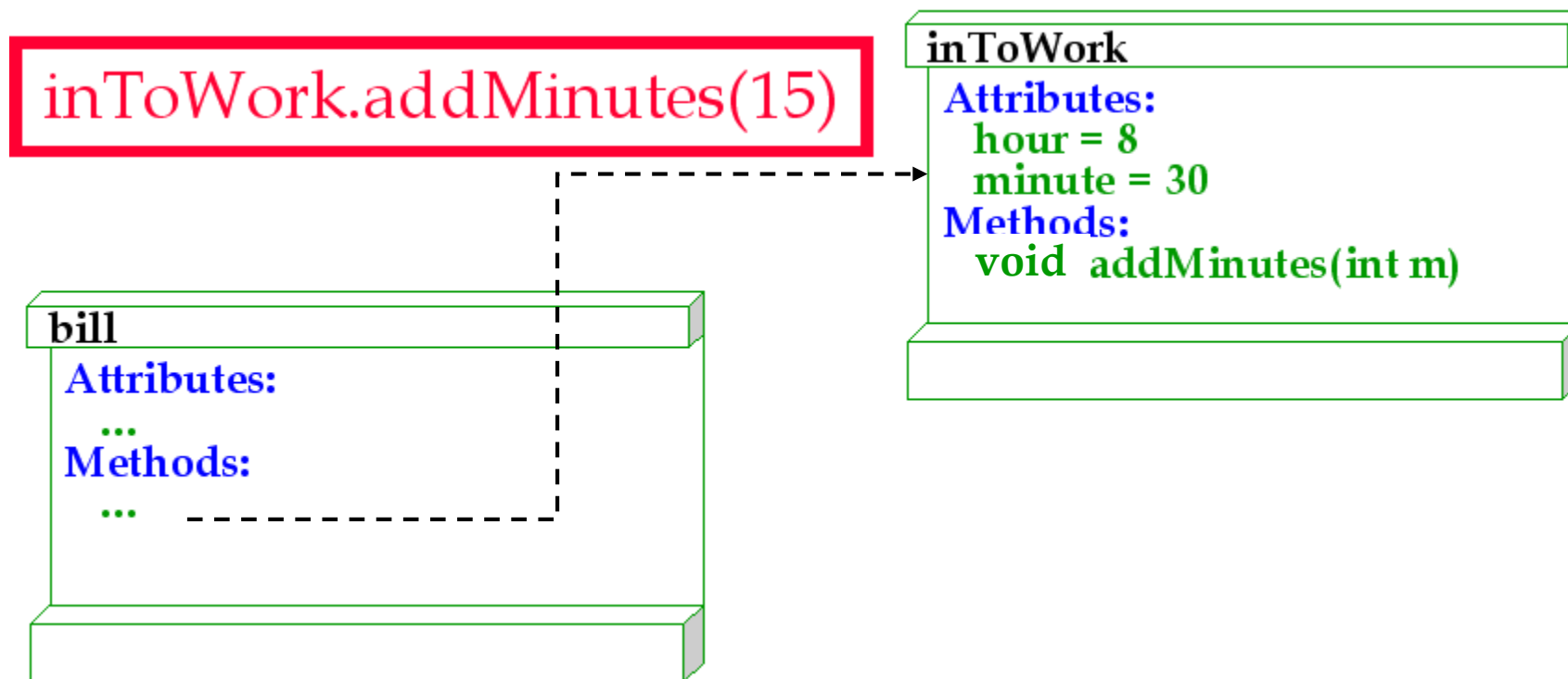
该方法能够根据对象状态来进行相应的计算



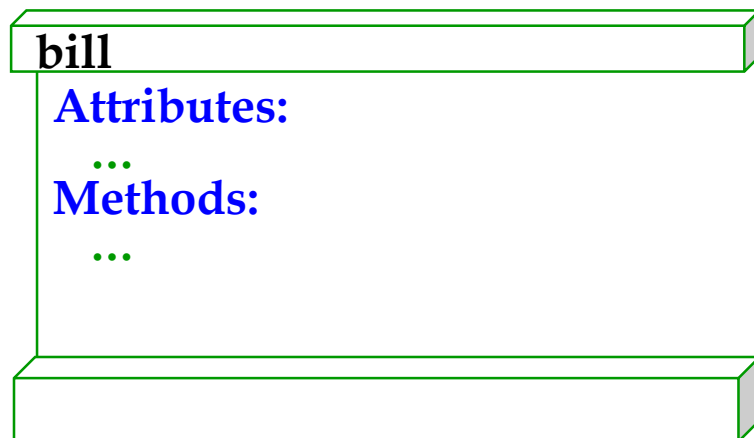
```
        public void addMinutes (int m) {  
            int totalMinutes =  
                ((60*hour) + minute + m) % (24*60);  
            if (totalMinutes<0)  
                totalMinutes = totalMinutes + (24*60);  
            hour = totalMinutes / 60;  
            minute = totalMinutes % 60;  
        }
```

```
}
```

对象与类---对象交互



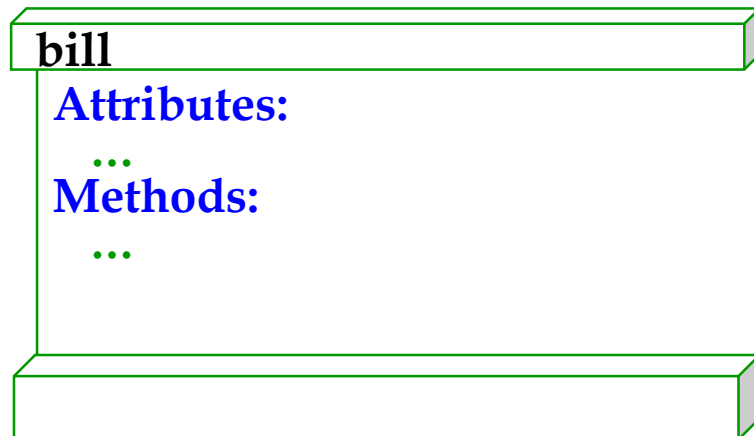
对象与类---对象交互



对象与类---对象交互

假设在**bill**的某个方法中有下面的代码:

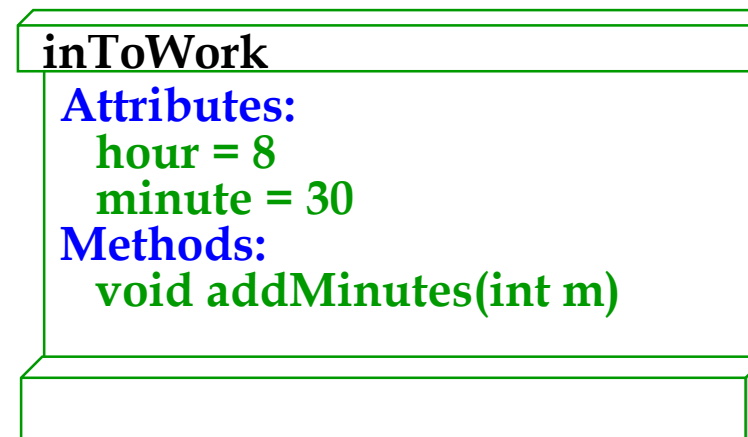
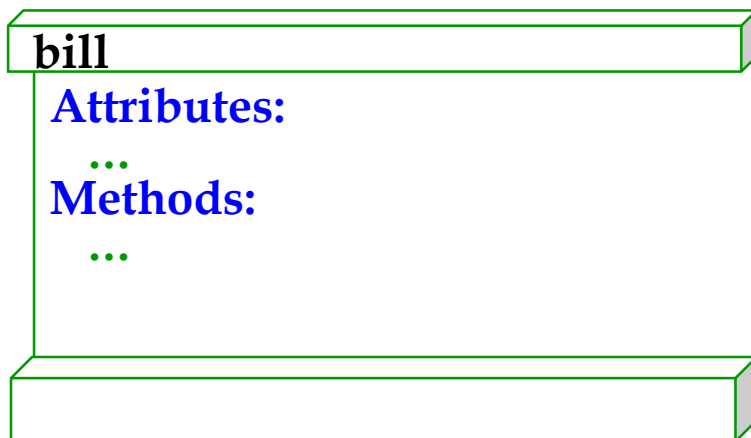
```
Time inToWork = new Time(8, 30);  
inToWork.addMinutes(15);
```



对象与类---对象交互

假设在bill的某个方法中有下面的代码:

```
➡ Time inToWork = new Time(8, 30);  
   inToWork.addMinutes(15);
```

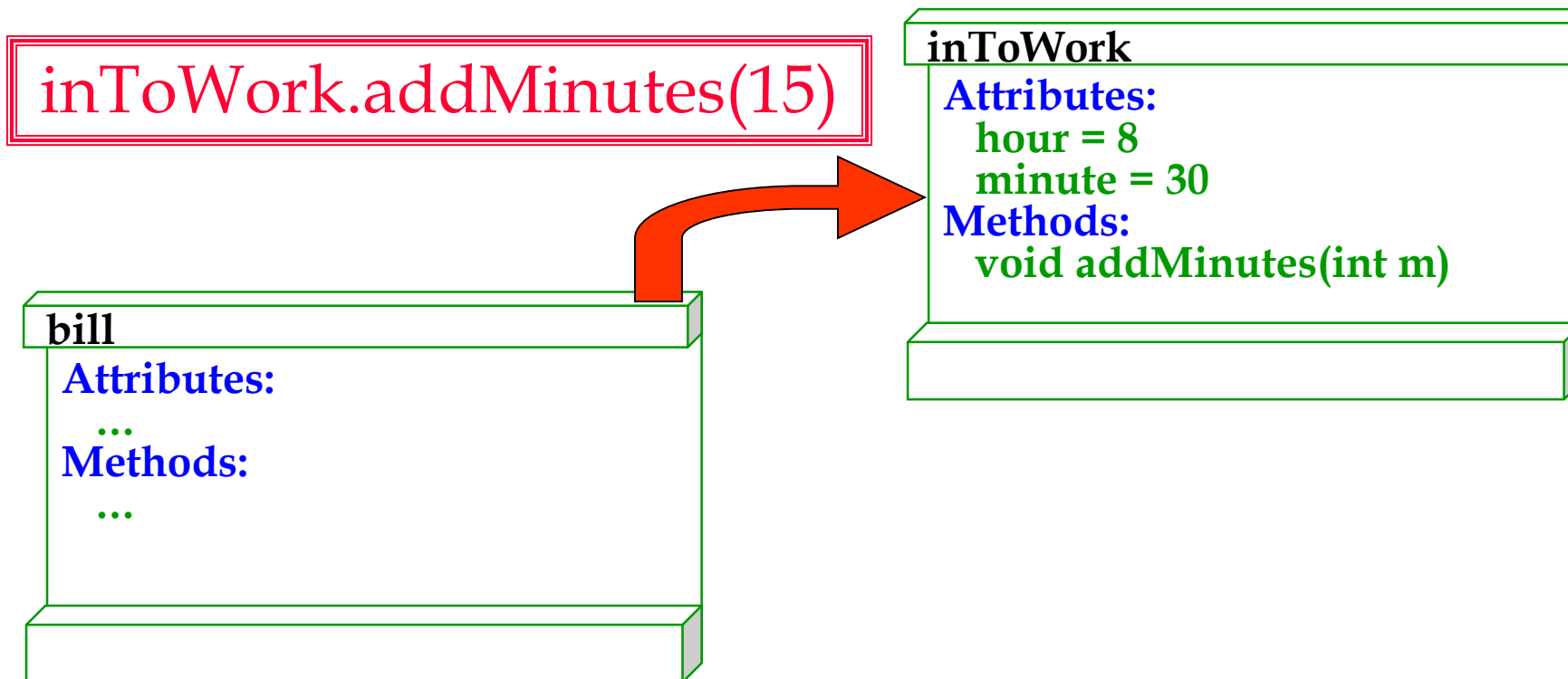


对象与类---对象交互

假设在bill的某个方法中有下面的代码:

```
Time inToWork = new Time(8, 30);
```

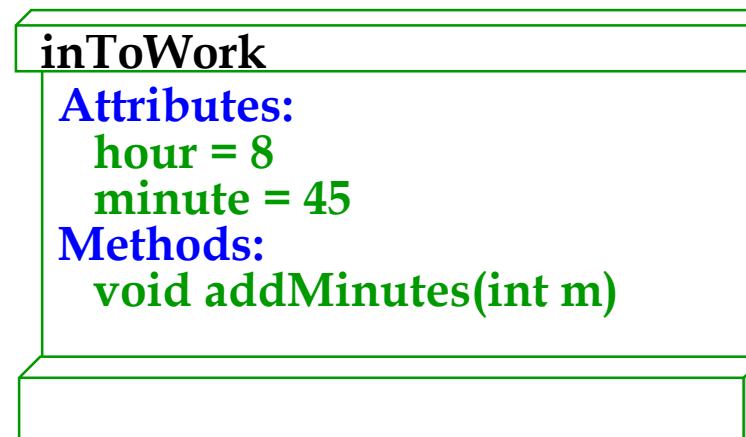
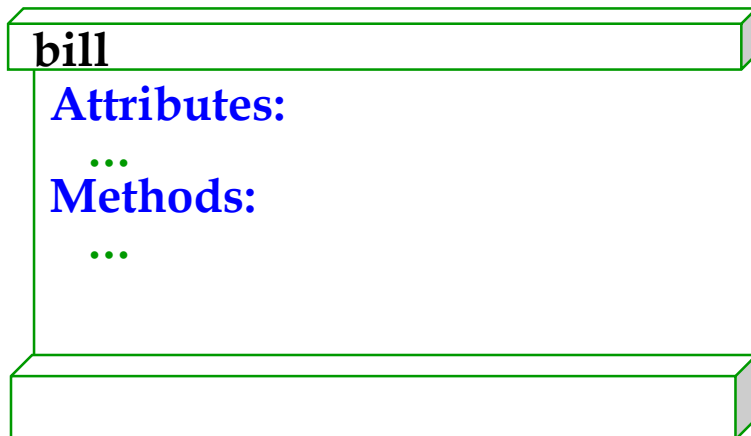
```
➡ inToWork.addMinutes(15);
```



对象与类---对象交互

假设在bill的某个方法中有下面的代码:

```
Time inToWork = new Time(8, 30);  
inToWork.addMinutes(15);
```



对象与类---类的结构

```
class name [extends ***][implements ***] {
```

declarations

← 属性和枚举常量

constructor definition(s)

← 对象构造和初始化手段

method definitions

← 对象状态查询与控制手段

```
}
```

这三部分之间没有次序规定

对象与类

- 从程序语法上看，似乎面向对象程序与过程式程序差别并不大
 - 都有数据结构
 - 都有过程式函数
 - 都有变量
 - 都有唯一的入口点main
- 差别在于
 - 过程式程序通常按照流程分解来设计开发
 - 面向对象程序按照数据抽象与处理来设计开发

过程式程序与面向对象程序的比较分析

- 假设要设计实现一个多项式加减运算程序
 - 多项式: $c_0+c_1x+c_2x^2+...+c_mx^m$
 - c_2x^2 为一个项, 其中2为该项的幂/阶(degree), c_2 为系数
 - 多项式的阶为其所有项中的最高阶
 - 多项式加减运算对应为阶数相同的项的系数加减
- 我们分别按照过程式程序设计和面向对象程序设计来实现

过程式程序设计实现

- (1)首先定义数据结构来表示多项式

```
struct Poly{  
    int coeff [];  
    int degree [];  
}
```

```
struct Poly{  
    Term pTerm [];  
}  
struct Term{  
    int coeff;  
    int degree;  
}
```

- (2)然后设计实现多项式加法和减法运算函数

```
void PolyAdd(Poly *p1, Poly *p2)  
//p1+p2 → p1  
{  
    ...  
}
```

```
void PolySub(Poly *p1, Poly *p2)  
//p1-p2 → p1  
{  
    ...  
}
```

过程式程序设计实现

- (3) 设计主函数main
 - (3.1) 读取多项式运算式，并构造多项式变量

```
//suppose user inputs poly in (c1,n1),(c2,n2)...
while (...){
    scanf("(%d,%d)", &c, &n);
    Poly p1 = malloc (sizeof(Poly));
    ...
    //为p1中的coeff和degree申请内存，但是不知道输入的多项式的阶是多少？
}
//suppose here we have two polys: p1 and p2
```

- (3.2) 调用加减法函数进行预算
- (3.3) 输出计算结果

对象式程序设计实现

- (1)数据抽象

- 多项式如何表示？如何构造？外部关心它哪些状态？该软件需要对它进行什么处理？

```
public class Poly{  
    private int[] terms;  
    private int deg;  
    public Poly(int deg) {...}  
    public Poly(int c, int n){...}  
  
    public int degree(){return deg;}  
    public int coeff(int d){...}  
  
    public Poly add(Poly q){...}  
    public Poly sub(Poly q){...}  
}
```

对象式程序设计实现

- (2)设计主类和实现入口函数main
 - (2.1)主类管理多项式对象
 - (2.2)主类main必须是public static void main
 - (2.3)在main中构造主类对象来管理相关对象
 - (2.4)主类提供读取多项式操作，通过Poly构造多项式对象，并提供多项式计算操作

```
public class ComputePoly{
    private Poly polyList[];
    private Operator opList[];
    private int num;
    enum Operator{ADD, SUB};
    public ComputePoly() {...}
    private void parsePoly(String s){...}
    private void compute(){Poly p = polyList[0]; Poly p1,p2;
        for(int i=1;i<num; i++){p2 = polyList[i];Operator op=opList[i-1];
            if(op==ADD)p1=p.add(p2); if(op==SUB) p1=p.sub(p2); p=p1;
        }
    }
    private void parseOperator(String s){...}
    public static void main(String args[]){
        //从console获取用户输入的多项式计算表达式: String
        ComputePoly cp = new ComputePoly();
        cp.parsePoly(s); cp.parseOperator(s);cp.compute();
    }
}
```

对比分析

- 过程式程序
 - 每个函数都必须了解数据结构的全部细节。一旦Poly数据结构修改怎么办？
 - main函数的工作量非常大
 - main函数中的变量非常多，难以管理
- 对象式程序
 - 每个类管理着它应该管理的数据，外部无法访问
 - 每个类的操作只处理该类所管理的数据
 - 每个类对外提供状态查询操作
 - Poly对象一旦构造之后不允许改变它

对比分析

- 过程式程序
 - `main`函数控制整个流程，按照计算步骤来初始化相关变量、调用相关函数来处理相关变量等
 - 一旦相关函数和数据发生变化，`main`函数必须进行调整
- 对象式程序
 - `PolyCompute`类管理`polyList`, `opList`, `num`, 不关心`Poly`是什么
 - `PolyCompute`负责与用户交互构造和管理相应的`polyList`, `opList`, 并维护好状态`num` (在`Poly`对象的协助下)
 - `PolyCompute`负责按照`opList`来对`polyList`进行计算（在`Poly`对象协助下）
 - `Poly`对象负责...

对象与类：小结

- 类是从数据视角的抽象结果
 - 不仅仅是数据结构定义
 - 从程序功能角度，把与特定数据相关的操作封装在一起
 - 一个类只管理和这个类职责密切相关的数据
 - 类之间形成层次和协作结构
 - 类的内部对外部不可见，只要确保相关方法规格不发生变化，类的内部细节变化就不会导致使用者跟着变化
- 对象是运行时数据抽象
 - 谁创建对象
 - 谁管理对象

再次讨论课程目标

- 逻辑清晰的程序
 - 结构清晰
 - 命名清晰
- 严密的编程思维
 - 防御
 - 破坏
- 规范的编程思维
 - 规格

作业

一元多项式，每个合法项的幂都是非负整数，且系数不为0。

- 针对多项式例子
 - 使用数组来表示多项式的幂和系数
 - 支持加减运算
- 要求
 - 使用C语言实现一个过程式程序
 - 使用Java实现一个对象式程序
 - **使用控制流程图展示两个程序的控制结构**
 - 自学Java中如何从命令行读取输入和解析输入
 - 两个程序的命令行输入格式相同
 - $\{(c_1, n_1), (c_2, n_2), \dots, ()\} + \{\dots\} - \{\dots\}$: 每个多项式以'{'和'}'来分割；多项式中的每个项以'('和')'分割，多项式之间只能是出现'+'（表示相加）和 '-'（表示相减）
 - 任何空格都将被程序自动识别和过滤掉

具体要求和限制参见按照作业指导书，并严格遵守。

输入划分

表达式

输入格式符合规定

多个多项式

单个多项式

项

多项

系数

指数

系数

指数

负系数

正系数

负系数

正系数

数字要求

前导0

空格过滤

同类项合并

系数为0项

非空项

加法

减法

边界测试

压力测试

大数多项式

超长非法

合法计算结果

数字 $>10^6$

数字大小合适长度超过

输入格式不符合规定

输入格式错误

单个多项式

多项式之间

异常字符

异常数字

字母

非法系数

非法指数

单个项

大括号不匹配

正负号

指数重复

小括号不匹配

逗号缺失

小括号缺失

数字缺失

多逗号

多正负号

正负号缺失