《面向对象设计与构造》课程
Lec1-对象与结构

2022
OO课程组
吴际
北京航空航天大学

内容提要

- 课程简介
- 过程式程序回顾
- 为什么引入对象
- 面向对象程序的构成
- 对象式程序的结构及其设计
- 作业解析

课程介绍

- 三个关键词
  - 设计(design)：整体性分析和规划架构，不可仓促编码
  - 构造(construction)：工程化开发，质量意识和工具链使用
  - 面向对象：一种思维方式，对象及其协作
- 课程目标：掌握面向对象思维方法，并按照工程化方式来设计与构造高质量复杂程序的能力
  - 工程化方式：综合分析软件功能和性能约束，建立问题模型，综合考虑约束进行设计和实现，并能使用测试和分析手段进行综合验证和优化
  - 高质量：能够使用技术手段来说明所开发的软件质量是否满足要求

体系化的课程

“昆仑课程”
（二下春季）

“先导课程”
（一下暑期）

“补给站课程”（二下暑期）

“昆仑课程”核心规则

- **32(授课)+16(实验)+16(研讨)**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、2次实验和2次研讨
  - 3次介绍新内容
    - 每周一个程序任务+互测任务
  - 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
  - 2次围绕作业和课程内容的研讨
    - 组织同学们交流心得体会，邀请企业界大咖介绍相关经验

“昆仑课程”核心规则

- 采用Java语言
- 全程使用云端系统
  - 使用gitlab进行代码管理：gitlab.oo.buaa.edu.cn
  - 使用oo系统进行作业和实验的测试、答疑和管理：oo.buaa.edu.cn
  - 使用博客园来撰写和提交单元总结博客：https://edu.cnblogs.com/campus/buaa/ObjectOriented2022
  - 建议大家认真读一读往年的博客ObjectOriented2021
- 注重指导与反馈的在线实验
  - 针对实验和作业提供铺垫性的技术训练
  - 课上时间在线实验，oo系统内完成
  - 启用线上评测，及时反馈评测结果

关于研讨课

- 研讨课目标
  - 思辨能力
  - 沟通能力
- 2022年改革举措
  - 主题报告：有深度的分享和启发讨论
  - 小组讨论：围绕给定主题的广泛讨论
  - 小组总结：每组推举同学进行总结发言和课后总结讨论(发布到讨论区)

台阶式的公测为你指路

- 中测
  - 基础测试：关注基本功能的正确性
    - 有效性检查的依据：100%
  - 进阶测试：功能覆盖和鲁棒性检查
    - 进入互测的依据：100%
- 强测
  - 深度的组合式测试，关注功能和性能
  - 至少通过一个强测用例，才可能进入互测
- 在开发期间，可以按照一定配额使用中测服务，利用系统反馈来提高代码质量。配额用完后，仍然分配一定的抵扣性配额，每使用一次都会导致扣一些最终的测试分。

兼具平衡性和挑战性的互测

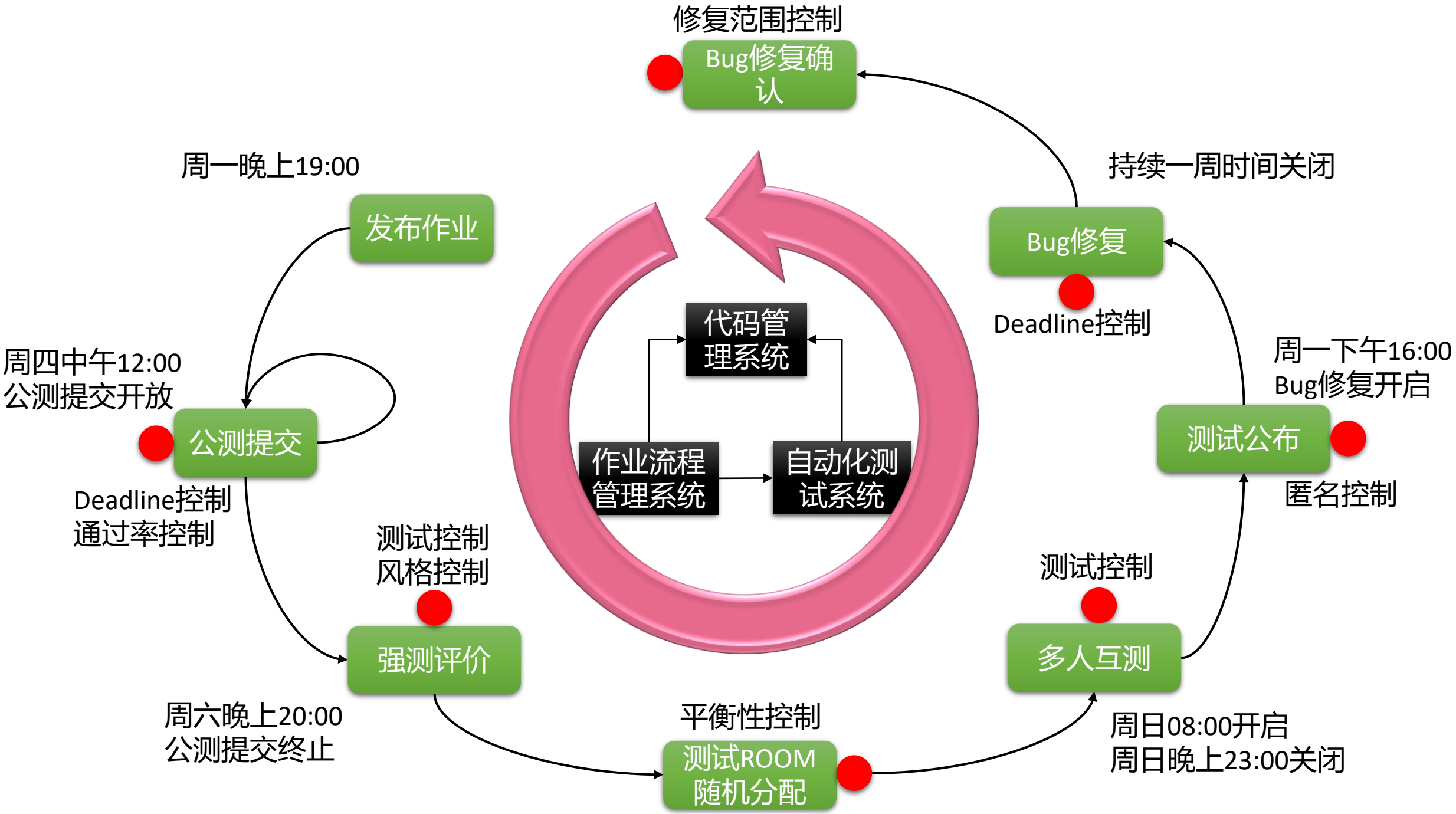
- 互测分为三个等级
  - 每个等级下设多个互测ROOM
  - 按照其公测成绩等确定参与的互测等级，并随机分配到相应的ROOM中
  - 互测期间不知道自己在哪个ROOM中，所有的ROOM都采用统一的编码
- 互测期间可随意查看同处一个ROOM中的他人代码，提交测试数据，如果发现了bug，则算作有效测试，并被测试系统采纳
- 测试系统针对所有有效互测用例，对一个ROOM中的所有程序进行测试
  - 测试得分：Hack了一个ROOM中的多少个程序
  - 被测失分：被Hack了多少个bug
  - 被测失分可以通过后续的bug修复进行补偿

鼓励修复bug

- 测试的目标是发现bug，但质量提升不能止于测试
- 在一次作业完成后的一周内，每个同学都可以积极去修复bug，从而找回测试阶段被扣掉的分数
  - 消除多个测试用例发现相同bug导致的重复扣分影响
- 一旦提交bug修复，系统会做严格检测
  - 使用所有的强测用例和互测用例
  - 不能引入新的bug
  - 成功修复所声明要修复的bug
- 具体得分规则见课程规则文件

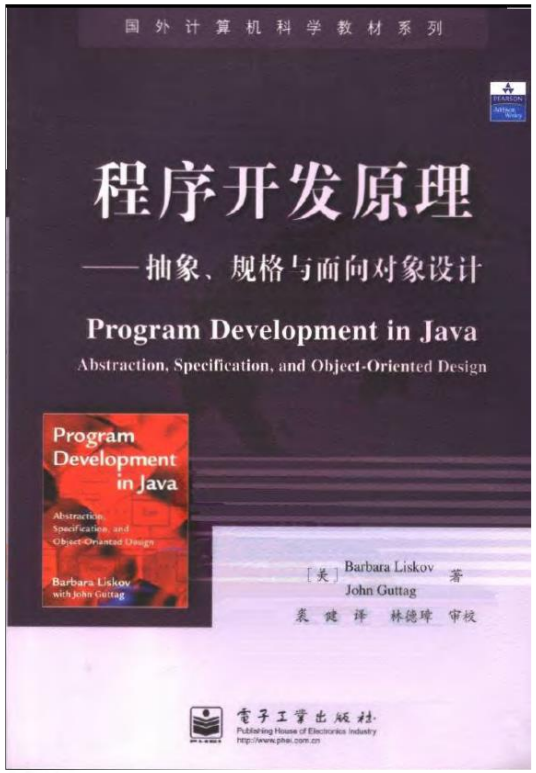
“昆仑课程”核心规则

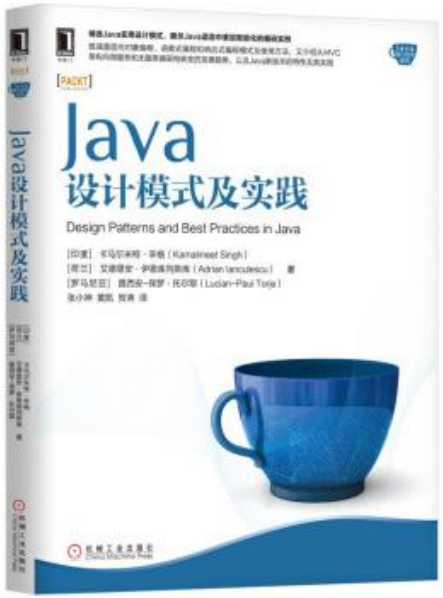
- 成绩评定
  - 作业成绩：55%，作业完成质量
  - 实验成绩：25%，实验完成质量
  - 研讨成绩：10%，参与度、贡献度和质量
  - 博客成绩：10%，总结深度与易读性
- 抄袭是**红线**
  - 基于深度程序结构比对+多名教师人工比对的抄袭检测
  - 发现并确认**一次抄袭**，**取消**作业成绩，所完成的其他有效作业可纳入补给站考察
- 进入**补给站**的条件
  - 综合成绩评定不及格，且无效作业次数**不多于9**次(<=9)

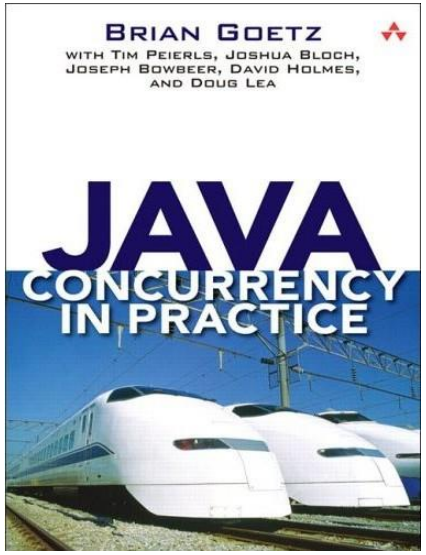

修复范围控制
Bug修复确认
周一晚上19:00
发布作业
持续一周时间关闭
Bug修复
Deadline控制
代码管理系统
作业流程管理系统
自动化测试系统
周四中午12:00
公测提交开放
公测提交
Deadline控制
通过率控制
周一下午16:00
Bug修复开启
测试公布
匿名控制
测试控制
风格控制
强测评价
测试控制
多人互测
周六晚上20:00
公测提交终止
平衡性控制
测试ROOM随机分配
周日08:00开启
周日晚上23:00关闭

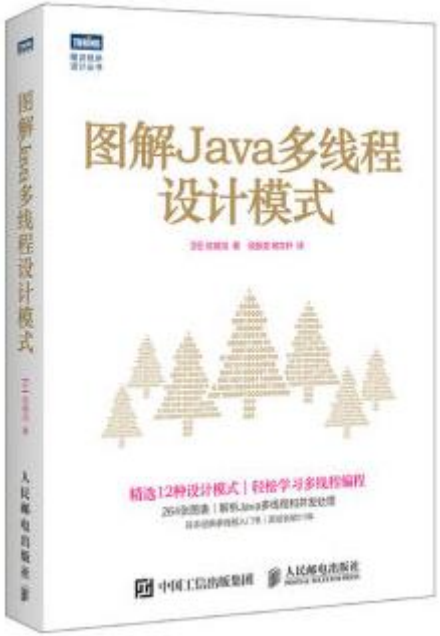
参考材料

- 参考教材
  - **程序开发原理—抽象、规格与面向对象设计**
  - **设计模式**
  - **图解Java多线程设计模式**
  - **Java Concurrency in Practice**
- 互联网
  - **百科**
  - **Jdk规范**
  - **Stackoverflow.com**
  - **技术博客(csdn)**



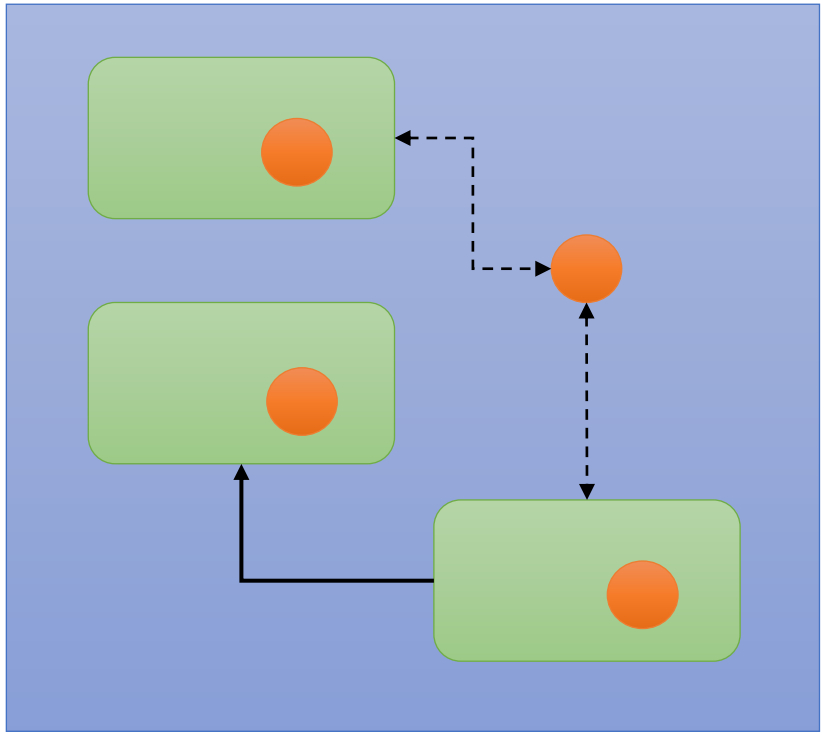






过程式程序回顾

- 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
- 数据结构与行为结构相分离
- 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：多个功能相关函数的集合体（.c文件+.h文件）
- 函数
  - **编程单位**
  - 具有明确的输入、输出，针对输入计算给出结果
  - 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，一般不直接实现软件的具体功能
  - 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
  - 测试难度
    - 高 vs 低

过程式程序回顾

- 函数调用是一种重要的程序组合机制
  - 调用者功能场景 vs 被调用者预期的功能场景
  - 形参与实参的匹配
  - 返回值的处理
- 变量
  - 全局变量：多个函数要使用和处理的变量，如电梯系统的电梯状态
  - 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
- 类型化
  - 基础类型和结构化类型

三种变量之间有什么关系？

为什么引入对象

- 程序编码视角
  - 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
  - 如何让一个函数使用之前运行所产生的中间数据?
    - 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

逻辑联系紧密的数据和函数需要聚合在一起，对象就可以！

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
  - 需要一种手段，通过数据类型来绑定和使用其相应的处理
  - 可以不使用全局变量来进行模块组合

需要一个“结构体”把逻辑聚合的数据和函数形成物理聚合，这就是对象！

为什么引入对象

- 程序思维视角
  - 按照数据处理流程来设计模块
  - 按照数据及其状态变化来进行处理
  - 按照数据的层次来设计针对性的处理

把对象作为一个单位进行设计，自然就为数据和相应的处理同时引入了层次化，且保持一致！

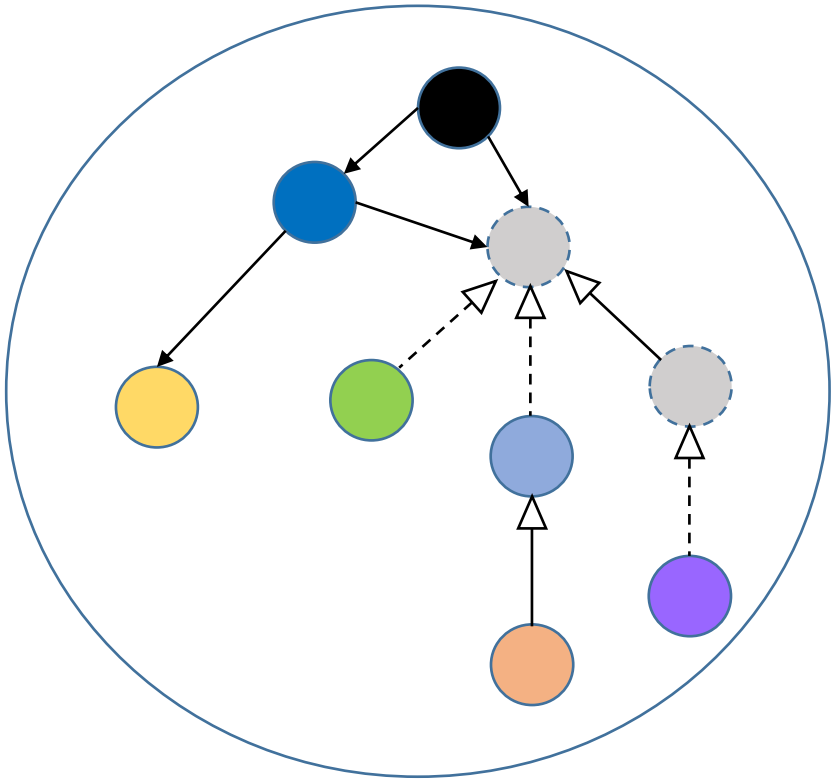
面向对象程序的构成

- 类是对象式程序的**编程单位**
- **类是封装了数据和函数的类型**
  - 类：一般对应某类事物或者逻辑概念
    - 比如人、电梯、时刻、多项式、哈希表等
  - 数据：刻画某类事物或概念的状态
  - 函数：施加于相应数据状态上的操作
- **对象是封装了数据和函数的变量**
- 类之间**协作**完成程序的功能
  - 方法调用
  - 共享数据访问
  - 聚合-分派式数据管理

```
class Time { //类：时间

        int hour, minute;
         //数据：时间由小时和分钟构成

        void addMinutes (int m) {
          //函数：将时间往后推迟m分钟
        }

}
```

面向对象程序的构成

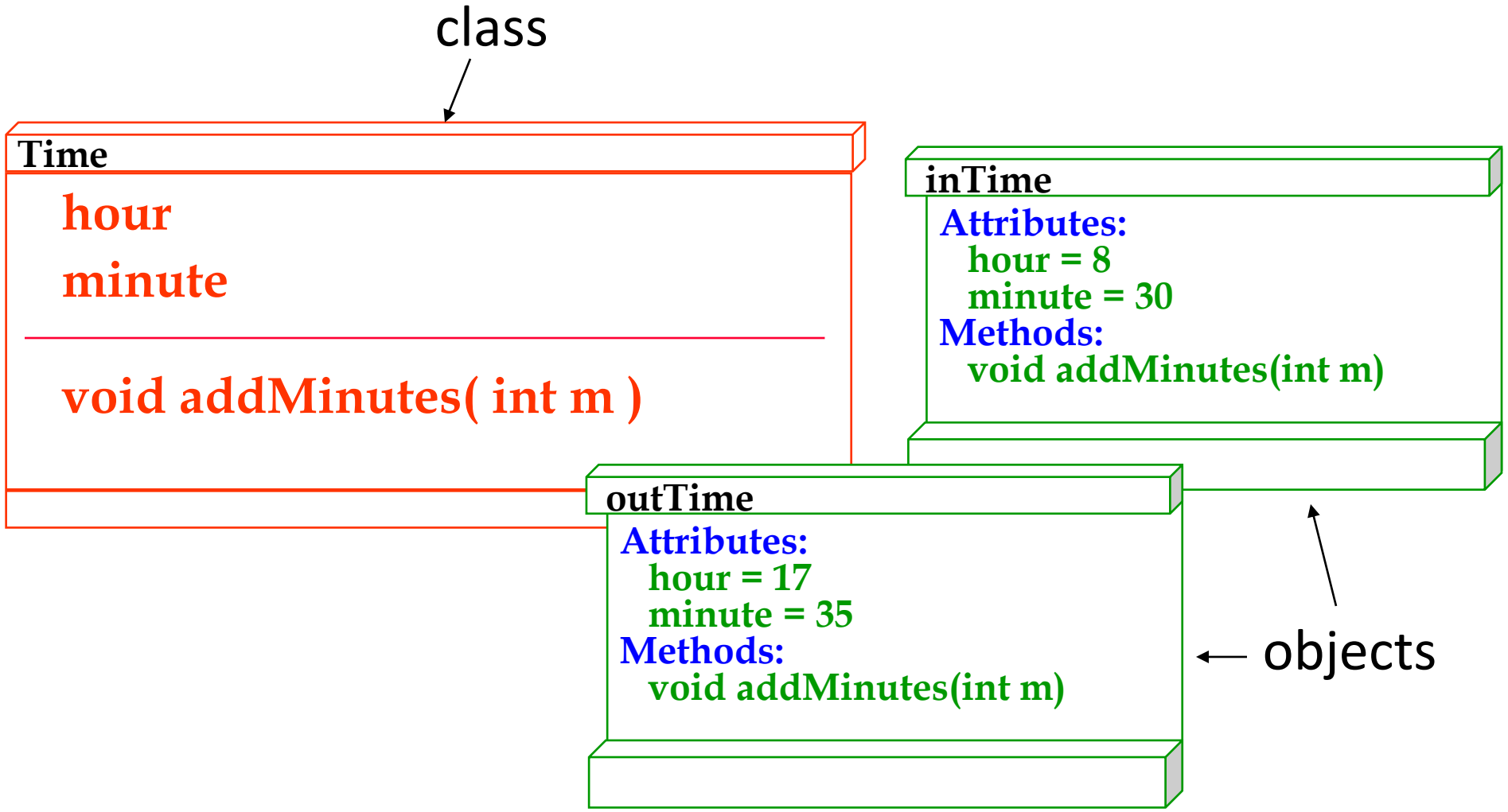
- 一个或多个主类
  - 提供一个标准的**程序入口方法：** public static void main (String[] args)
- 一个或多个一般类
  - 封装业务数据和操作方法
- 一个或多个接口
  - 封装操作
- 类之间的多种关系
  - 继承、关联、聚合
- 类与接口之间的关系
  - 实现
- 接口与接口之间的关系
  - 继承



对象与类

- 对象是类的**实例化**结果
  - 对象是**运行时概念：表现为一个变量及其引用的内存块**
  - 类是**规格概念：表现为一个类型**
- 在**类内部**通过对象来管理具体的数据
  - 实例化完毕的对象： A a = new A(…);
  - 未实例化对象： A 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
hour
minute
void addMinutes(int m)
inTime
Attributes:
hour = 8
minute = 30
Methods:
void addMinutes(int m)
outTime
Attributes:
hour = 17
minute = 35
Methods:
void addMinutes(int m)
objects

对象与类

- 类提供了构造对象的模板
  - 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
- 类通过提供构造器来规范对象的实例化
  - 数据初始化和设置初始状态
- 对于程序中定义的可实例化类A，程序中任意一个类B都可以**构造**A的对象以实现B的方法/功能
  - B可以是A本身

对象与类

一个类可以提供多个构造器，构造具有不同初始状态的对象。

Time newDay = new Time();

Time trainDepart = new Time(9,45);

```
class Time {

        private int hour, minute;

        public Time () {
                hour = 0;
                minute = 0;
        }

        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;
        }
}
```

无参数constructor

有参数constructor

对象与类

- 对象是一个管理着具体数据并可执行计算行为的实体
  - 封装(Encapsulate) 着具体状态，对外部屏蔽细节
    - 对象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
  - 方法执行效果由对象状态和方法计算逻辑共同决定
    - 在不同状态下的执行效果可能会不同
  - 方法调用是一种最常用的交互方式
    - 格式：***object***.*method(p1,p2,…,pm)*
    - 具体调用的是哪个方法具有动态性(后面会解释)

对象内容的外部可见性

- 可见性(visibility)
  - 用于访问控制
  - OO基本准则：隐藏细节，使得细节变更尽可能不影响使用者
  - private: 仅限相同类的对象访问（**可跨同类型对象访问**）
  - public: 对外部完全公开（**可跨任意类型对象访问**）
  - protected: 仅对当前对象和子类对象公开（**可跨相似类型对象访问**）
- 属性与方法的修改影响(change impact)
  - 应尽量保持private，对其修改外部类不可见
  - protected：对其修改后可能需要修改子类实现
  - public：对其修改需要对任何使用相应对象的类进行修改

过程式程序与对象式程序的差异对比

- 从程序表达上看，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
  - 过程式程序以函数来组织
  - 对象式程序以类来组织

  - 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及其处理来设计开发

过程式程序与对象式程序的差异对比

过程式

- 强调过程分解
- 程序由函数组成
- 运行时由函数调用栈+变量表示
- 函数间通过全局变量共享数据
- 逻辑聚合的函数与数据在物理上松散

对象式

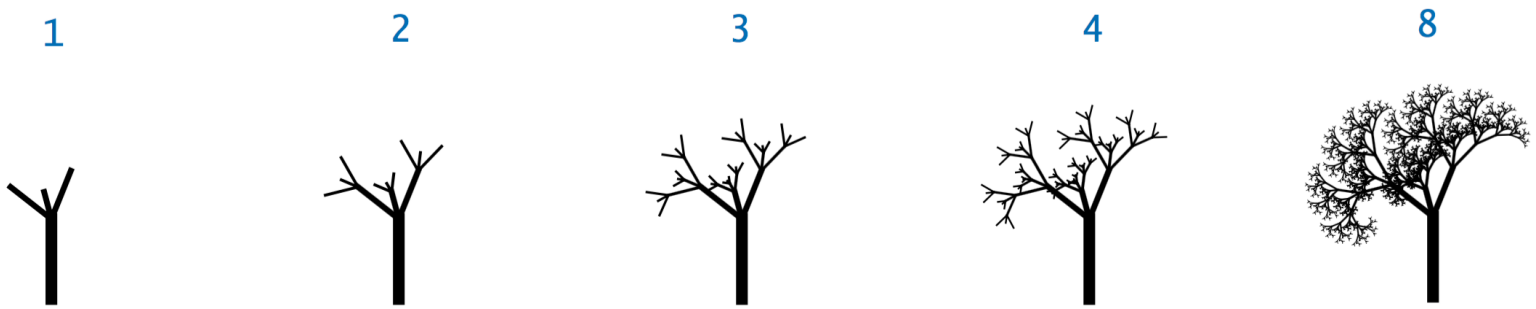
- 强调数据封装
- 程序由类组成
- 运行时由方法调用栈+对象表示
- 共享数据由对象进行保护
- 逻辑聚合的方法与数据物理上也聚合

程序设计的核心是建立结构

- 程序=数据结构+算法 (Niklaus Wirth)
  - the possibility of **defining an infinite set of objects by a finite statement**.
  - **An infinite number of computations can be described by a finite** recursive program
- 数据所蕴含的结构
  - **数据元素与关系**
  - 使用有限数据结构来表示无穷数据实例
- 行为所蕴含的结构
  - **行为动作及其关系**
  - 使用有限控制结构来表示无穷计算能力

程序设计的核心是建立结构

- 序列结构
  - 元素之间呈现线性序列特征
  - 关系：prev，next
  - 数据示例：线性列表
  - 行为示例：顺序计算
- 层次结构
  - 元素之间呈现层次特征
  - 关系：parent, children
  - 数据示例：单位部门组织
  - 行为示例：函数调用
- 嵌套结构
  - 元素之间呈现递归嵌入特征（一个元素作为其自身组成部分重复出现）
  - 关系：kind of, instance of
  - 数据示例：树
  - 行为示例：递归调用



数据维度的结构定义与表示

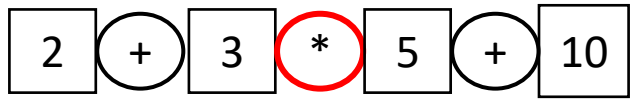
- 数据维度
  - 涉及哪些数据元素
  - 有哪些关系
- 选择合适的结构
  - 不带括号的算术表达式
    - 1+3
  - 单层括号的算术表达式
    - 3*(2+5)
    - (1+2)*(7+8)
  - 带括号嵌套的算术表达式
    - ((1+2)*(7+8)+10)/5

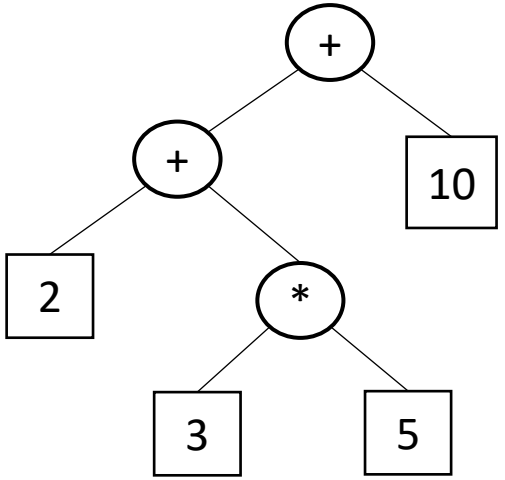
是否可以统一表示?

是否有必要统一表示?

行为维度的结构定义与表示

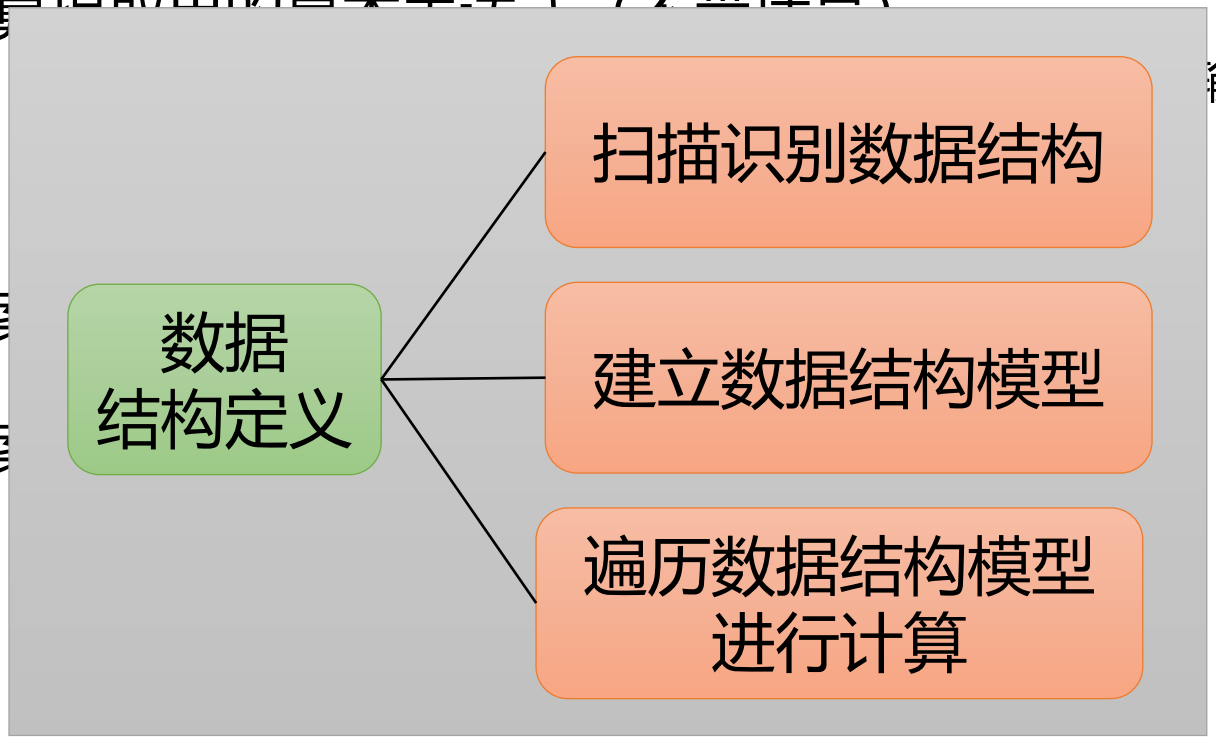
- 行为有明确的输入和输出
- 数据的表示结构与行为结构之间具有紧密关系
  - 计算提取出的算术表达式（不带括号）
    - 依次取出操作数（输入）和操作符，计算得到结果（输出）并放回数据序列
    - **2+3**+5+10→**5**+5+10→ **5+5**+10→**10**+10→ **10+10**→**20**
    - **2+3***5+10→**5***5+10→ **5***5+10→**25**+10→ **25+10**→**35**
  - 方案1：按照优先级多遍扫描计算
    - 2+**3*5**+10→**2**+15+10→**2+15**+10→**17**+10→**17+10**→**27**
  - 方案2：改变数据表示结构，统一规则计算
    - 表达式树
    - 通过结构层次关系表示算符优先级和顺序关系

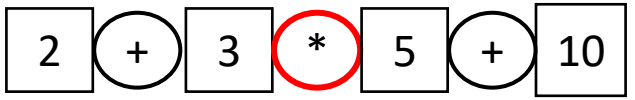


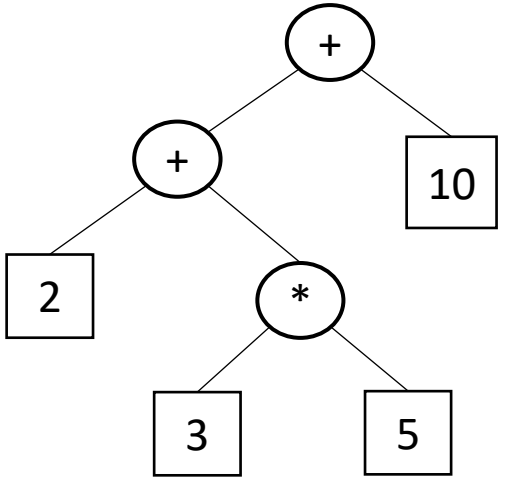


行为维度的结构定义与表示

- 行为有明确的输入和输出
- 数据的表示结构与行为结构之间具有紧密关系
  - 计算提取出的算术表达式（不带括号）
    - 输出）并
    -
    -
  - 方案
    -
  - 方案
    -
    -

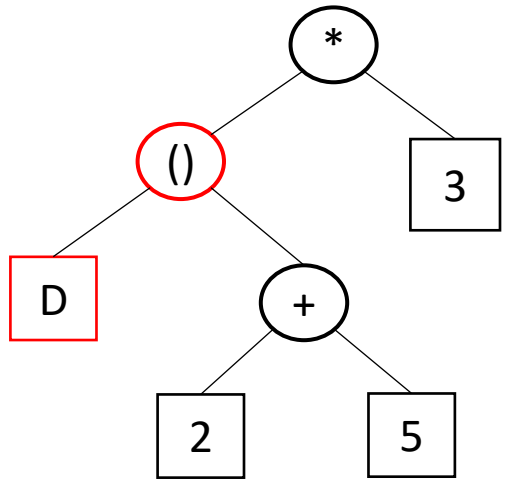






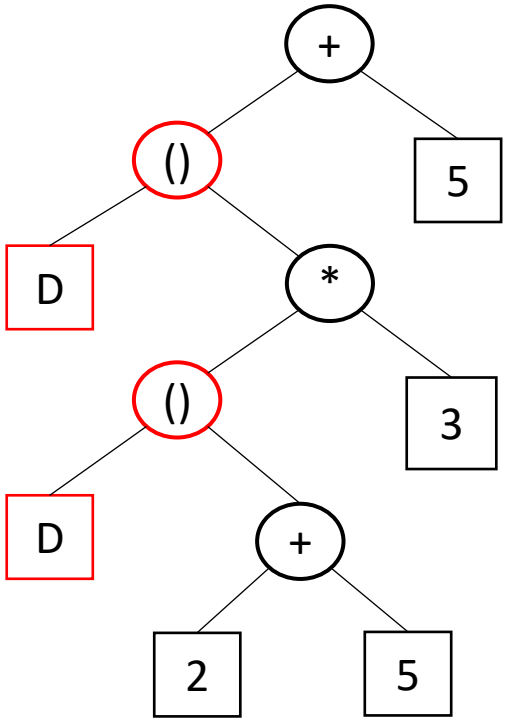
行为维度的结构定义与表示

- 计算提取出的算术表达式（带单层括号）
  - 括号引入了复合结构，也引入了优先级
  - 在表达式中引入特殊算符：’()’
    - 例子：(2+5)*3
    - 为了保证二叉树的结构一致性，引入D(dummy)操作数，不参与计算
- 通过结构表示的统一（二叉树），带单层括号的表达式计算无需修改算法
  - 只需增加‘()’算符和‘D’占位操作数
- 从数据结构的扫描识别角度，要求必须**向前看**才能准确识别括号



行为维度的结构定义与表示

- 计算提取出的算术表达式（带嵌套括号）
  - 单层括号与普通算术表达之间具有层次关系
    - 单层括号中可以出现任意形式的表达式（**但不能有括号**）
  - 嵌套括号进一步松弛了这个约束
    - 单层括号中可以出现**任意形式**的表达式
  - ((2+5)*3)+5
- 二叉树本身自带递归表示能力
  - **统一表示**数据结构和处理结构
- 通过结构表示的统一（二叉树），嵌套括号的表达式计算无需修改算法



面向对象的算术表达式处理

- 使用一个类来表示基本的算术表达式
  - 数据结构
    - 操作符：枚举定义
    - 操作数：引用到算术表达式
  - 行为结构
    - 表达式构造（构造方法）
    - 表达式提取（从给定字符串）
    - 表达式计算（按照算术规则）
- 思考：如果有多种类型的算术表达式该如何处理？
  - 比如常数算术表达式、带变量算术表达式

表达式提取行为的结构分析

- 从输入字符串来识别算术表达式
  - 程序（员）一定拥有算术表达式结构的先验信息：即事先定义了该结构
    - 和前面提到的二叉树有什么区别?
    - 表示结构与逻辑结构
  - 从算术表达式的表示结构→逻辑结构
    - 本质上是parsing!
- 字符串是线性序列结构（字符序列）
  - 我们关心四类特殊符号
    - 运算符：+，-，*，/
    - 操作数：整数表示，是否带正负符号?
    - 左括号：复合结构的开始标记
    - 右括号：复合结构的结束标记（对应哪个复合结构？）

表达式提取行为的结构分析

- 通过结构分析，我们可以识别模式（pattern）
  - 可重复出现的结构单位
- 算术运算pattern
  - a+b, a-b, a*b, a/b
  - a和b均为占位符，可替换为任意合适的表达式
- 复合结构pattern
  - (a)，a为占位符，可替换为任意合适的表达式
- 数字pattern
  - [+/-]dd…d，d为0..9数字，第一个d不能为0
  - +可忽略，-需保留
- 提取（数据以及行为）结构就是扫描字符串来识别相应的pattern

对象与类

- 类是从数据视角的设计结果
  - 不仅仅是数据结构定义
  - 从程序功能角度，把与数据相关的业务所需操作封装在一起
  - 一个类只管理和这个类职责密切相关的数据
  - 类之间可以形成层次和协作结构
  - 类的内部对外部不可见，只要确保**方法的外部可见行为**不发生变化，类的内部细节变化就无需告知使用者
- 对象管理着程序运行时的业务数据
  - 谁创建对象
  - 谁管理对象

对象式程序的模块结构

- 一个程序真正要处理的核心数据有哪些?
  - 按照数据特征构造相应的类，来管理数据和提供相应的数据处理行为
  - 算术表达式应该成为一个类
    - 要管理哪些数据？应采用什么结构来管理数据?
- 数据从哪里来，到哪里去?
  - 程序需要一个或多个类来专门处理数据输入和输出
- 程序必须要有一个控制框架
  - 提供入口方法的主类
  - 管理程序执行过程中实际构造的顶层对象
- 建立这三种不同角色的类之间关系

对象式程序的数据管理结构

- 原子数据
  - 通常使用原子类型来表示的数据
- 复合数据
  - 建立数据之间的聚合关系，有可能是多个聚合层次
  - 手段：通过类之间的关联关系来构造
- 单例数据
  - 只使用单体变量就可以表示相应的数据
- 多例数据
  - 需要使用数组等容器来表示相应的数据
  - Q1：规模是否静态可知?
  - Q2：数据是否在运行中动态获得?

对象式程序的数据管理结构

- 推荐使用Java类库提供的数据容器
  - 提供了统一的接口
  - 提供了多种容器类型
- ArrayList与LinkedList
  - 数据存储：基于数组 vs 基于双向链表
  - 数据访问：按照下标来访问数据get(index)
  - 容器规模可自动增长
  - ArrayList适用于数据量较多，且无需频繁改变数据次序关系
  - 性能差异：ArrayList的add和get无差别；LinkedList的get比add要慢很多

对象式程序的数据管理结构

- HashMap
  - Map类型，存储key-value对
  - Key用来获取Map中所存储元素的对应Value
  - 可以分别遍历Entry集合（key-value）、Key集合和Value集合
  - 不支持下标遍历，只能通过Iterator
- HashSet
  - Set类型，基于HashMap实现
  - 不支持下标遍历，只能通过Iterator

```
Iterator iter = map.entrySet().iterator();
while(iter.hasNext()) {
    Map.Entry entry = (Map.Entry)iter.next();

    key = (String)entry.getKey();
    integ = (Integer)entry.getValue();
```

```
// 假设set是HashSet对象
for(Iterator iterator = set.iterator();
        iterator.hasNext(); ) {
    iterator.next();
}
```

作业分析

- 数学表达式的括号展开与合并
  - 表达式由项组成，直接通过’+’或’-’连接
  - 项由因子组成，通过’*’连接
  - 因子有三种：常量因子、复合因子、变量因子
- 括号展开的本质是什么？
  - 按照一定的**规则**对多项式进行**变换**
  - 项中的**因子类型**不同，**适用规则**不同
- 合并的本质是什么？
  - 同类项合并、常数项合并
- 一个完整的OO程序
  - 主类、输入输出类、对象管理类
  - 各自职责是什么？

该描述是否蕴含了某个层次结构?

变换规则依赖于结构，也会改变结构

作业准备工作建议--String

- 字符串(String)是最常用到的类之一
  - 这是个不可变类，下节课会具体介绍不可变概念，直观含义是这个类不提供改变一个对象的方法（只能构造生成新的对象）
  - s1 = s1 + “abc”？
  - **不希望**见到这种代码：char str[];
- 两种初始化方法
  - String s = new String (“abc”)
  - String s = “abc”

作业准备工作建议--String

- 查找方法
  - charAt方法：按照索引值(从0开始递增)，获得指定位置的字符(char)
  - indexOf方法：查找特定字符或字符串在当前字符串中的第一次出现位置
- 截取与改变
  - trim方法：把字符串头尾的空格去掉
  - substring方法：截取字符串中的“子串”
  - concat方法：字符串拼接，将两个字符串连接以后形成一个新的字符串
  - replace方法：替换字符串中所有指定的字符
  - 所有的改变都以一个新的字符串对象来呈现
    - str.trim()不会改变str，如果忘记保留返回的字符串对象，等于无用功

作业准备工作建议--String

- 检验方法：检查字符串是否满足某个条件
  - equals方法：判断两个字符串对象的**内容**是否相同
  - compareTo方法：比较两个字符串的大小（按照字典序依次比较每个字符）
  - startsWith方法：判断字符串是否以某个字符串开始
  - endsWith方法：判断字符串是否以某个字符串结尾
- 分割方法：按照一定的模式把字符串分割成几个子串
  - String[ ] split(String regex)：按照给定的**正则表达式**来分隔，返回分隔后形成的多个字符串对象数组

作业准备工作建议--Regex

- 正则表达式
  - 定义字符串序列结构的规则模式，是一种**特殊的字符串**
- 正则表达式的主要构成元素
  - 原子符号
  - 元字符

元字符
* 匹配前一个内容的0次1次或多次
. 匹配内容的0次1次或多次，但不包含回车换行
+ 匹配前一个内容的1次或多次
？匹配前一个内容的0次或1次
| 选择匹配
^ 匹配字符串首部内容
$ 匹配字符串尾部内容
\b 匹配单词边界，边界可以是空格或者特殊符合
\B 匹配除带单词边界以外内容
{m} 匹配前一个内容的重复次数为m次
{m,} 匹配前一个内容的重复次数大于等于m次
{m,n} 匹配前一个内容的重复次数m次到n次

原子:
①a-z A-Z _ 0-9 //最常见的字符
②(abc) (skd) //用圆括号包含起来的单元符号
③[abcs] [^abd] //用方括号包含的原子表，原子表中的**^代表排除或相反内容**
④转义字符

\d： 任意一个数字[0-9]

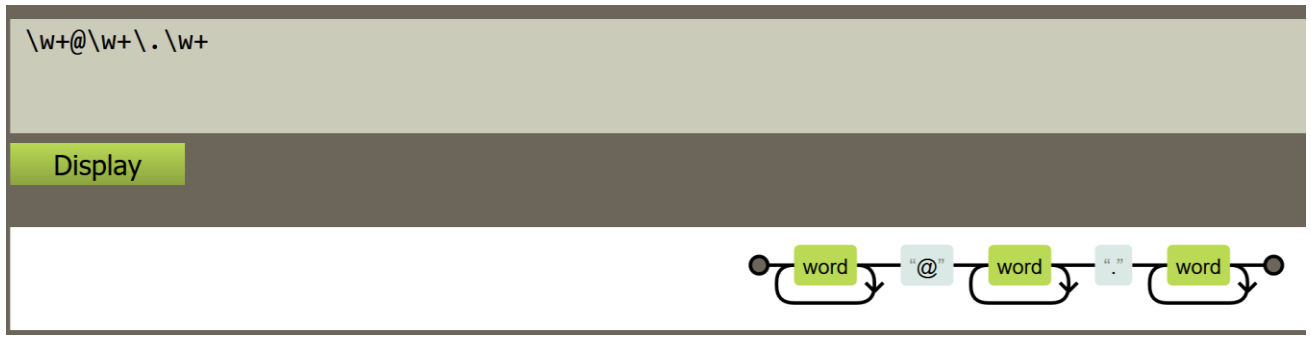
\D： 除所有数字外的字符[^0-9]

\w： 任意一个英文字符[a-zA-Z_0-9]

\s： 空白符号如回车、换行、分页等 [\f\n\r]

作业准备工作建议--Regex

- 常见的正则表达式及其含义解读
  - 新主楼门牌号：楼号+层号+房间号 `[ABCDEFGH]\d{1,2}\d{2}`
  - 固定电话号码：区号+话机号 `\d{3,4}-\d{6,8}`
  - 出生日期：年+月+日 `\d{4}\d{2}\d{2}`
  - Email地址：账户号+@+邮箱服务器地址 `\w+@\w+\.\w+`
    - 邮箱服务器地址：运行商名称+’.’+公司类型后缀
- 如何知道自己构造的正则表达式是否正确？
  - https://regexper.com
  - 预期字符串模式的可视化展示



作业准备工作建议--Regex

- 通过java.util.regex包来使用正则表达式的强大处理能力
  - Pattern 类：构造字符串模式分析对象
  - Matcher 类：使用模式分析对象对输入字符串进行匹配
- 这段代码用于检测一个字符串是否为合法的手机号
  - 我们假设合法的手机号是1开头，一共11位数字

```
String input; //input是要检测的输入字符串
String check = "1\\d{10}";
Pattern regex = Pattern.compile(check);
Matcher matcher = regex.matcher(input);
boolean isMatched = matcher.matches();
```

check=“1[34578]\\d{9}”;

作业提醒建议

- 注意阅读和了解课程规则
- 注意仔细阅读作业指导书，然后**整体性**思考作业要求
- 参与Pre训练的重要性
  - 熟悉Java语言和编程环境，特别是如何**调试**Java程序
  - 掌握课程工具链的使用，特别是**gitlab**
- 注意作业的**时间节点**要求，尽早开展工作，并使用中测服务来提高程序的质量
- 完成作业的关键步骤
  - 问题的数据和行为**结构分析**
  - **架构性**的设计：关键性的类和关系
  - 编码+测试，**持续迭代**