第四讲—单元训练总结

OO课程组2018

北京航空航天大学计算机学院

内容提纲

- 作业训练要点分析
- 作业中共性问题的分析
- 作业程序的推荐设计
- 本次作业

作业训练要点分析

- 作业1训练要点分析
- 作业2训练要点分析
- 作业3训练要点分析

作业1训练要点分析

- 多项式加减运算
  - 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命令和解析命令
  - 两个程序的命令行输入格式相同
    - {(c1,n1),(c2,n2),…,()}+{…}-{…}: 每个多项式以'{'和'}'来分割；多项式中的每个项以'('和')'分割，多项式之间可以出现'+'（表示相加）和'-'（表示相减）

Tips for testing

是否能够处理空多项式？
阶数超大多项式能否处理？
负数阶能否识别？
系数为负？
输入格式错误能否识别？

作业1训练要点分析

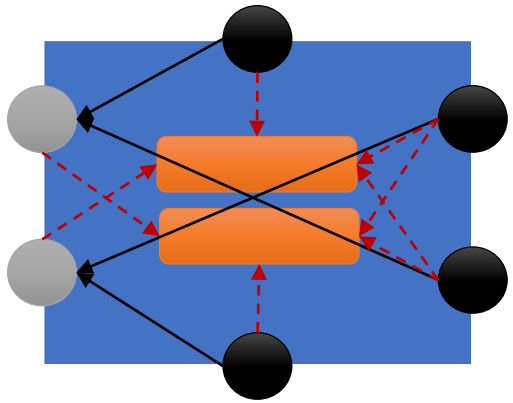
- 初步认识面向对象程序的结构特点
- 区分过程式程序与面向对象程序在结构上的主要差异
- 初步认识面向对象程序的运行行为和调试
- 初步认识输入处理与程序鲁棒性的关系
- 初步掌握结构化输入的处理技巧

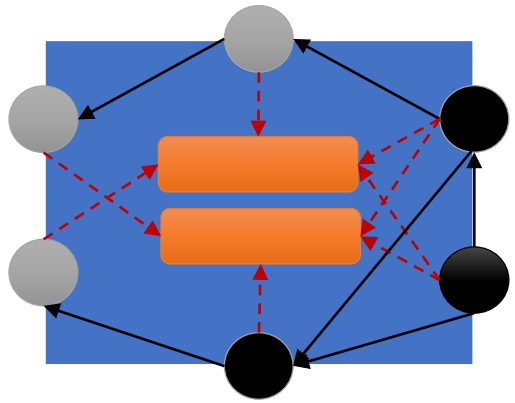
作业1训练要点分析

- 初步认识面向对象程序的结构特点
  - 类作为编程单位
  - 提供public static void main(…)方法的称为主类，程序的执行入口点
    - 谁来调用这个main方法？
  - 类具有相对的独立性：任意两个类之间不应该维护***相同的可变数据***
    - 为什么？
  - 类在方法上自包含，即一个类能够完成处理自己所管理的数据
    - 为什么？
  - 每个类方法都有明确的职责

作业1训练要点分析

- 区分过程式程序与面向对象程序在结构上的差异
  - 函数是编程单位
  - 数据结构和函数相对独立
  - 数据的维护职责不清晰
- “真”面向对象程序与“伪”面向对象程序的差异



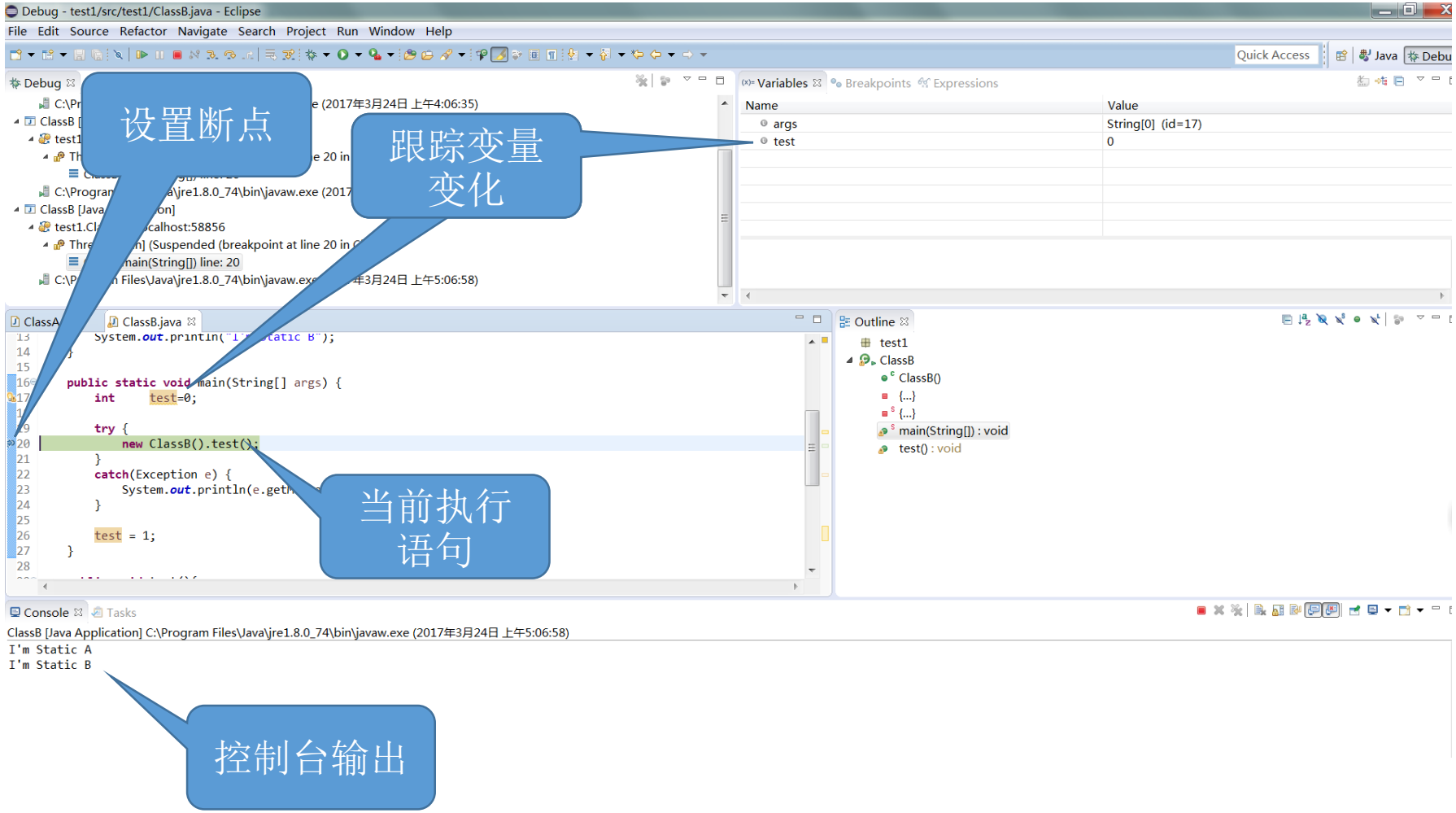


作业1训练要点分析

- 自我诊断手段
  - 是否能用一句话概括类方法的功能
  - 分析不同方法之间的代码逻辑相似性
  - 检查public方法被外部调用的比例
  - 检查类方法的调用深度

作业1训练要点分析

- 用好IDE工具
  - 巧用快捷键
  - 查看变量变化
  - 观察系统输出
  - 设置调试断点
    - Line
    - Method
    - Watchpoint
    - Class
    - Exception
  - 跟踪代码执行
    - Step Over
    - Step Into
    - Step Return
    - Run to Line



调试行为具有一定模式特征

通过调试行为片段分析

以sd-(ns|nf)*-ed为主

序号	片段
1	ab-db
2	ab-nf
3	ab-ns
4	ab-sd
5	ab-db

序号	片段
1	ns-nf
2	ns-ed
3	ns-db
4	ns-ab

序号	片段
1	nf-ed
2	nf-sd

序号	片段
1	db-ab
2	db-ed
3	db-ns
4	db-nf
5	db-sd

ab	Add breakpoint
db	Delete breakpoint

ns	Next step
nf	Next function

sd	Start debug
ed	End debug

以士谔书院2017AwayFromBug程序调试大赛参赛选手的在决赛中产生的真实数据

通过对数据进行识别和清洗,得到来自16位同学的71个调试片段,包含6个种类2048个具体动作

调试行为模式的含义

一级行为模式	二级行为模式	行为
假设行为模式	打断模式	调试人员在某处增加断点，假设缺陷存在于该位置
	断点变更模式	调试人员在调试过程中增减断点的模式，是其对缺陷位置的进一步假设行为
演绎行为模式	推理模式	调试人员为了对假设的问题进行验证的推理行为，首先在推理起点增加断点，随后在此断点之后进行单步调试操作
	思考模式	调试人员的思考行为。调试者可能在调试的过程中由已经出现的调试结果花费不定长的时间进行思考
	程序上下文与断点关联模式	调试人员在一个位置暂停执行后，在另外一个位置设置了新的断点
修正行为模式	修正行为模式	调试人员终止一次调试后，在下一次调试活动中依然会悬停在上一次终止调试的位置

作业1训练要点分析

- 初步认识面向对象程序的运行行为和调试
  - 对象是运行时的行为单位
    - 线程是运行时的并发控制单位
  - 运行栈中看到的信息：方法+对象==》对象的状态
  - 不存在纯粹的方法调用，而是请求一个对象的服务
    - static方法 vs non-static方法
  - 方法断点在调试时会出现多种上下文场景
    - 在执行不同对象的方法
    - 查看this变量

作业1训练要点分析

- 初步认识输入处理与程序鲁棒性的关系
  - 用户输入处理是软件的重要组成部分
    - 鲁棒性、性能、用户体验
  - 用户输入的主要形态
    - 流输入：语音流、视频流等
    - 被动的非结构化输入：图片
    - 被动的结构化输入：结构化文本
    - 被动的控制输入：中断
    - 主动的非结构化输入：利用设备捕捉图片
    - 主动的结构化输入：利用GUI等获取用户输入
    - 主动的控制输入：端口监听

作业1训练要点分析

- 初步认识输入处理与程序鲁棒性的关系
  - 鲁棒性的内涵
    - 程序在任何情况下都能保持自身不崩溃的特性
    - 程序能够主动识别输入异常的特性
    - 程序能够正确处理有效输入的特性
  - 如何识别输入中的无效内容？
  - 识别到输入异常/程序运行中的状态异常该怎么办？

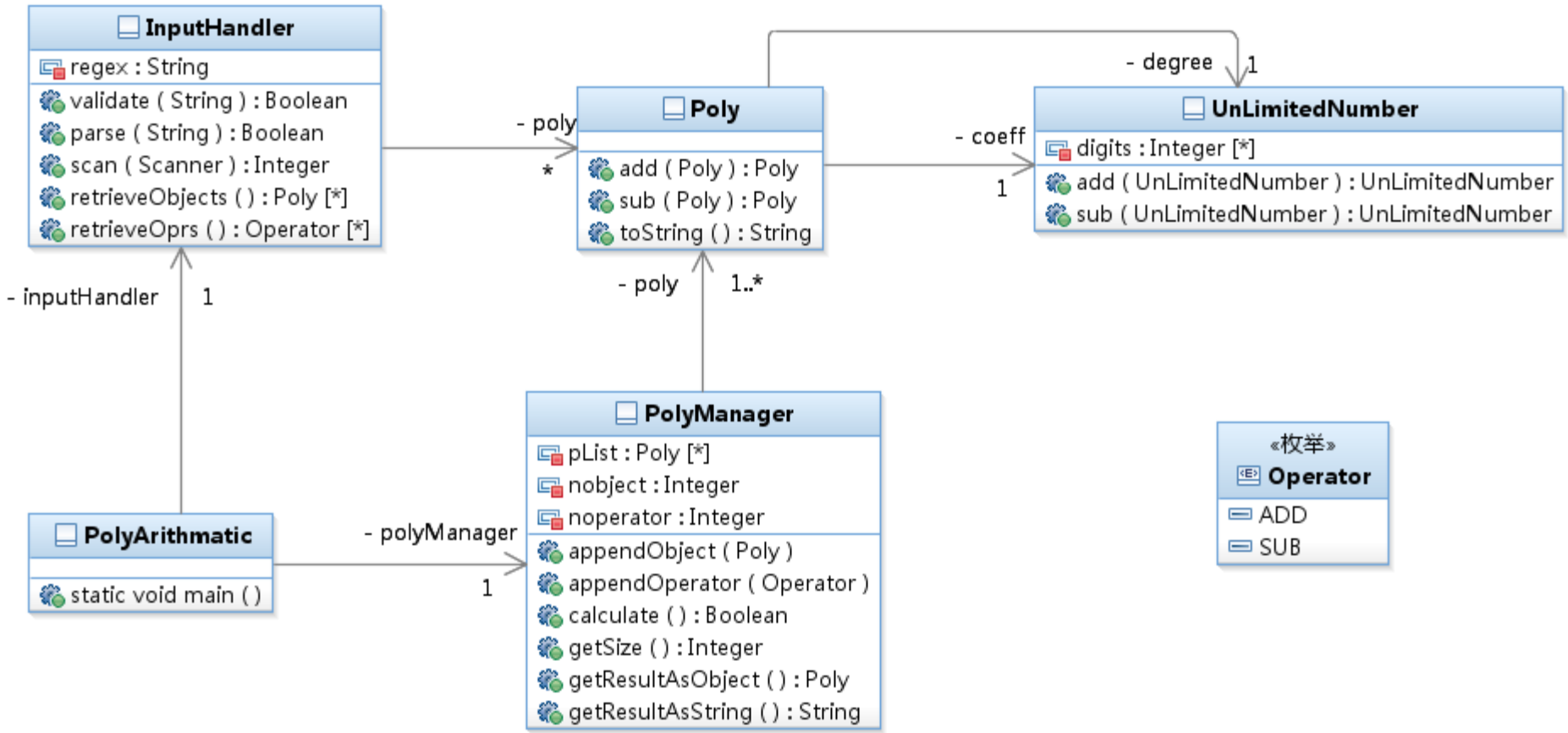
作业1训练要点分析

- 初步掌握结构化输入的处理技巧
  - “结构化”指什么？
    - 任何输入都具有结构
      - 图片：矩阵
      - 自然语言文本：字符串
      - 音频：字节流
    - 能够从结构中清晰提取出一定模式的输入
  - 结构模式
    - 不论输入长度如何变化，普遍适用于输入的每个片段
    - 结构模式不依赖于输入的内容变化
  - 使用正则表达式来显示表达相应的结构模式
    - 正则表达式也有相应的处理极限：注意它会抛出的异常

作业1易出问题分析

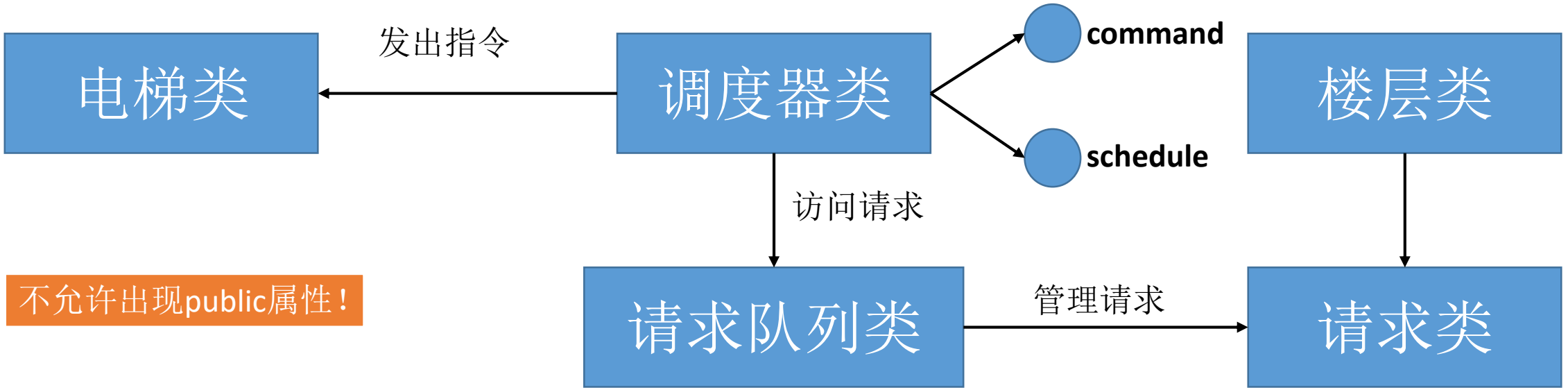
- 单个方法规模过大
  - “one-for-all”的main方法，几乎实现全部功能
- 多项式类同时处理输入内容的有效性和多项式运算---输入处理与业务处理未分离
  - 缺点是什么？
- “数豆子”式、枚举式输入有效性检查处理
  - 逐个字符进行处理，大量的if、for、while嵌套
- 有些类从输入处理到输出，有些类只是简单保存一个数据
  - 分类的返回值，但是调用时未做分类处理
- 隐含的鲁棒性风险
- 程序风格问题
  - 变量、类、方法命名
  - 代码缩进
  - public属性

作业1建议设计


InputHandler
regex : String
validate (String) : Boolean
parse (String) : Boolean
scan (Scanner) : Integer
retrieveObjects ( ) : Poly [*]
retrieveOprs ( ) : Operator [*]
- poly
*
Poly
add (Poly) : Poly
sub (Poly) : Poly
toString ( ) : String
- degree
1
UnLimitedNumber
digits : Integer [*]
add (UnLimitedNumber) : UnLimitedNumber
sub (UnLimitedNumber) : UnLimitedNumber
- coeff
1
- inputHandler
1
- poly
1..*
PolyArithmatic
static void main ( )
- polyManager
1
PolyManager
pList : Poly [*]
nobject : Integer
noperator : Integer
appendObject (Poly)
appendOperator (Operator)
calculate ( ) : Boolean
getSize ( ) : Integer
getResultAsObject ( ) : Poly
getResultAsString ( ) : String
«枚举»
Operator
ADD
SUB

作业2设计建议

Testing tips:
(1)从请求队列状态和请求时间间隔角度来设计测试用例；
(2)从输入有效性设计测试用例。



不允许出现public属性！

电梯类与调度器类都需要了解电梯的运动状态和运动方向，采取什么措施来解决相同数据问题？

建议：
(1)从输入读取请求
(2)构造请求对象
(3)加入到队列中
(4)启动调度
(5)记录电梯对象对请求的响应

Scheduler类有一个command方法，其结果是调用schedule方法获得当次调度请求，并交给电梯类响应。
Scheduler类的schedule方法根据请求队列情况和电梯运行状态选择可调度的请求（本次作业采用傻瓜策略）。

作业2训练要点分析

- 如何把程序功能“均衡”分配给多个类
- 如何让多个类之间进行协同
- 初步熟悉基于队列的调度
- 初步了解另一种测试形态

作业2训练要点分析

- 如何把程序功能“均衡”分配给多个类
  - 程序需求涉及多个实体或控制
  - 程序功能开始增多，需要多个类来实现
  - 避免出现两种类
    - God class that knows everything
    - Idiot class that knows nothing
  - 均衡的含义
    - 类之间的均衡
      - 每个类做其“份内”之事→从需求和问题域进行分析
      - 所有类承担的职责相当→从设计结果进行分析
    - 类方法间的均衡
      - 方法职责相当

作业2训练要点分析

- 如何发现不均衡的设计？
  - 统计每个类的属性数目
  - 统计每个类的方法数目
  - 统计每个类的代码行数
  - 统计每个方法的代码行数
  - 做一个基本的均值和方差分析
- 为什么强调均衡设计？
  - 保持结构的简洁
  - 确保局部逻辑不至于太复杂
  - 均衡的结构具有稳定性！
- 如何做到均衡设计？
  - 每个类的职责要明确，尽量单一化
  - 把功能流程切开分配到相关类，避免面条式程序
    - 确保只存储和管理必要的数据
    - 确保每个类只处理自己管理的数据

作业2训练要点分析

- 如何让多个类之间进行协同
  - 类要保持逻辑上的独立性
  - 通过协同进行交互（协同模式）
    - Sit-together to work: 一个类需要联合其他类的多个对象来完成某个工作
    - Exchange data: 两个对象之间需要交换数据以完成相应的计算和处理
    - Request for help: 一个对象请求另一个对象的服务
  - 电梯、调度器、请求队列、请求之间有哪些具体协同？
    - 电梯----调度器
    - 调度器----请求队列
    - 请求队列----请求
    - 电梯----请求

作业2训练要点分析

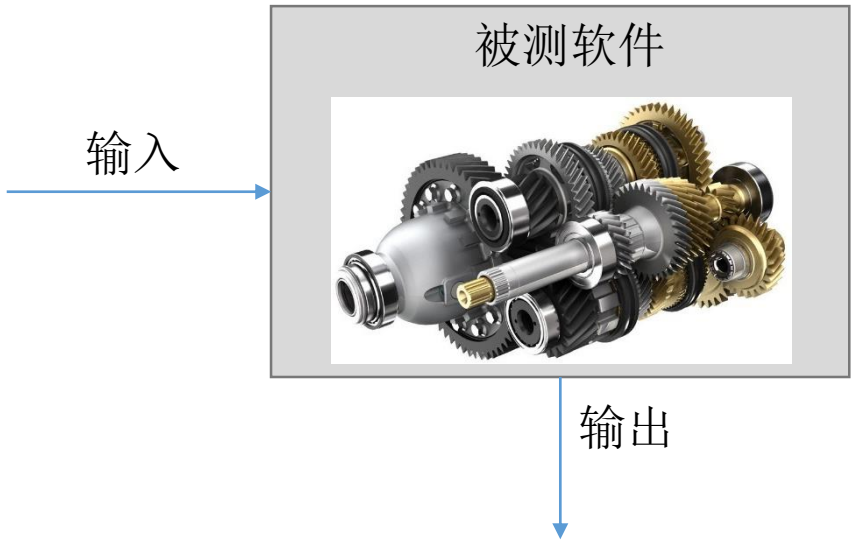
- 初步熟悉基于队列的调度
  - 调度是计算机科学中的一个核心概念：操作系统层次的任务调度、线程调度；运筹管理中的任务调度；实时系统中的请求调度等
  - 调度在本质上是把紧缺的处理资源分配给任务，从而最大化任务综合性能的过程
    - 排他性处理资源：CPU/CPU核
    - 共享性处理资源：电梯
    - 要考虑任务的性能要求
      - 计算机任务：优先级、执行时间
      - 电梯请求任务：发出时间(优先级)、距离
    - 两类调度
      - All-or-nothing：电梯请求
      - Replaceable/preemptive: 操作系统任务

作业2训练要点分析

- 初步熟悉基于队列的调度
  - 使用队列是一种经典的方法
    - 队列的序蕴含着任务或请求的优先级 ==》极大影响调度性能
    - All-or-nothing使用什么队列？
    - Replaceable/preemptive使用什么队列？
  - 队列的操作
    - 入队、出队
    - 查询
    - 队列的操作性能是关键
  - 如果任务量很大，该设计什么样的队列？
    - 银行高峰期如何排队？

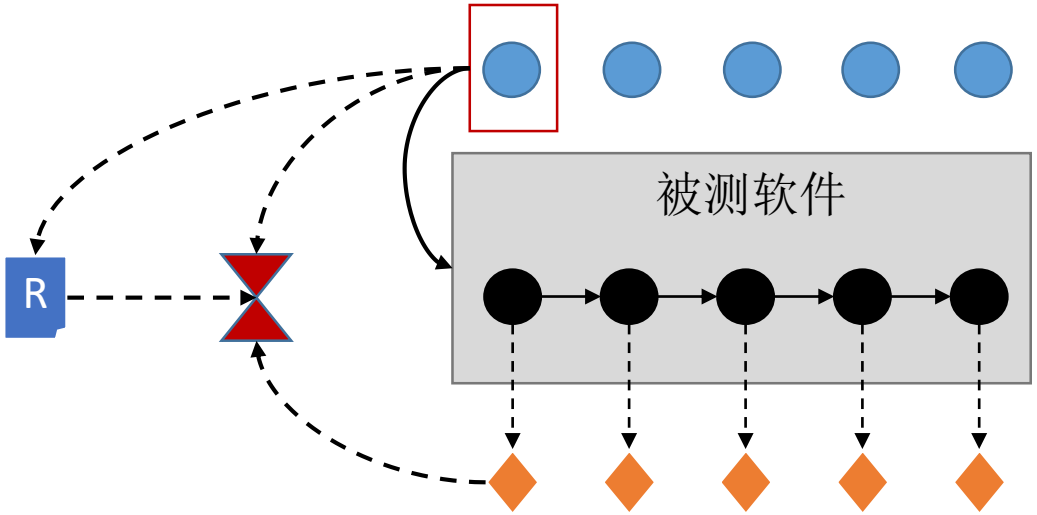
作业2训练要点分析

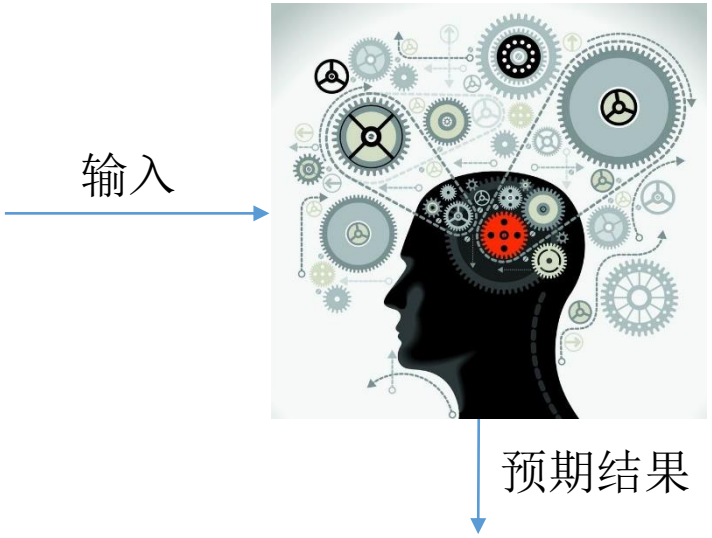
- 初步了解另一种测试形态
  - 对比第一次作业的测试和第二/三次测试，有什么差异？
  - 你是怎么做测试的？
    - 如何设计输入？
    - 如何判断执行结果是否正确？

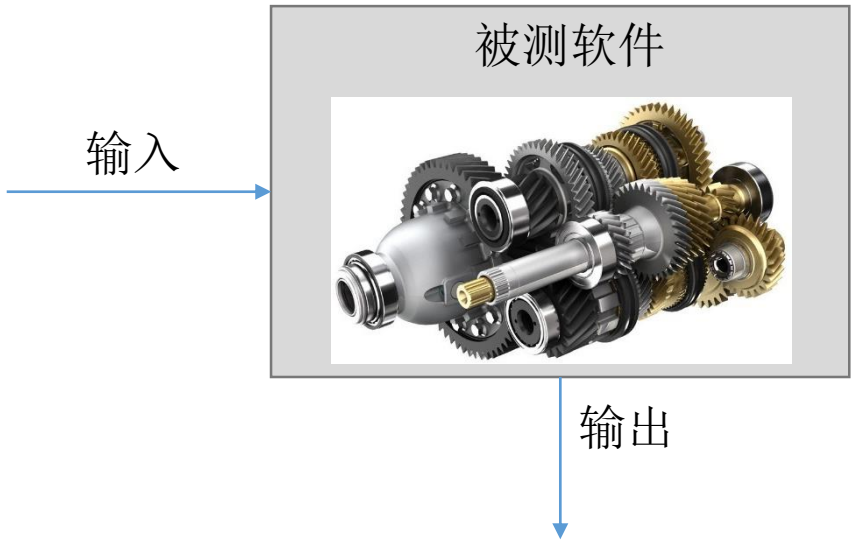


作业2训练要点分析

- 初步了解另一种测试形态
  - 预期结果的推理是难点
    - 第一次作业：线性推理
    - 第二次作业：单调调度推理
    - 第三次作业：优先级调度推理







作业2易出问题分析

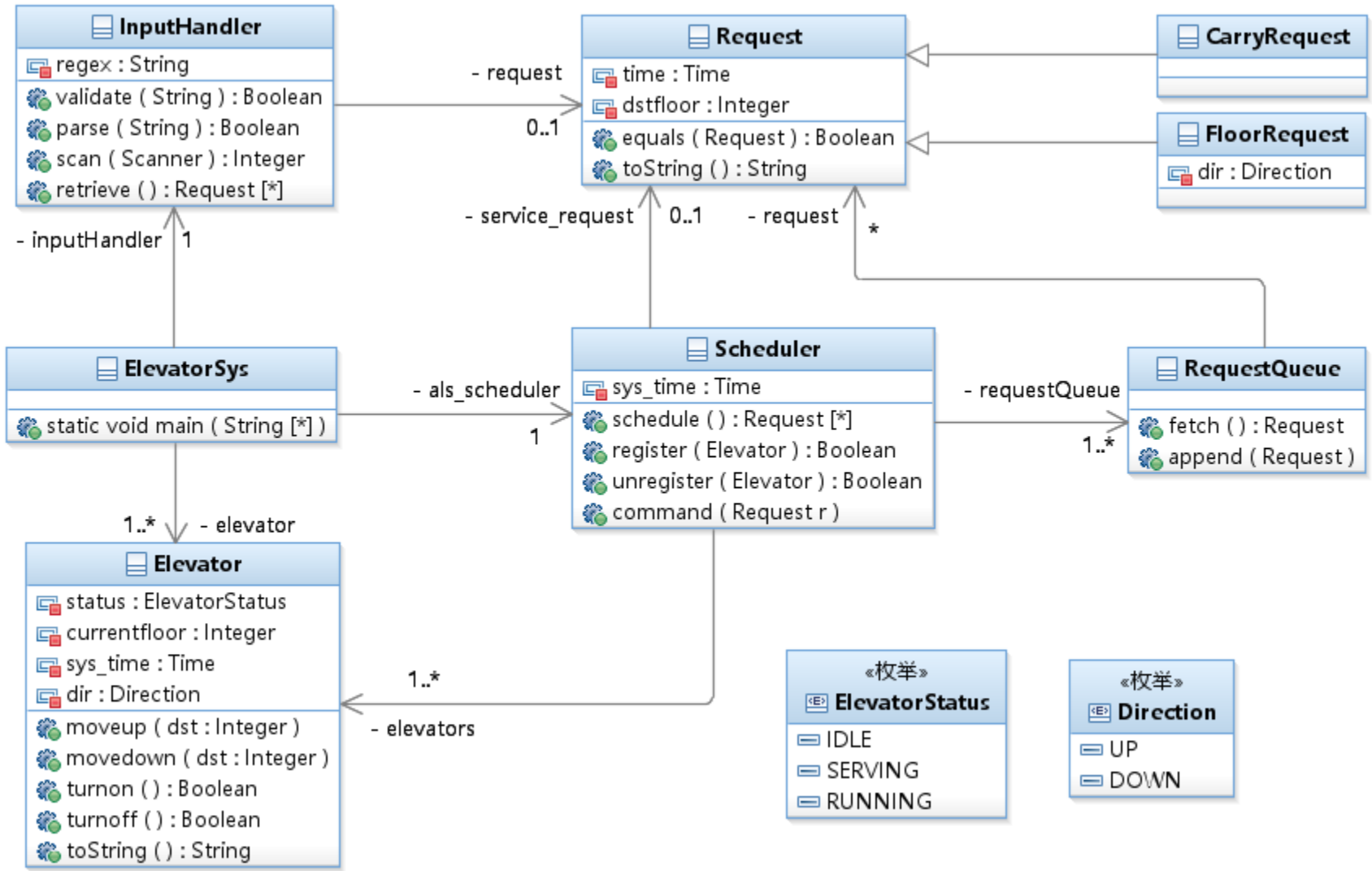
- 请求的物理表达和逻辑表达未能分开
  - 在Request(电梯请求)类中进行输入处理，导致其易变
  - 在电梯运动方法中进行输入处理
- 电梯类不能提供维护电梯状态的方法
  - 在其外部直接修改其属性状态
- 有方法hold住所有层次
  - 输入处理
  - 构造请求
  - 调度电梯
  - 修改电梯状态
- 直接在main方法中铺开进行输入的处理
- 逐个字符去做输入检查和处理

作业2共性问题总结

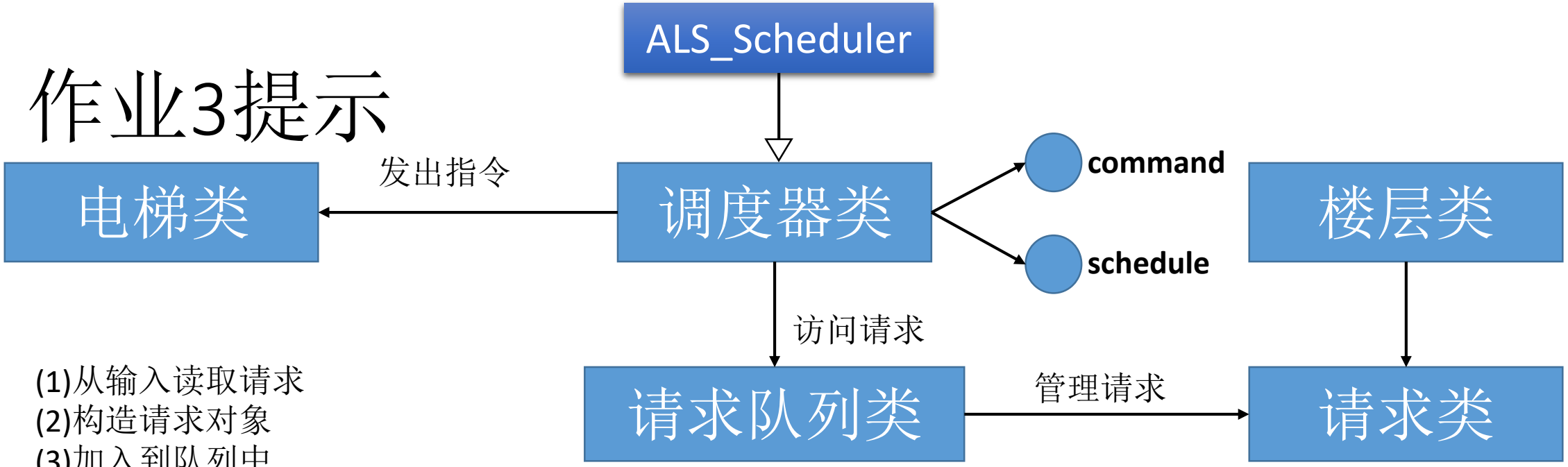
- 超长方法，多层逻辑嵌套
  - 职责分配的均衡性问题
- 类对属性的保护不够，直接通过reference就传递出外部
  - 居然还有直接声明为public访问！
- 土豪式的数组声明
- 频繁出现相似的代码块
- 方法职责不明
  - 如队列的获取请求方法进行调度电梯运行，并维护系统时间
  - 把输入处理、请求识别、调度等都封装在电梯类中

```
sign = new int[100000];
direction = new int[100000];
floor = new int[100000];
Des = new int[100000];
outfl = new int[100000];
time = new int[100000];
outT = new double[100000];
String[] str = new String[1000000];
```

作业2推荐设计


InputHandler
regex : String
validate (String) : Boolean
parse (String) : Boolean
scan (Scanner) : Integer
retrieve ( ) : Request [*]
- request
0..1
Request
time : Time
dstfloor : Integer
equals (Request) : Boolean
toString ( ) : String
CarryRequest
FloorRequest
dir : Direction
- inputHandler
1
- service_request
0..1
- request
*
ElevatorSys
static void main (String [*])
- als_scheduler
1
Scheduler
sys_time : Time
schedule ( ) : Request [*]
register (Elevator) : Boolean
unregister (Elevator) : Boolean
command (Request r)
- requestQueue
1..*
RequestQueue
fetch ( ) : Request
append (Request)
1..*
- elevator
Elevator
status : ElevatorStatus
currentfloor : Integer
sys_time : Time
dir : Direction
moveup (dst : Integer)
movedown (dst : Integer)
turnon ( ) : Boolean
turnoff ( ) : Boolean
toString ( ) : String
1..*
- elevators
«枚举»
ElevatorStatus
IDLE
SERVING
RUNNING
«枚举»
Direction
UP
DOWN

作业3提示



(1)从输入读取请求
(2)构造请求对象
(3)加入到队列中
(4)启动调度
(5)记录电梯对象对请求的响应

ALS_Schedule (A Little Smart Schedule)
(1)只要队列不为空，每次都取出队列头请求来调度
(2)电梯在运动过程中不能突然改变运动方向
(3)在调度电梯完成一个请求的过程中，可以让电梯完成“顺路”的楼层请求
(4)调度算法要确保电梯在当前运动方向上完成所有能完成的电梯内请求

关于“顺路”的楼层请求
设电梯当前状态为e=(e_n,sta,n)，即当前所处楼层为e_n，运动状态为sta，当前运动的目标为楼层n，则
(1) (e.sta = UP → 10>=e.n>e.e_n) || (e.sta = DOWN→1<=e.n<e.e_n)
(2)对于任意一个楼层请求r=(F_R,n,dir,t)，如果是电梯当前运动状态下的顺路请求，则一定有: (r.dir=e.sta) && ((r.dir=UP→(r.n <= e.n)&&(r.n>e.e_n)) || (r.dir=DOWN→(r.n>=e.n)&&(r.n<e.e_n)))
(3)对于任意一个电梯运载请求r=(E_R,n,t)，如果是电梯当前运动状态下的顺路请求，则一定有: (e.sta=UP→(r.n > e.e_n)) || (e.sta=DOWN→ (r.n<e.e_n))

作业

- 继续第二次作业的单电梯系统，基本功能要求保持不变
- 新增功能
  - 要求同时满足如下两个原则，一旦违反视为wrong
    - 电梯在完成对一个请求的响应之前，不能改变运动方向
    - 电梯在响应一个请求的过程中，如果有在当前运动方向上能够响应的请求，则必须进行响应，这样的请求称为“捎带响应请求”
  - 要求按照电梯实际响应情况，输出请求与响应序偶[request, response]
- 新增设计要求(设计要求也要进行检查，如果违背报incomplete型bug)
  - 使用继承机制，不要覆盖或删除第二次作业的傻瓜调度schedule代码，增加一个子类来重写schedule方法，注意子类方法要与父类方法有交互
  - 使用interface来归纳电梯的运动方法
  - 重写toString方法来获得电梯运行状态和时刻的观察

作业3训练要点分析

- 进一步理解调度逻辑及其设计
- 开始掌握如何使用继承构造逻辑处理层次
- 进一步训练类的均衡设计
- 开始了解针对状态的测试设计

作业3训练要点分析

- 进一步理解调度逻辑及其设计
  - FoolSchedule显然对电梯这个紧缺资源的利用率不高
  - 提高资源利用率是调度算法设计的核心目标
    - “顺路捎带”：在去响应一个请求的路途中可以把资源共享给顺路可完成的请求
    - 操作系统调度中哪个情况和这个类似？
  - 谁来负责找可以捎带的请求？
  - 如何找可以捎带的请求？
  - ➔需要几个类的协同才能完成

作业3训练要点分析

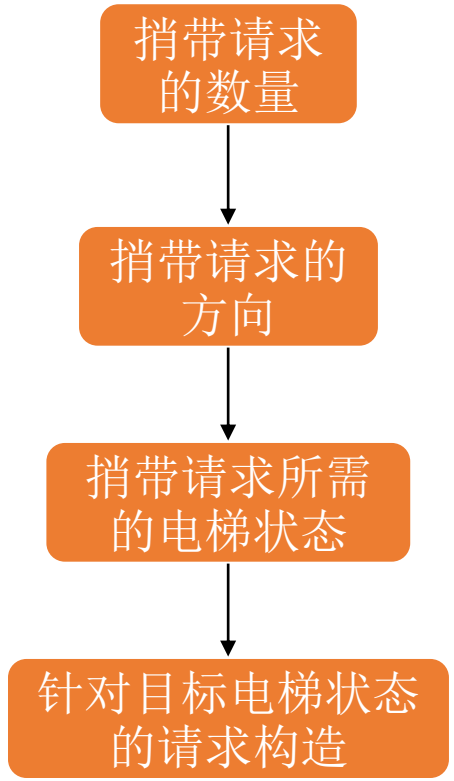
- 开始掌握如何使用继承构造逻辑处理层次
  - 继承不只是重用，构造类型层次更重要
  - Scheduler
  - 两种方法相结合
    - 自顶而下：从需求分析出发
    - 自底向上：针对已有设计进行提炼

作业3训练要点分析

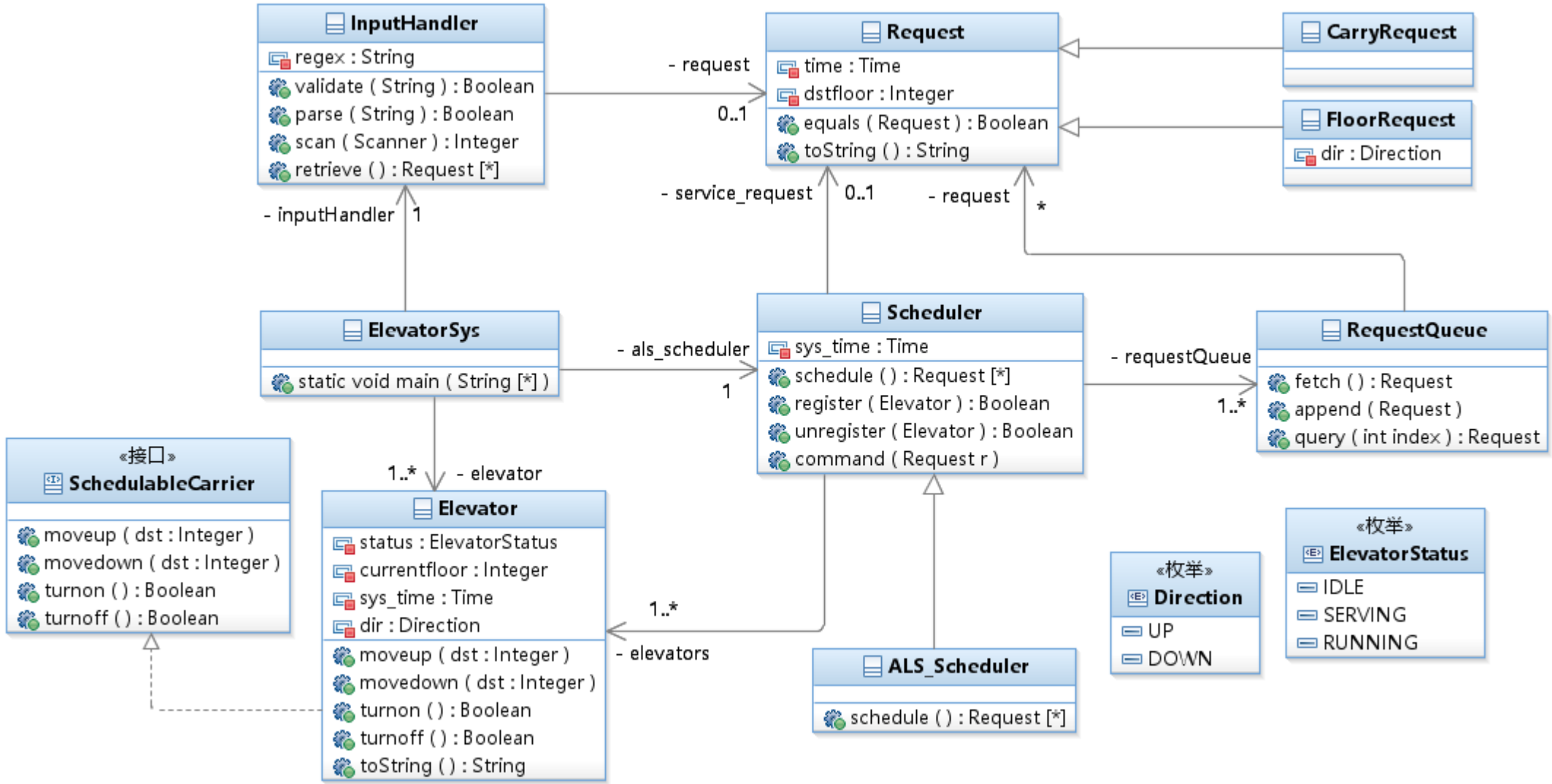
- 进一步训练类的均衡设计
  - 每个类只做自己该做的事情
  - 电梯做什么？
  - 调度器做什么？
  - 请求队列做什么？
  - 请求做什么？

作业3训练要点分析

- 开始了解基于状态的测试设计
  - 电梯的运行受调度器控制
  - 调度器根据队列中的请求来进行调度
    - 请求的楼层、方向及时间顺序
    - 是否可以捎带
  - 请求队列的状态



作业3推荐设计


InputHandler
regex : String
validate (String) : Boolean
parse (String) : Boolean
scan (Scanner) : Integer
retrieve ( ) : Request [*]
- request
0..1
Request
time : Time
dstfloor : Integer
equals (Request) : Boolean
toString ( ) : String
CarryRequest
FloorRequest
dir : Direction
- inputHandler
1
- service_request
0..1
- request
*
ElevatorSys
static void main (String [*])
- als_scheduler
1
Scheduler
sys_time : Time
schedule ( ) : Request [*]
register (Elevator) : Boolean
unregister (Elevator) : Boolean
command (Request r)
- requestQueue
1..*
RequestQueue
fetch ( ) : Request
append (Request)
query (int index) : Request
«接口»
SchedulableCarrier
moveup (dst : Integer)
movedown (dst : Integer)
turnon ( ) : Boolean
turnoff ( ) : Boolean
1..*
- elevator
Elevator
status : ElevatorStatus
currentfloor : Integer
sys_time : Time
dir : Direction
moveup (dst : Integer)
movedown (dst : Integer)
turnon ( ) : Boolean
turnoff ( ) : Boolean
toString ( ) : String
1..*
- elevators
ALS_Scheduler
schedule ( ) : Request [*]
«枚举»
Direction
UP
DOWN
«枚举»
ElevatorStatus
IDLE
SERVING
RUNNING

作业

- 教学模块问卷调查：了解同学们的学习效果和相关建议
  - cnblogs上发布，匿名作答，但要求所有人都要参与。
- 总结性博客作业
  - 针对所讲授内容、自己发现别人的问题、3次作业被发现的bug(包括公测)、分析课所介绍的共性问题
  - (1)基于度量来分析自己的程序结构
    - 度量类的属性个数、方法个数、每个方法规模、每个方法的控制分支数目、类总代码规模
    - 计算经典的OO度量(可使用工具)，分析类的内聚和相互间的耦合情况
    - 画出自己作业的类图，并自我点评优点和缺点
      - 使用UML工具，网上有很多免费资源

http://metrics.sourceforge.net/

http://fileadmin.cs.lth.se/cs/education/edan70/CompilerProjects/2014/Reports/TervalampiOlssonLacerda-OOMetrics.pdf

作业

- (2)分析自己程序的bug
  - 分析未通过的公测用例和被互测发现的bug：特征、问题所在的类和方法
  - 关联分析bug位置与设计结构之间的相关性
  - 从分类树角度分析程序在设计上的问题
- (3)分析自己发现别人程序bug所采用的策略
  - 列出自己所采取的测试策略及有效性，并特别指出是否结合被测程序的代码设计结构来设计测试用例
- (4) 心得体会
  - 关于设计问题、bug等方面的心得体会
- 如何评价
  - 鼓励同学们互相阅读和学习，并积极点评
  - 在cnblogs上发布，一直伴随你，会有很多人看到你的博客，只要精彩，定有很多转发。